TOPAS, TOPAS-Academic

What's New

Version 7

Alan A Coelho www.topas-academic.net

Bruker–AXS www.bruker-axs.de

September 7, 2020

Ab initio solution of proteins at atomic resolution, Fast simultaneous refinement of 1000s of data sets, Amazon EC2 cloud computing, PDF Generation, Deconvolution, Capillary aberration, LP-Search, Sine Transform, DPI awareness.

1 … UNSEEN IMPROVEMENTS

Many improvements are unseen; some of these include:

- Improvements to Marqaurdt constant determination.
- Improvements to the conjugate gradient solution rotuine.
- Improvements to the BFGS method.
- More speed improvements.
-

1.1 …… Referencing

- Coelho, A. A. (2018). *J. Appl. Cryst.* 51, https://doi.org/10.1107/S1600576717017988, "*TOPAS & TOPAS-Academic: An optimization program integrating computer algebra and crystallographic objects written in c++*"

2 … AMAZON EC2 CLOUD COMPUTING – TC-CLOUD

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

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

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

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

2.1 …… Operation

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

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

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

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

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

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

2.2 ..... Pre-requisites

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

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

2.3 ..... Pricing of AWS cloud resources

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

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

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

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

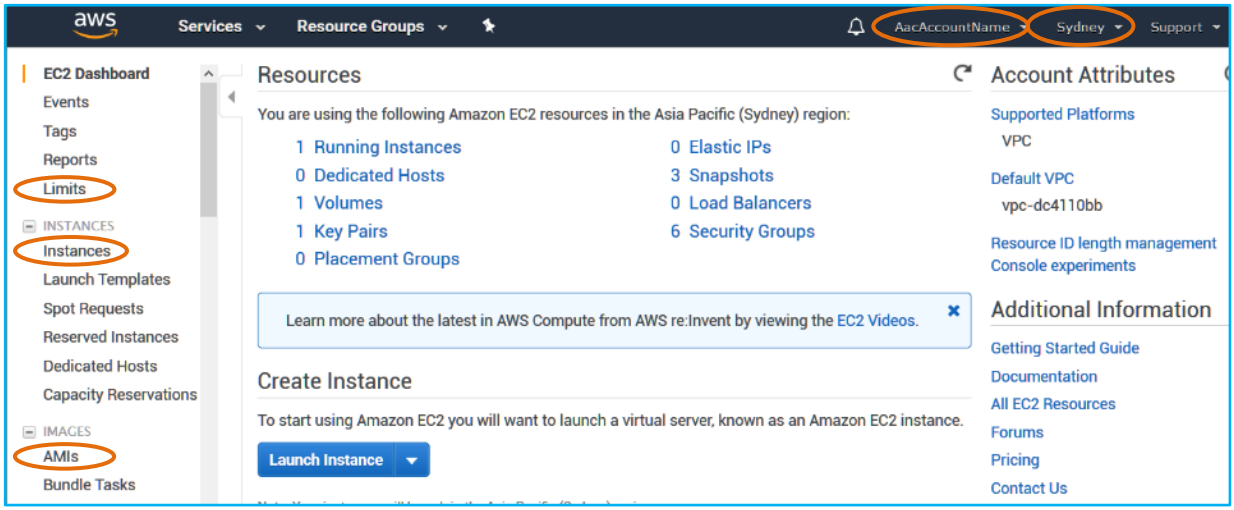
Each TV-VM is a Linux VM; it comes with an 8 Gbyte EBS volume which stores TC-Cloud and the operating system. EBS volumes are relatively inexpensive at 0.125 USD per Gbyte per month, or

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

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

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

AWS includes a comprehensive browser dashboard called EC2 Dashboard https://ap-southeast-2.console.aws.amazon.com/ec2/. In the case of running TC-Cloud, the dashboard is primarily used to create TC-VMs from TC-AMI as well as deleting files created on the S3 cloud storage. The rest of TC-Cloud operations are performed from TA.EXE. The important parts of EC2 Dashboard are circled in the following:



Note: AWS web screens may change due to improvements etc...; the general operation however should remain the same. Clicking on the *Account* (circled on the top) brings up account options which includes real time billing information (AWS cloud costs). Also, on the top is the AWS region being operated on. AWS operates on a regional basis; regions chosen should be in close geographical proximity to the local computer. This reduces response times and data transfer costs. TC-VMs are created by clicking on *AMIs*. Once created, details of TC-VMs for the selected region can be viewed by clicking on *Instances*. AWS limits the number of VMs available to 20 on most VM types; request for increasing this number can be made from the circled 'Limits' item. The author had no trouble getting regular access to 500 spot instance VMs.

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

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

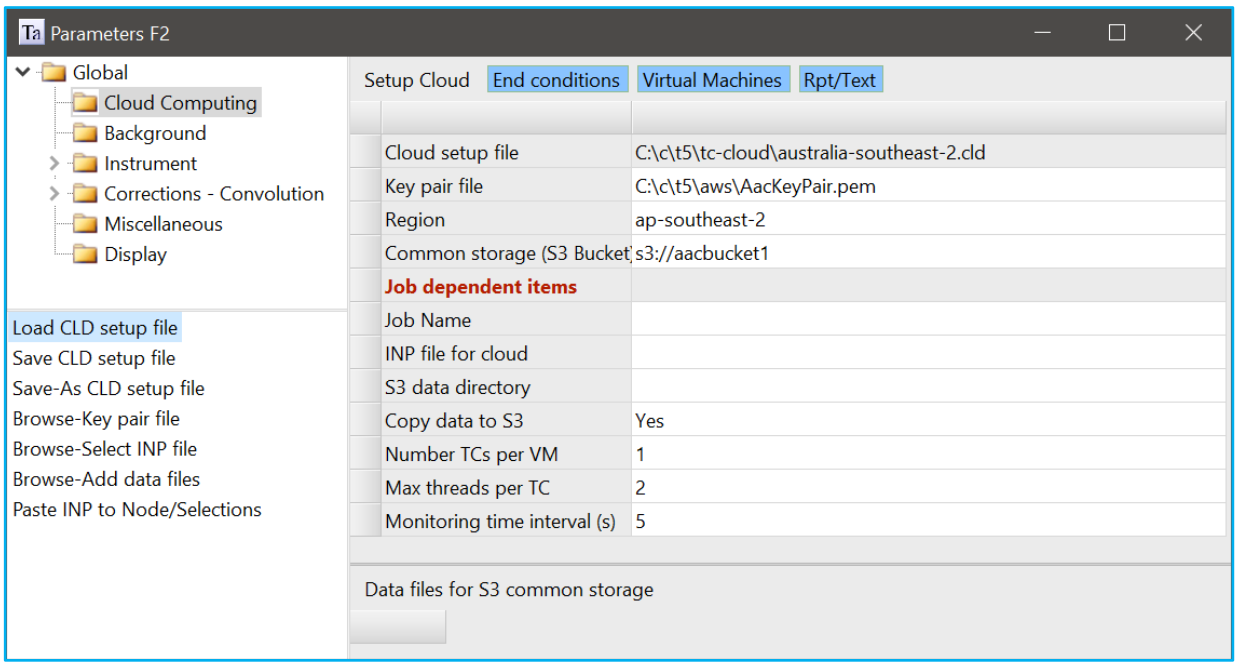
https://docs.aws.amazon.com/cli/latest/userguide/install-windows.html.

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

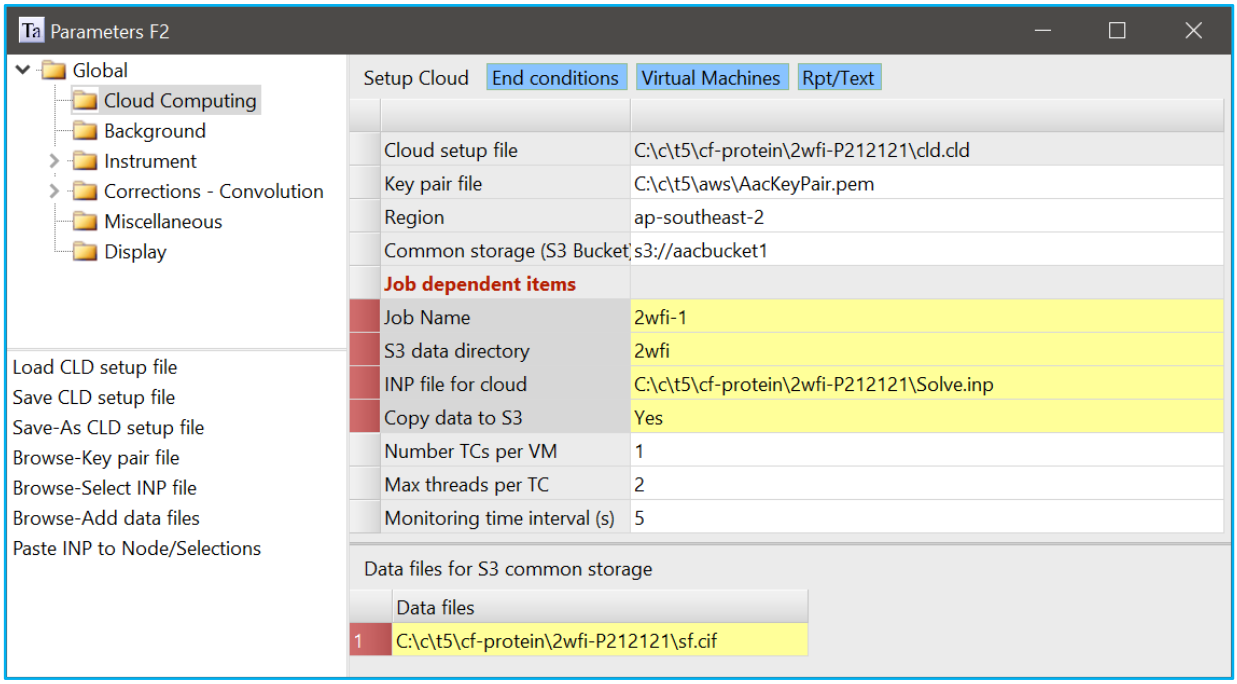
After the preliminary setting up and testing of an INP file with TA.EXE on the local computer, the INP file can be fed to AWS for parallel operation on many VMs. Summing up the process we have:

1) Set up INP file and ensure it runs as expected on TA.EXE on the local computer.
2) Create a small number of VMs (3 for example) and ensure that the INP file runs as expected on the VMs.
3) Create many more VMs (user determined) and run the INP file on the VMs.

Stage-1 is normal TOPAS operation. Stage-2 involves creating a *job* (*.CLD files) from the 'Setup Cloud' tab in the GUI. Before creating a job its best to create a template that can be used for all jobs in the AWS region. Enter your *'Key pair file'*, the AWS *Region* being used and your *S3* bucket name details in the *Setup Cloud* tab; it should look something like:



Save the details using '*Save-As CLD setup file*' to a file. Load this file when creating other CLD files. To run a job then enter the rest of the setup details; an example is:

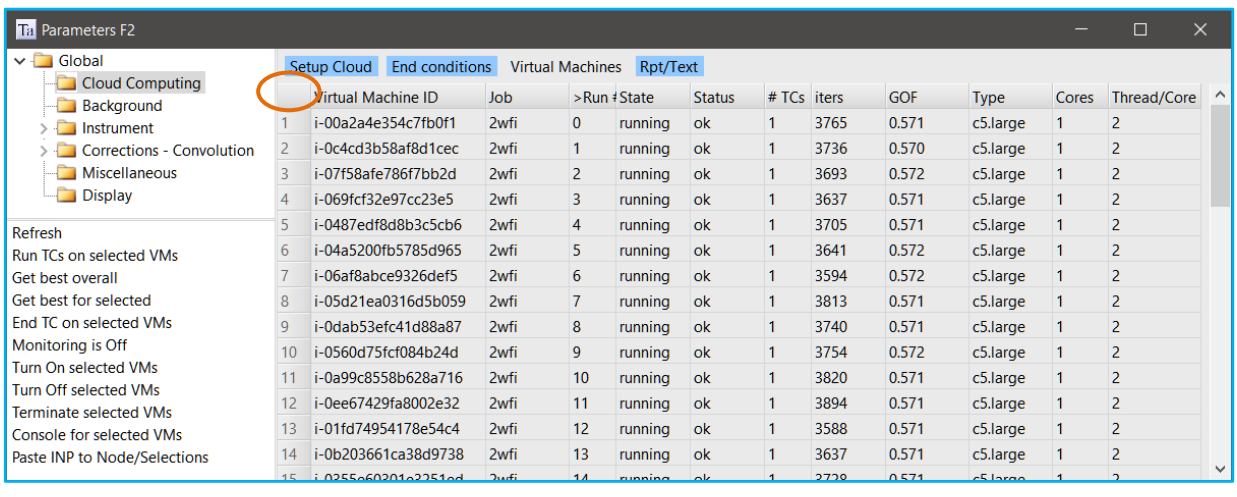


The highlighted lines require input to create a job. This input comprises the INP file to be run on the Cloud as well as the necessary data files. In the above example the INP file is placed in the S3 job directory called 2wfi-1 and the data file is placed in the S3 directory called 2wfi. S3 will therefore contain the following two directories:

s3://aacbucket1/swfi-1

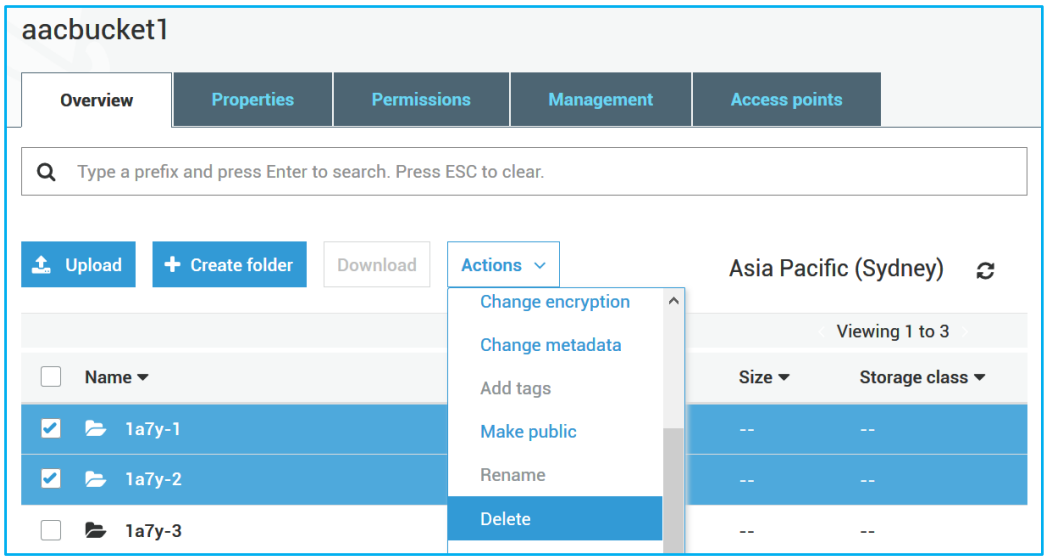
s3://aacbucket1/swfi

The INP file as well as other communication files are copied to the job directory, 2wfi-1 in this case. The name of the INP file on S3 is changed to in.inp; in.inp is used in the retrieval of out from the VMs; it is unchanged during Cloud operation and it can be also viewed as a backup for the job. Each run on the Cloud requires a unique job name; an exception is thrown otherwise. Many jobs however, can use the same S3 data directory. In cases where many jobs are run sequentially, each using the same data files, then the '*Copy data to S3*' option can be set to No after the first job; this speeds up processing as copying large data files over the internet can be slow. CLD files contain information necessary for launching the INP file on the cloud. Once the information is entered, it becomes possible to view the created VMs in the 'Virtual Machines' tab, or:



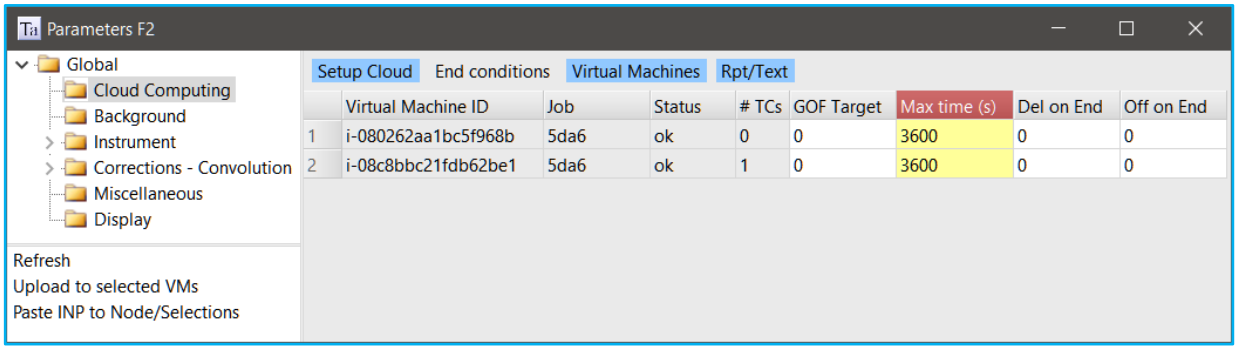
Data can be displayed in sorted order by double clicking on column headings. To launch the INP file on a VM then select the VM and click *'Run TC on selected VMs'*. To select all VMs then click on the empty rectangle circled. Only VMs with an *ok* Status can be launched. If a selected VM is *starting* or *pending* then *Status* will not be *ok*. The number of TCs running on each VM (typically one) is shown in the *# TCs* column. This data as well as other VM details maybe out-of-date; to show the latest then click on the *Refresh* option. The *iters* column shows the total number of refinement iterations executed on the respective VM; this number supplies a means of determining if a VM is running in an expected manner. For example, if *iters* has stopped increasing in an expected manner and *#TCs* is not zero then the running TCs have stopped operating in an expected manner.

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



2.7 ..... Terminating/Stopping TC-VMs and tc-mon.a

Terminating or stopping TC-VMs reduces AWS fees. TC-VMs can be automatically stopped or terminated depending on *'End conditions'*, or:



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

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

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

Max time (s) = 10 60 60 = 10 hrs of running

GOF Target = 10, Off_on_End = 1, Del_on_end = 1

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

Max time (s) = 0

GOF_Target = 10, Off_on_End = 0, Del_on_end = 0

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

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

In addition to the terminating/stopping criteria of section 2.7, VMs are automatically powered off (stopped but not terminated) after 100 minutes of TC-Cloud inactivity including inactivity on VM start-up. The net effect is that VMs are stopped after 100 minutes of TC-cloud not being run. Situations where 100 minutes of inactivity is possible include internet-down situations as well as users forgetting to power-off or terminate VMs. For example, the fee incurred for forgetting to turn off 100 spot instance VMs would be ~3.40 USD. Typical usage comprises turning on 500 VMs and running many jobs interactively to analyse data. The VMs are not turned off during the running of these jobs.

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

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

'INP File for cloud' = C:\DATA\PBSO4.INP

'Get best overall' places output in C:\DATA\PBSO4 -1.INP

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

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

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

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

2.11 .... Random number generator automatically seeded

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

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

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

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

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

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

```
}
```

2.13 .... 'Setup Cloud' details

Cloud setup file

Name of file containing cloud details for a job.

Key pair file

Name of file containing encrypted login information, see:

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

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

Region

Geographical region where VMs reside.

S3 Bucket

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

s3://my-bucket/job-1

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

Job Name

Name of job. Job names cannot contain spaces.

S3 data directory

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

INP file for cloud

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

Number TCs per VM

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

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

Max threads per TC

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

Max_threads_per_TC = (Virtual Cores) / Number_TCs_per_VM

Monitoring time interval (s)

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

2.14 .... 'Virtual Machines' tab options

Refresh

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

Run TC on selected VMs

Launches TC-Cloud on selected VMs.

Get best overall

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

Get best for selected

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

End TC on selected VMs

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

Monitoring is On/Off

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

Turn On selected VMs

Turns selected VMs On.

Turn Off selected VMs

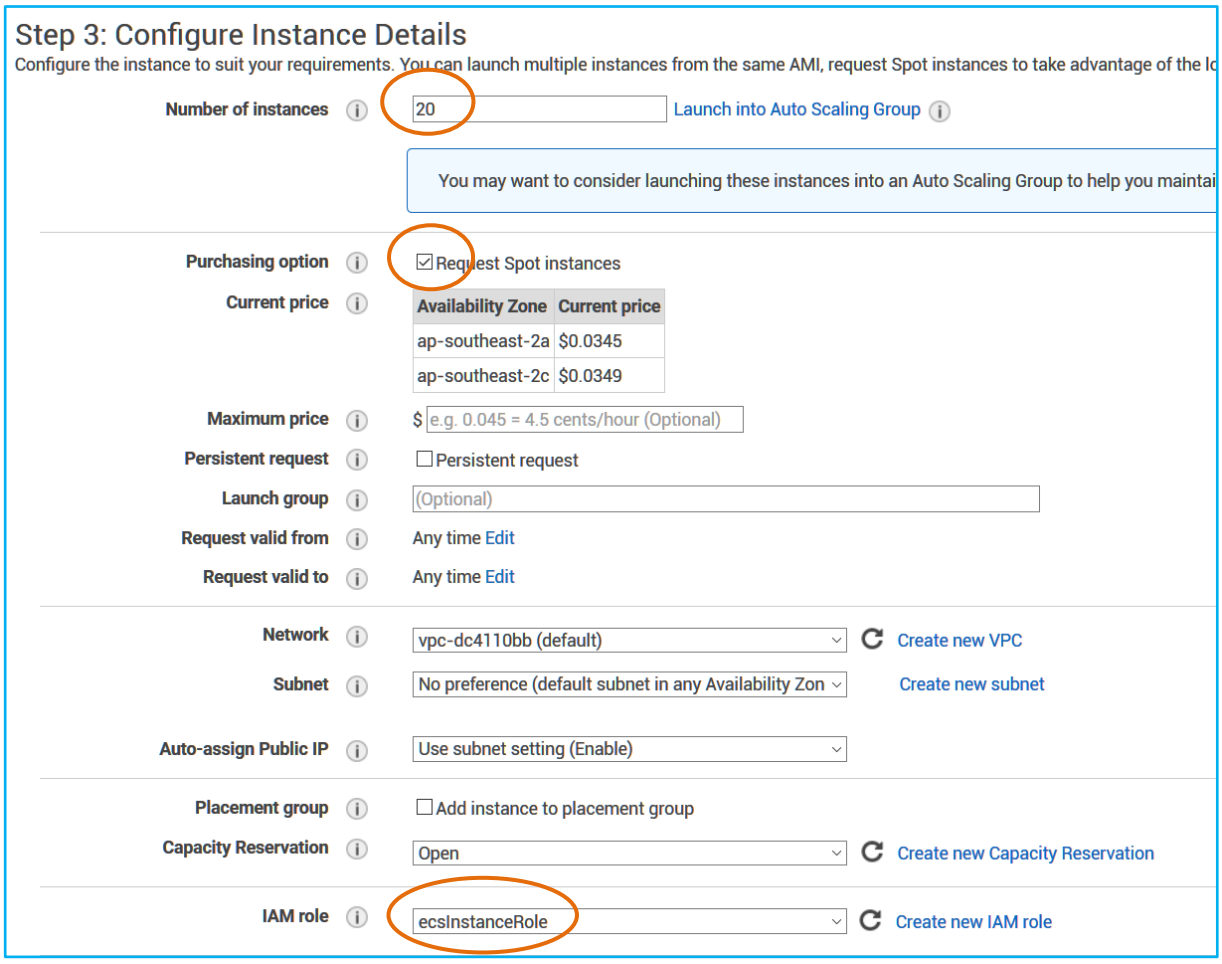
Turns selected VMs Off.

Console for selected VMs

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

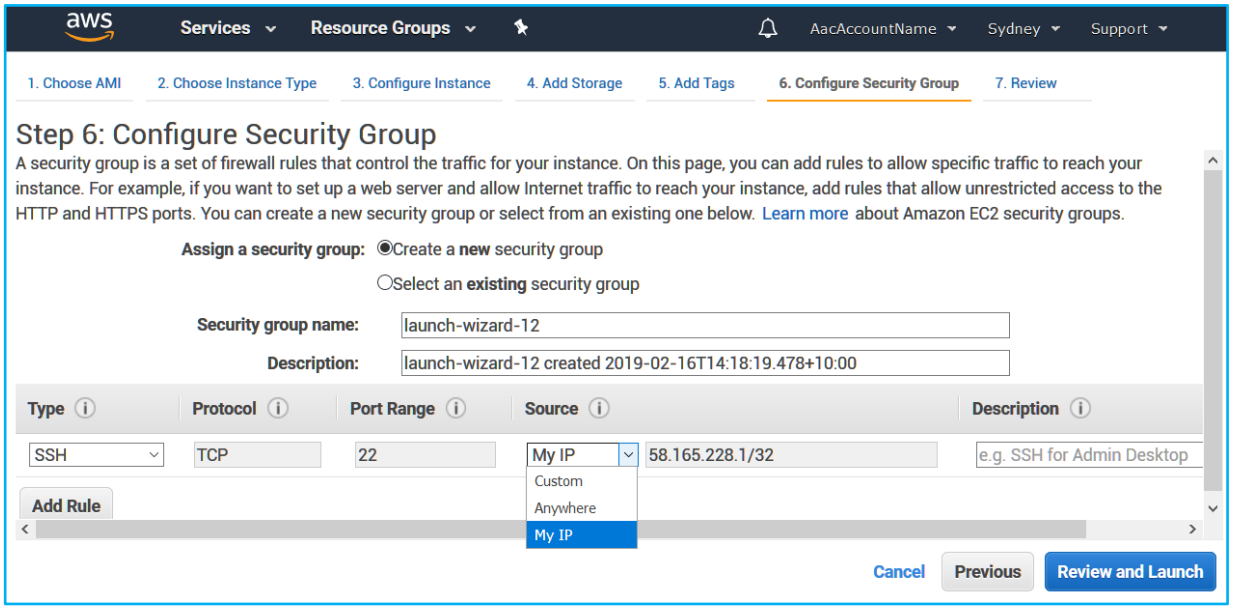
2.15 .... Creating TC-VMs – Spot Instances

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



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

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



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

2.16 .... Choosing the optimum VM type

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

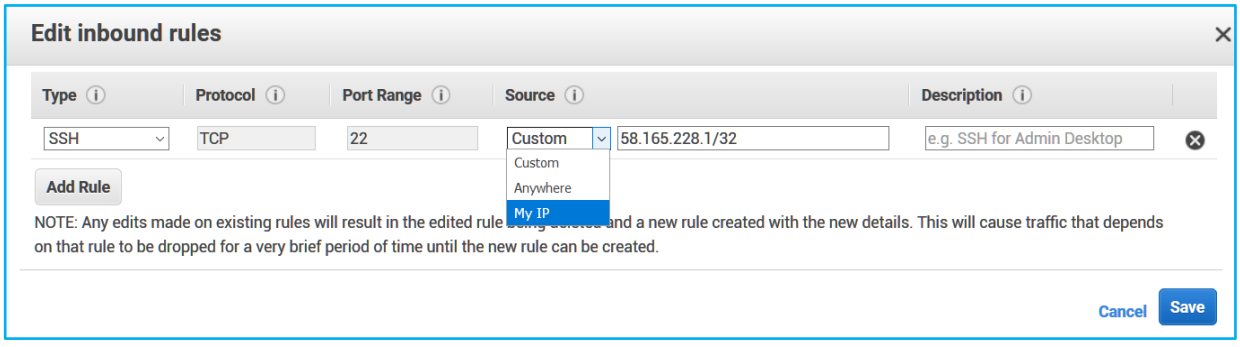
Note, TOPAS is threaded to a large extent, however, an excessive number of threads could slow down operation. For example, the large VM type of *c5.18xlarge* operating on the TEST_EXAMPLE\SINGLE-CRYSTAL\PN_02_2-ADPS.INP (3970 parameters) produces the following as a function of number of threads:

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

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

2.17 .... Unable to connect to TC-VMs after local computer restart

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



3 ... PROTEIN REFINEMENT

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

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

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

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

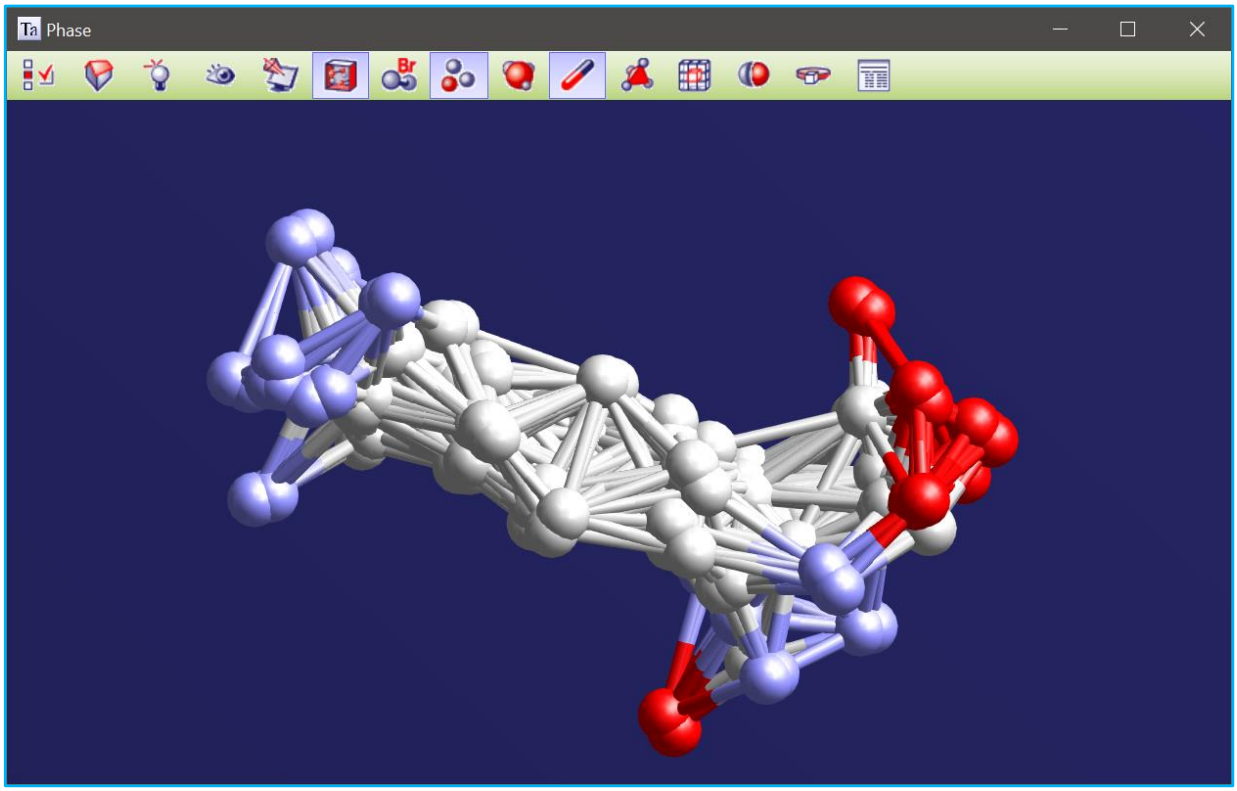
A rigid body is generated for each residue with coordinates set relative to its geometric center. Refinement can proceed on the generated INP text by setting the file name of *xdd_scr* to the name of the structure factor file 2PVB-SF.CIF, also downloaded from https://www.rcsb.org/. Running 2PVB\GEN.INP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

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

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

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

Running example 2PVB\MATCH.INP produces the following showing overlay of LYS residues:

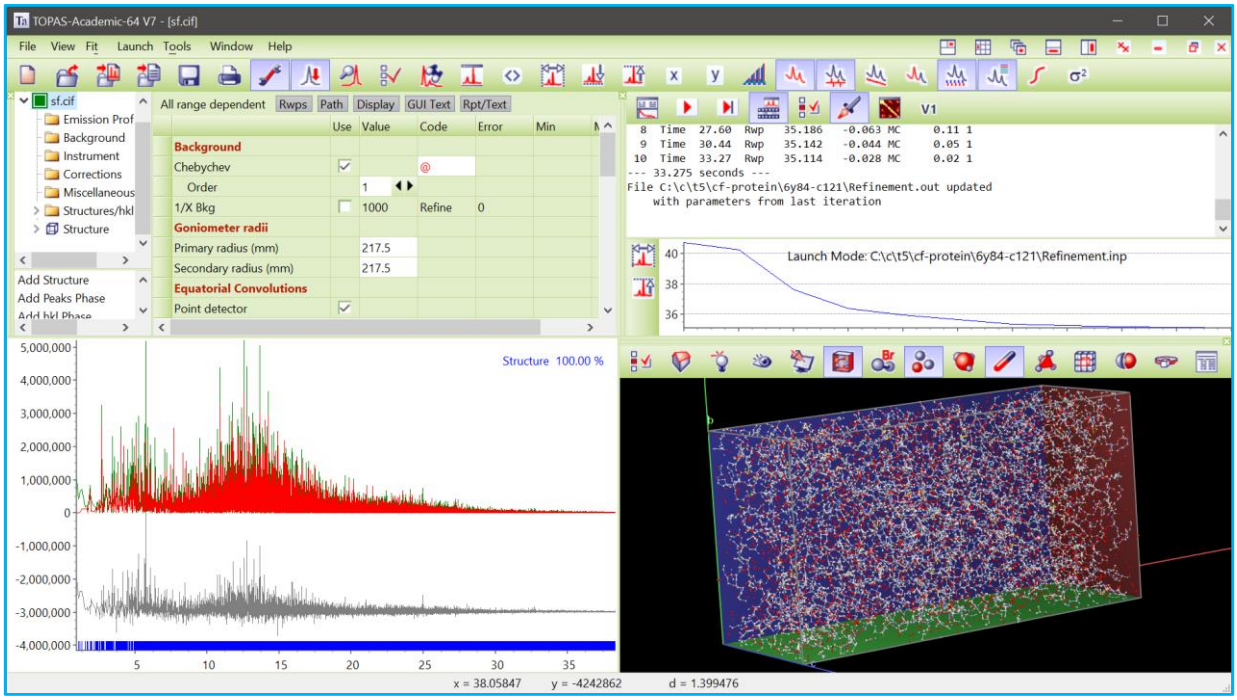


3.2 ..... Protein Refinement, 6y84, SARS-CoV-2 main protease

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

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

After refinement, the INP file can be updated with the structure generated from the CIF file. Refining on the updated INP file gives:



The refinement comprised 50348 unique reflections and 1826 parameters and the time to convergences was 33s on a laptop with all graphics operational. Restrains/constrains can of course be added.

4 … SOLVING PROTEINS AT ATOMIC RESOLUTION

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

Macros in CHARGE_FLIPPING.INC

Examples

CF-PROTEIN\

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

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

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

- 4LZT-P1\2-ATOMS.INP

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

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

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

Table 4-1 show difficult benchmark structures, as listed by Elser *et al.* (2017) and Burla *et al.* (2011), that have been solved *ab initio*; see corresponding SOLVE.INP files for details. It is best to do preliminary investigations on the local computer (non-Cloud) to determine which strategy might work best. Once a strategy is chosen, INP files can be fed to the Cloud for rapid structure solution. Up to 500 spot instance Virtual Machines (VMs) are easily obtained on the Amazon AWS system in Australia at a cost of ~0.035 USD cents per VM per hour, or, 3.40 USD per hour for 100 machines. These prices are Amazon AWS dependent. Prices are shown prior to the creation of the VMs. The times shown in Table 4-1 can be easily doubled when one considers the preliminary analysis taken to arrive at the appropriate strategy. Typically, strategies are tried on the local computer before migrating the problem to the Cloud. Also, the structure solution process is normally halted after the first solution is found; for the investigative purposes, however, the structures in Table 4-1 were each solved at least 5 times. The two strategies mentioned in Table 4-1 are:

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

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

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

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

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

Table 4-1. *Ab initio* structure solution strategies. Time indicates time to solution on average. Each structure was solved at least 5 times. *Num_VMs* greater than 8 refers to the number of VMs used on the Cloud; *Num_VMs*=9 corresponds to an 8 core local computer (a laptop) . Cost corresponds to the average Cloud cost to a solution using the strategy indicated.

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

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

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

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

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

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

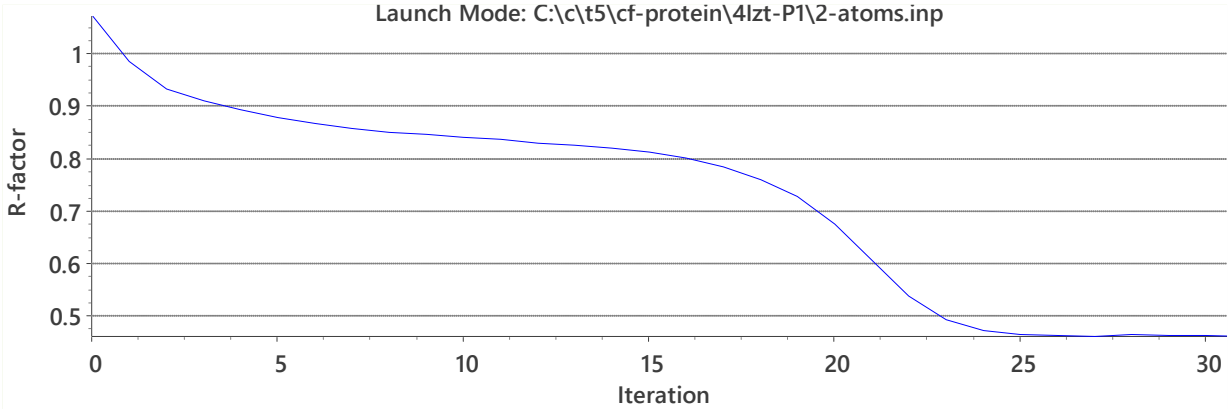
- Try the simple S0 strategy first for number of atoms less than about 300.
- If a heavy atom is present, then try S0.
- Inspect the ED graphically; if it does not show distinct atoms after a few iterations then change strategy.
- Use S1 for large difficult structures.
- Try the tangent formula when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

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

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

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

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

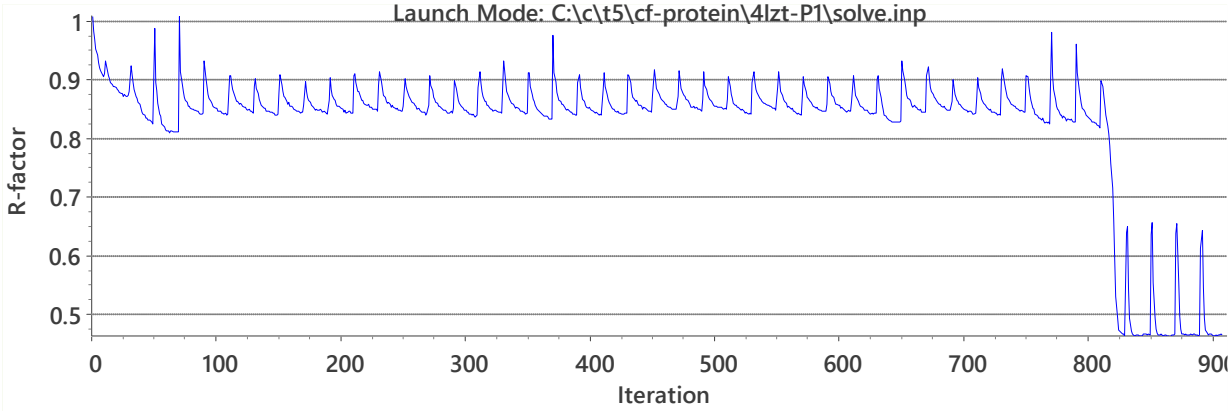


In fact, any two of the five highest peaks produce similar R-factor plots. However, these optimal ED peak positions are unknown. The strategy that works therefore involves picking an atom

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

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

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



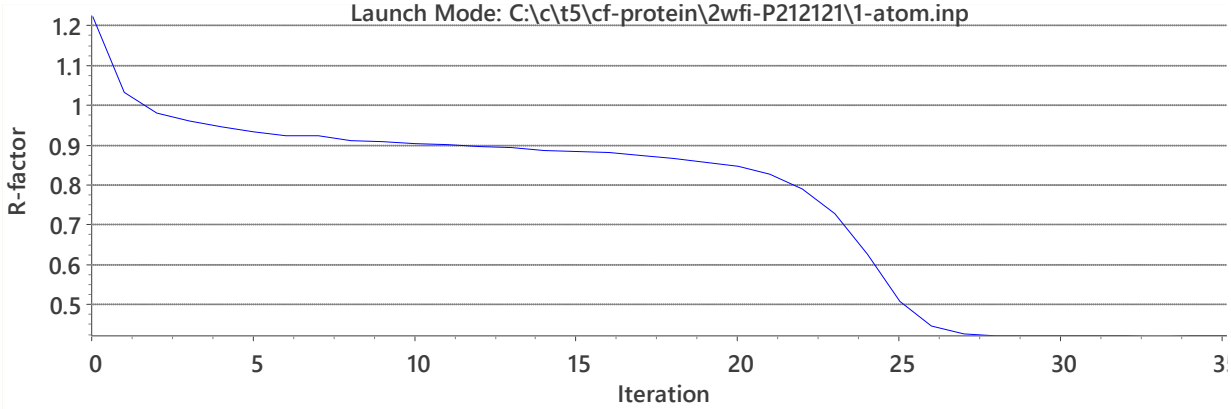
4.2 ..... Solution of non-triclinic lattices using a known atomic position

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

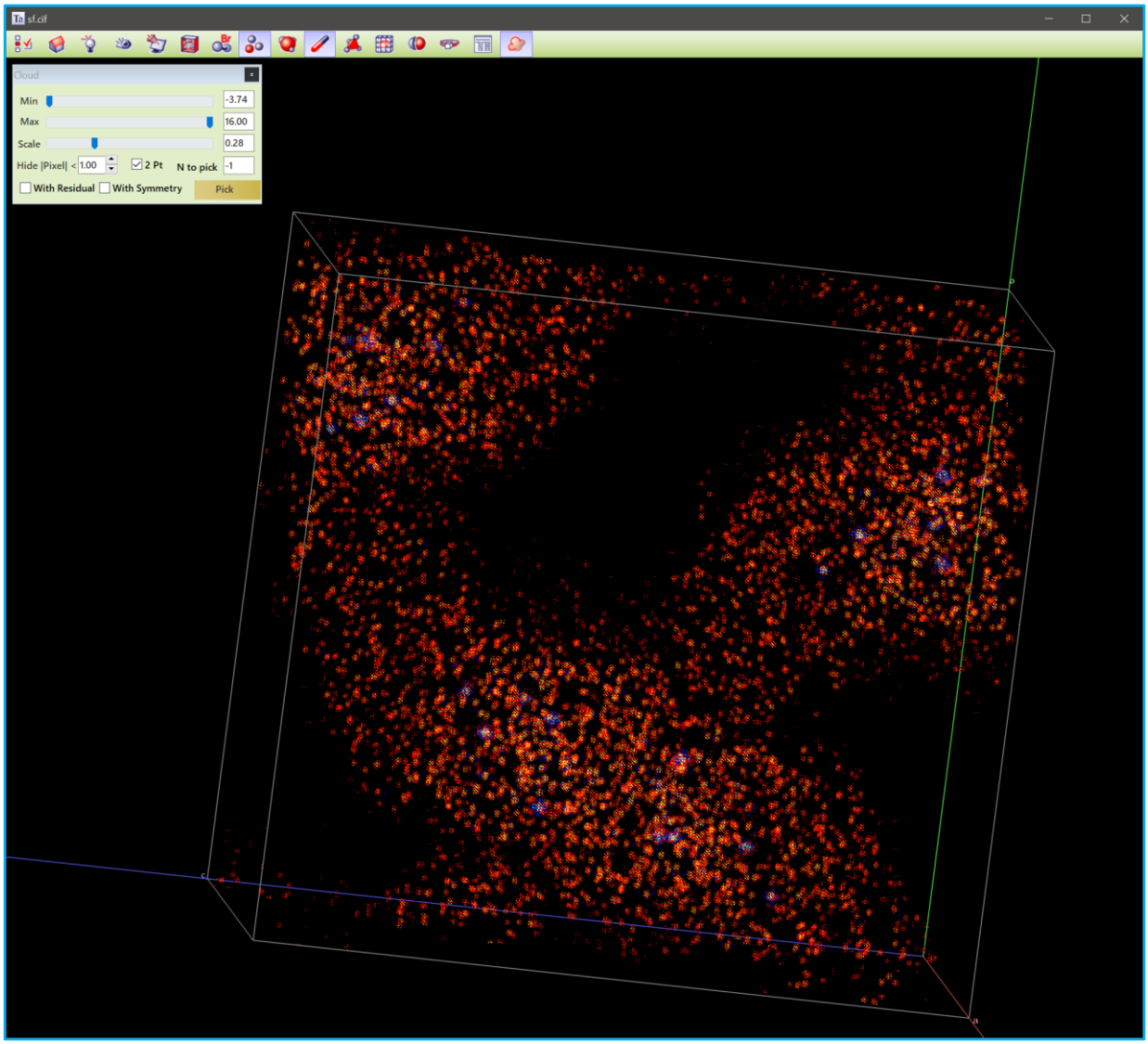
```
charge_flipping
    cf_hkl_file sf.cif ' Structure fact file from PDB
    space_group P212121
    a  37.544  b  65.144  c  69.680
    fraction_reflections_weak 0.5
       add_to_phases_of_weak_reflections = Rand(-180, 180);
    fraction_density_to_flip 0.97
       scale_flipped 0.2
    symmetry_obey_0_to_1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

```
        }
```

The *x*, *y*, *z* coordinates of *insert_atom* can be in Cartesian coordinates using the *in_cartesian* keyword at the *insert_atoms* level. The use of *symmetry_obey_0_to_1* often assists in solution determination for non-triclinic structures. 2WFI can be solved *ab initio*, however it can be easily solved if the position of one atom was known as seen by tunning 2WFI-P212121\1-ATOM.INP; it gives and R-factor plot that looks like:



The OpenGL plot shows the solution as follows:



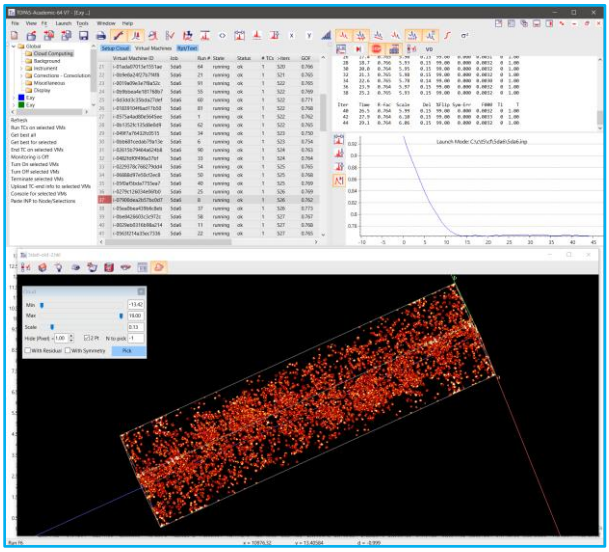
Using any one of the first six highest optimal ED peaks results in a solution. Many structures can be solved from knowing the position of just one atom. 1-ATOM.INP files, similar-to the 2wfi case, are given for 1ctj, 2pvb, 1c75, 5da6, 1cku, 2wfi.

4.3 ..... Ab initio solution of 5da6 in space group $R3_2$

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

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

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



5 … FAST SIMULTANEOUS REFINEMENT OF 1000S OF PATTERNS

[*str*…] [*peak_buffer_similar_tag* !E] [*hkl_similar_tag* !E]	Example TEST_EXAMPLES\1000S-OF-PATERNS\FIT.INP

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

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

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

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

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

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

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

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

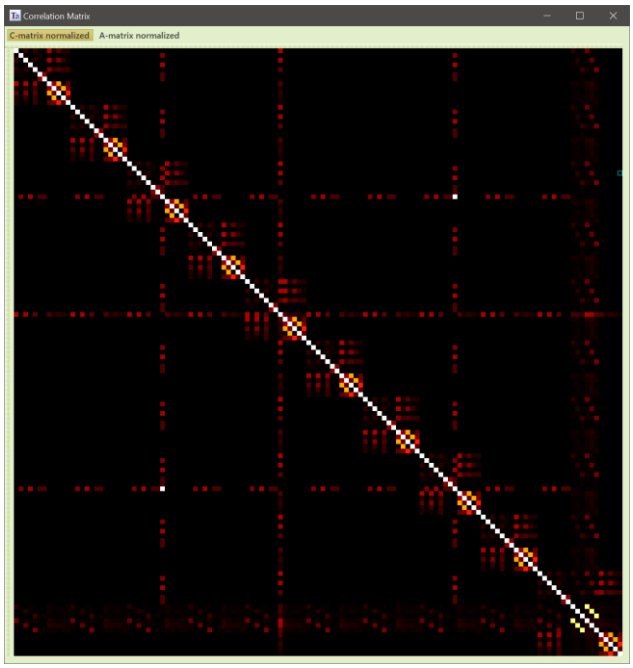
- Phases with the same peaks-buffer can have different *fit_obj*(s), *bkg*(s), *th2_offset*(s), *scale_phase_X*(s), *scale_pks*(s) or *pk_xo*(s).
- Phase with the same structural parameters (same sites) can have different *fit_obj*(s), *bkg*(s), *th2_offset*(s), *scale_phase_X*(s), *scale_pks*(s), *pk_xo*(s)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

5.1 ..... Example refinement of 1000s of patterns

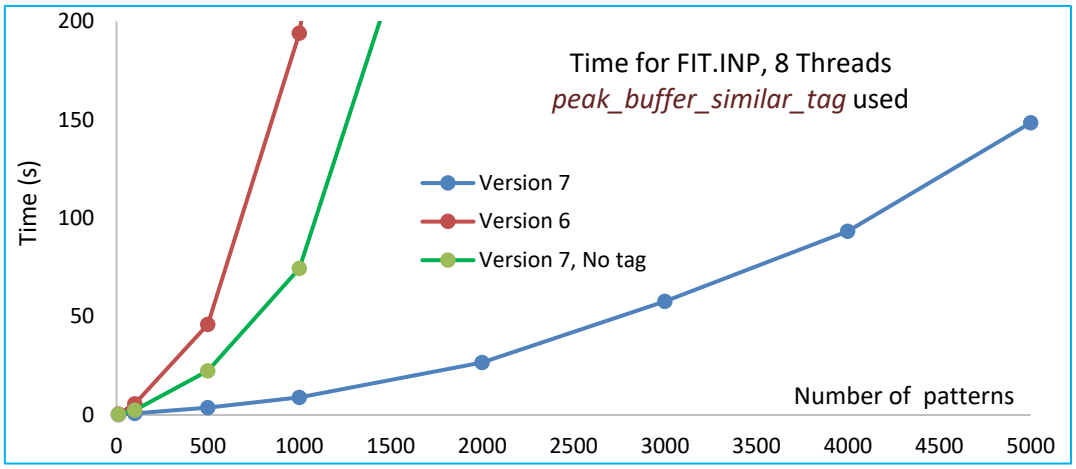
First, it may be best to set the number of threads in the file MAXNUMTHREADS.TXT to the number of physical CPU cores in your computer. FIT.INP creates a test pattern with 4001 data points when "*#define* CREATE__" is used. When "*#define* CREATE__" is commented out then refinement proceeds on many patterns,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

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

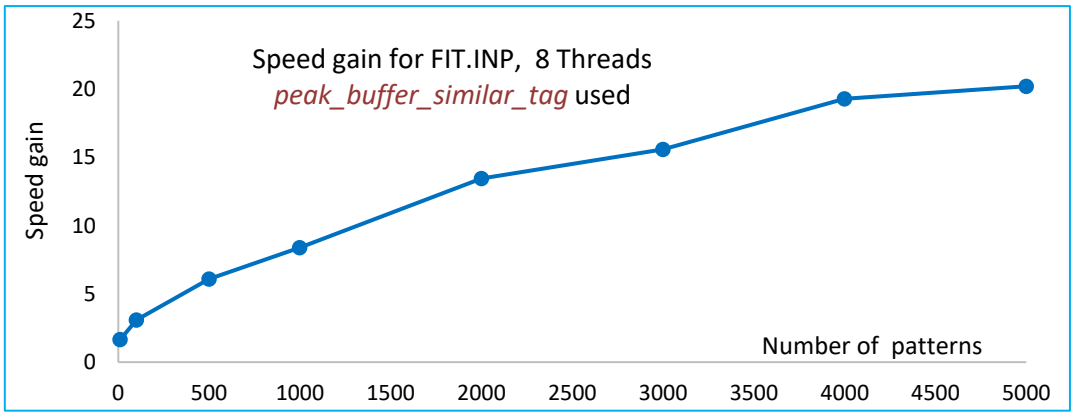
Global to all patterns are three CS_L and three CS_G parameters. This results in 12009 independent parameters for 1000 patterns, or 60009 independent parameters for 5000 patterns. The **A** matrix is sparse as seen in the (right) for 10 patterns.



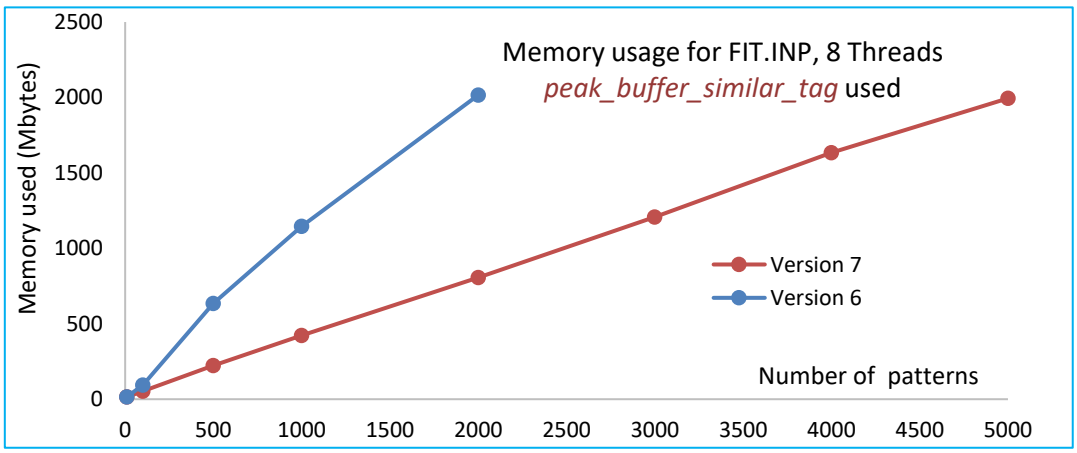
The following plots the time taken to perform 30 iterations as a function of Number-of-patterns for Version 6 and Version 7 with the use of *peak_buffer_similar_tag*; and additionally, for Version 7 without the use of *peak_buffer_similar_tag*.



Seen is the very large reduction in refinement times when *peak_buffer_similar_tag* is used. Version 6 is considerably slower due to threading at the *str* level rather than the *xdd* level; Version 7 with *peak_buffer_similar_tag* is in fact well-over 120 times faster than Version 6. For Version 7 only, the following shows that the speed gain of Time with *peak_buffer_similar_tag* divided by Time without *peak_buffer_similar_tag* increases with Number-of-patterns.



The following shows the reduction in memory usage for Version 7 compared to Version 6.



The above refinements for FIT.INP had three unique structures across all patterns. If the *beq* parameters of the three structures were refined independently then there would be 3*Number-of-

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

6 ... DECONVOLUTION

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

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

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

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

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

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

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

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

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

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

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

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

pen_weight over-rides the default; the default works but with slower convergence. Note, both the peak intensity and Bkg penalties are *Yobs* scale invariant where scaling of *Yobs* does not change the magnitude of the penalties relative to χ^2_0. Yobs_at is a new function that returns the value of *Yobs* at x. w_min in the *Deconvolution_Bkg_Penalty* macro allows for the setting of the expected minimum of *Yobs*Bkg*; a value of 1 for counting statistics. For XYE files, where *Yobs* is small and where *SigmaYobs* used (tof data for example), then w_min should be reduced.

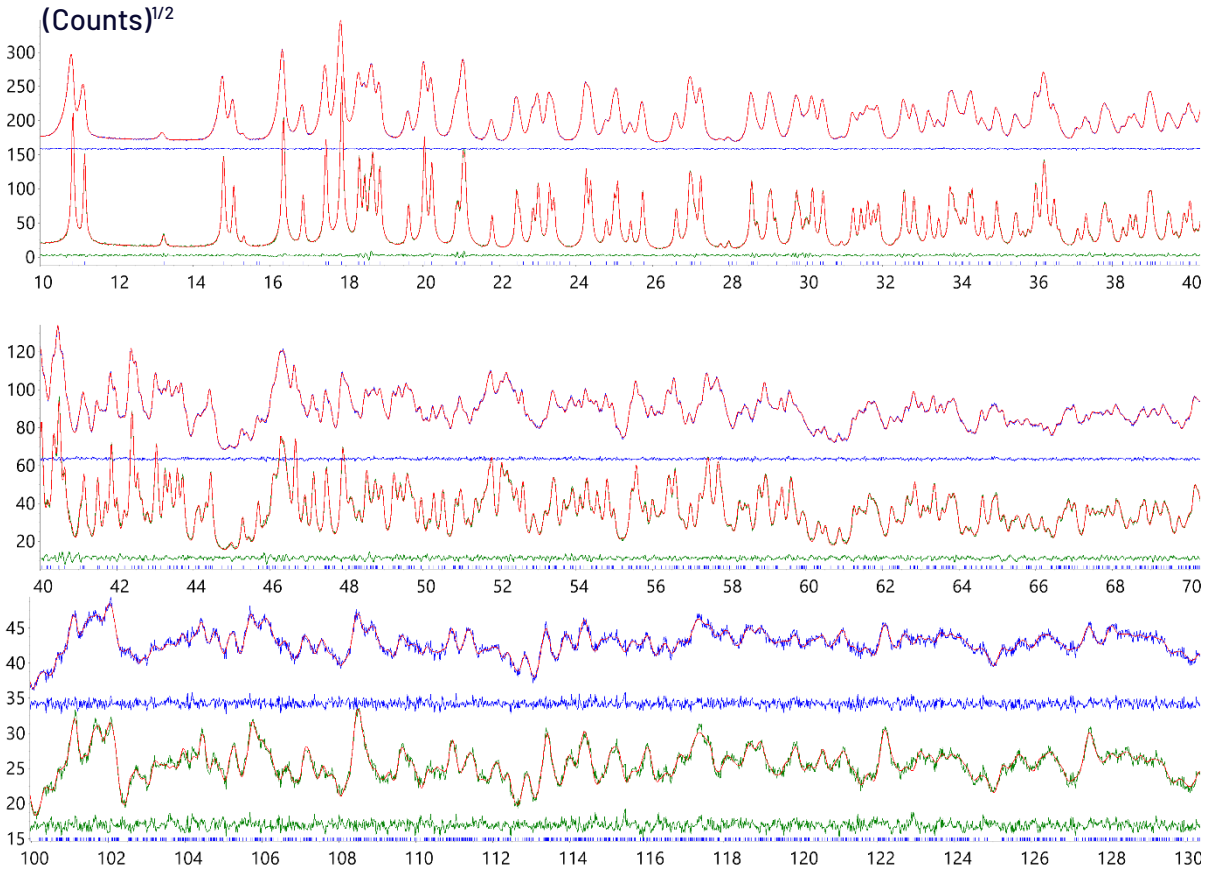
6.1 ..... Deconvolution – Simulated pattern

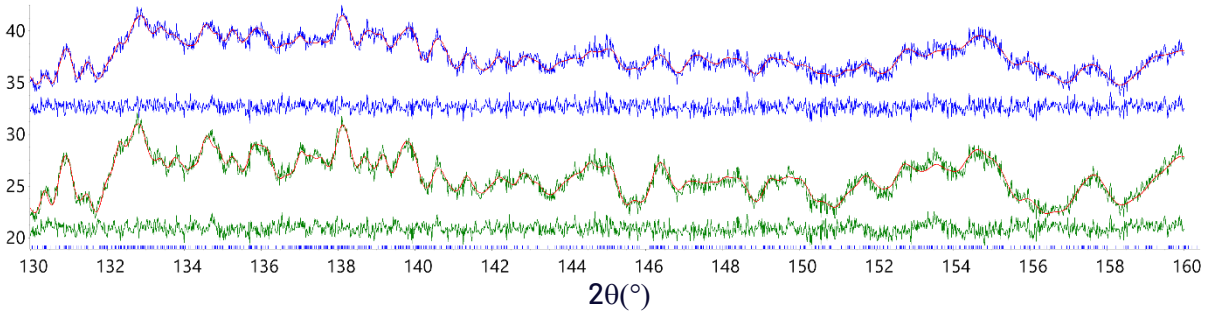
A simulated pattern was created with noise using SIM-CREATE.INP and the instrument contribution deconvoluted using SIM-DECON.INP; the latter INP file looks like:

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

       pbso4-decon-final.xy

   Define one at a time in the following:

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

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

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

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

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

The following figure is the deconvoluted pattern (green line, bottom plot) compared with the expected deconvoluted pattern (red line on top of green line). The top plot (blue line) is the original simulated pattern with noise and without noise (red line on top of blue line).





Parameter errors determined from refinement using the deconvoluted pattern are almost identical to errors produced using the original pattern, see [a]Coelho (2018).

7 ... PDF-GENERATION

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

PDF generation comprises an inverse Sine transform operating on an ideal diffraction pattern where background is absent, atomic scattering factors constant, and 2θ and peak shapes are symmetric. The task therefore becomes one of correcting real data such that it matches an ideal pattern as closely as possible. The corrections include determining a background, atomic scattering factors (if X-ray data), removing Lorentz polarization and removing asymmetry from peak shapes. To generate the PDF, a deconvolution process similar-to that described in section 5 is used. It allows for corrections in reciprocal space of peak asymmetry, instrument and emission profile aberrations, Lorentz polarization and atomic scattering factors corrections. The process comprises two operations described in a single INP file; these operations are:

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

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

7.1 ..... PDF-Generating - LiFePO4

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

```
Include_PDF_Generate
'-------------------------------------------------------
'              START USER INPUT SECTION
'-------------------------------------------------------
macro Data_File         { LFP_0-8Kcap_AgFGM_2x4soll_Eiger1D_8h.xy }
macro Capillary_Scan    { capillary.xy }
```

```
macro Capillary_Rebin   { 0.1 } ' Smooth the capillary scan. Zero means no smoothing.
#prm operation = 0; ' Set to 0 to fit to reciprocal space data
                    ' Set to 1 to generate F(Q) and G(r)
                    ' Set to 2 to fit structure to G(r)
#prm use_narrow_peak_shape  = 1; ' A 0 means use full peak shapes in generating G(r)
'-------------------------------------------------
' Inputs for reciprocal space fit, operation = 0
macro & Average_f           { f0_Li + f0_Fe + f0_P + 4 f0_O } ' formula of unit cell
#prm lab_no_monochromator   = 1; ' Set to 1 if using Laboratory instrument.
#prm use_Xo_Is_phase        = 1; ' Set to 0 if not fitting peaks
#prm use_bkg_penalty        = 1;
#prm use_simple_bkg_penalty = 1; ' Set to 1 if counting statistics is not right,
                                 ' or, maybe when there's Fluorescence.
macro & Bkg_Weighting               { 1 }
macro & Intensity_Penalty_Weighting { 1 }
macro & Scale_Peaks   { 1 } ' Useful if capillary absorption is inhibiting fitting.
macro & Scale_Yobs_By { 1 } ' Useful if data does not obey counting statistics.
prm pc0  1   ' Poly_Capillary coefficients; comment out if
prm pc1  0   ' not using Capillary as background.
inp_text fluorescence_bkg
   {
      bkg @ 3.49160163` -0.96682842`  0.292687899`
   }
inp_text fit_extra
   {
      penalty = 10000 (Bkg_at(X2) + (pc0 + pc1) Value_at_X(cap_, X2) - Yobs_at(X2))^2;
   }
macro Start_X       { 2.4  }
macro Finish_X      { 103  }
macro Step_X        { 0.02 } ' Set to zero to use measured step size.
                             ' Set to non-zero if scale_yobs_by is used.
                             ' Set to non-zero if unequal x-axis steps.
'-------------------------------------------------------------
' Inputs for generating F(Q), operation = 1
#prm poly_fq = 7;  ' Number of parameters for Poly when fitting Poly to F(Q).
                   ' View F(Q) plot, it needs to look right.
macro & Qmin                  { 0.1 }
macro & Qmax                  { 17.5 }
macro & Soper_Lorch_Constant  { 0 } ' best not to use
macro & Exp_Constant          { 0 } ' best not to use
macro & Lorch_Constant        { 0 } ' best not to use
inp_text fq_poly
   {
      bkg @  0 0 0 0 0 0 0 0  0
   }
macro FQ_Bkg_Penalty
   {
      weighting = If(X > (X2 - 1), 10, 1); ' Weigh the F(Q) data more at Qmax
      penalty   = Bkg_at(X1)^2;            ' Restrain F(Q=0) to 0
   }
'-------------------------------------------------------------
' Inputs for generating G(r) from F(Q), operation = 1
macro R_Max         { 100 }
macro dR            { 0.01 }
macro Num_Hats      { 3 } ' Best smoothing function for speed and accuracy
macro & Hat_Size    { 4.4934 / Qmax }
'-------------------------------------------------------------
' Reciprocal space peak details, operation = 0
```

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

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

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

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

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

Background = *Poly_Capillary* * *Capillary_Scan* + *Poly_Fluorescence*

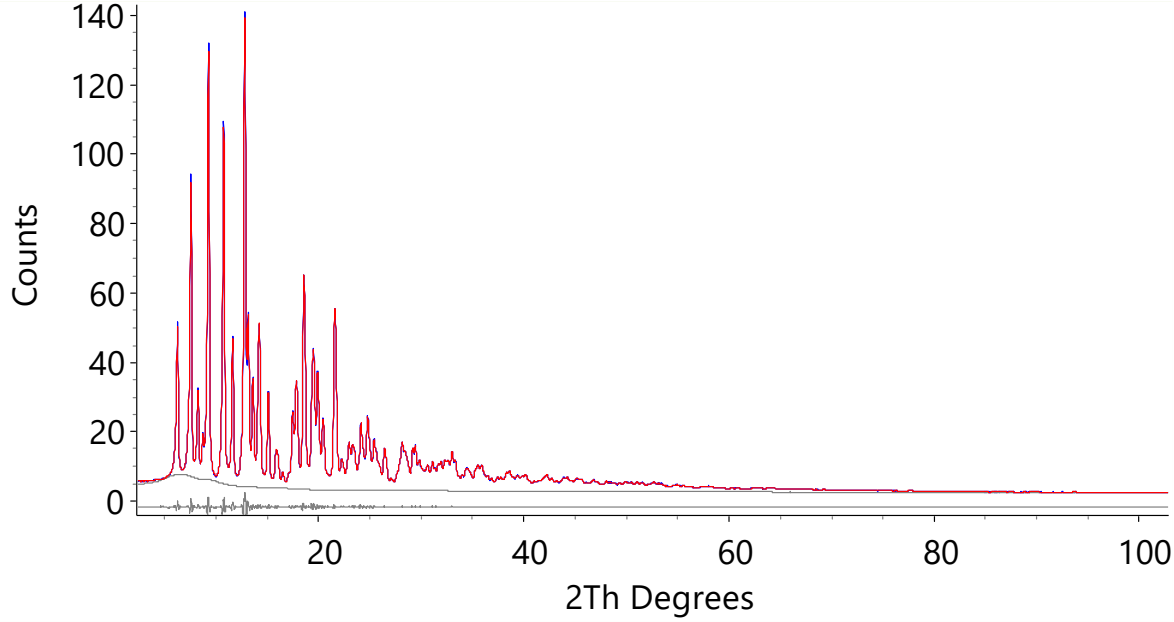
Poly_Capillary is a 1st order polynomial with coefficients defined by the pc0 and pc1 parameters. *Poly_Fluorescence* is also a nth order Chebyshev polynomial with coefficients defined by the user at the *inp_text* fluorescence_bkg {} construct; set this construct to blank when not using. LiFePO4 fluoresces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

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

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

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

The fit for LiFePO4 looks like:



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

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

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

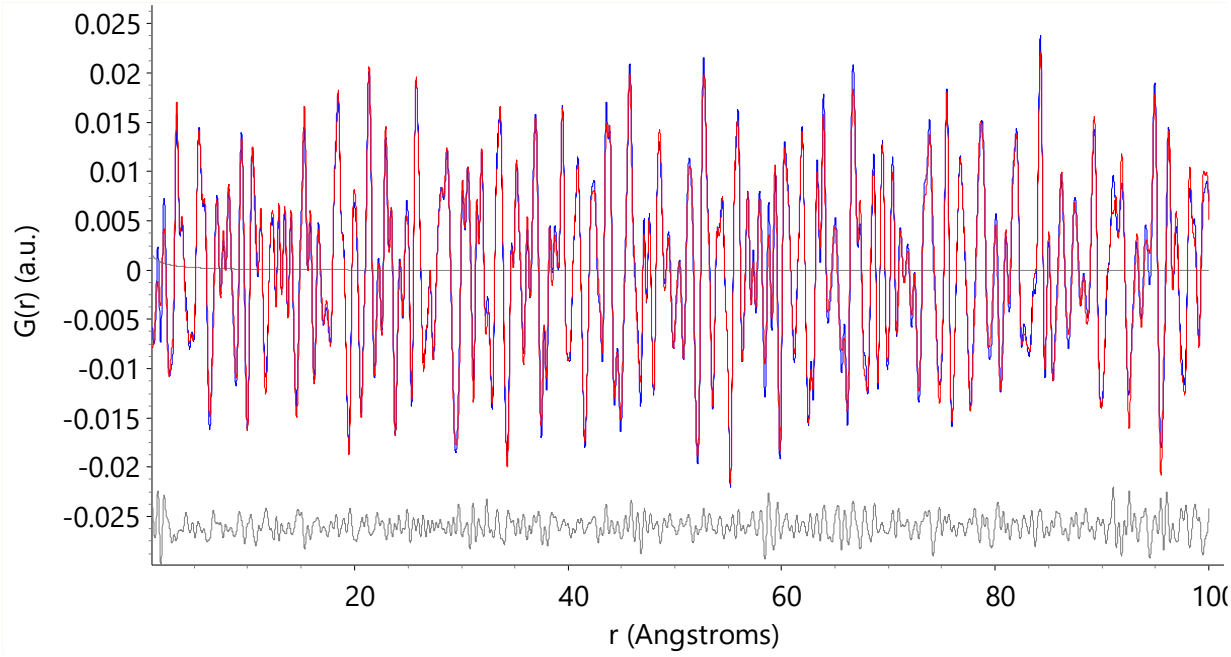
1.1.2 .....Operation 1 – Generation G(r) from the fitted peaks

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

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

fq_poly describes *Poly_FQ* using the Chebyshev polynomial of *bkg*; the optimum number of coefficients is difficult to determine automatically. Its best to inspect the plots produced by operation 1; these are generated and loaded into the GUI and, using the Tiling option, looks like:

Changing *fq_poly* and rerunning operation 1 updates the four plots; this updating is achieved using the keyword *gui_reload*. Using the structure of LiFePO4, the generated *G*(*r*) can be fitted-to by setting *operation*=2. With *use_narrow_peak_shape*=0 we get:



The grey line at the center of the plot is a correction added to the calculated *G*(*r*) using:

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

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

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

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

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

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

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

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

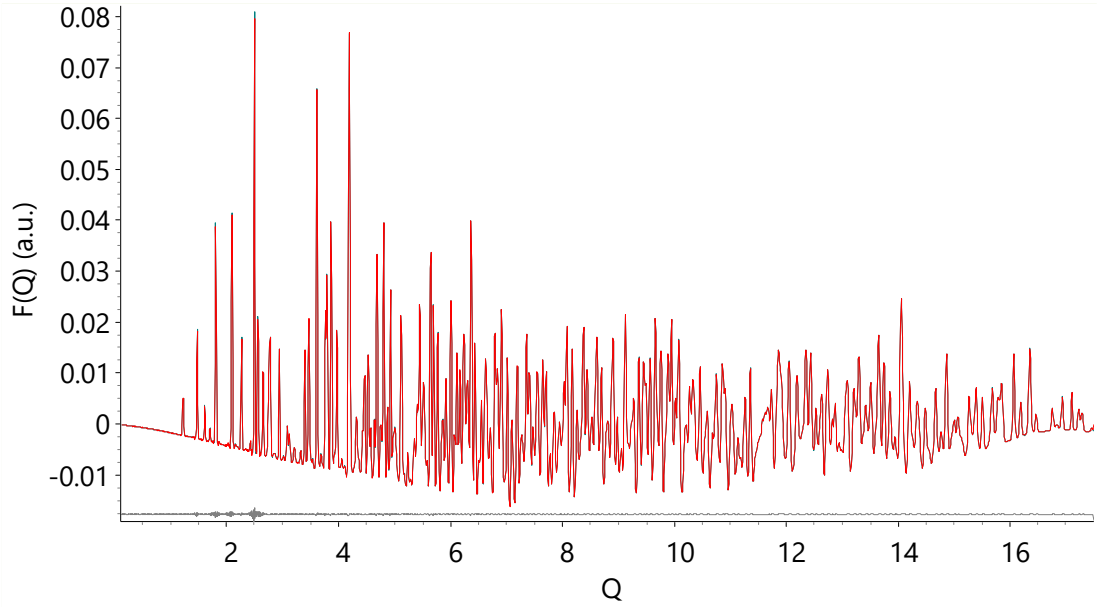
A zero-error added to peak positions in reciprocal can be subtracted from the deconvoluted pattern of operation 1.0. Thus, a zero-error determined from fitting to a standard in reciprocal space needs to be subtracted from the deconvoluted pattern from within the *Deconvoluted_Peak_Shape* macro.

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

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

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

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

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



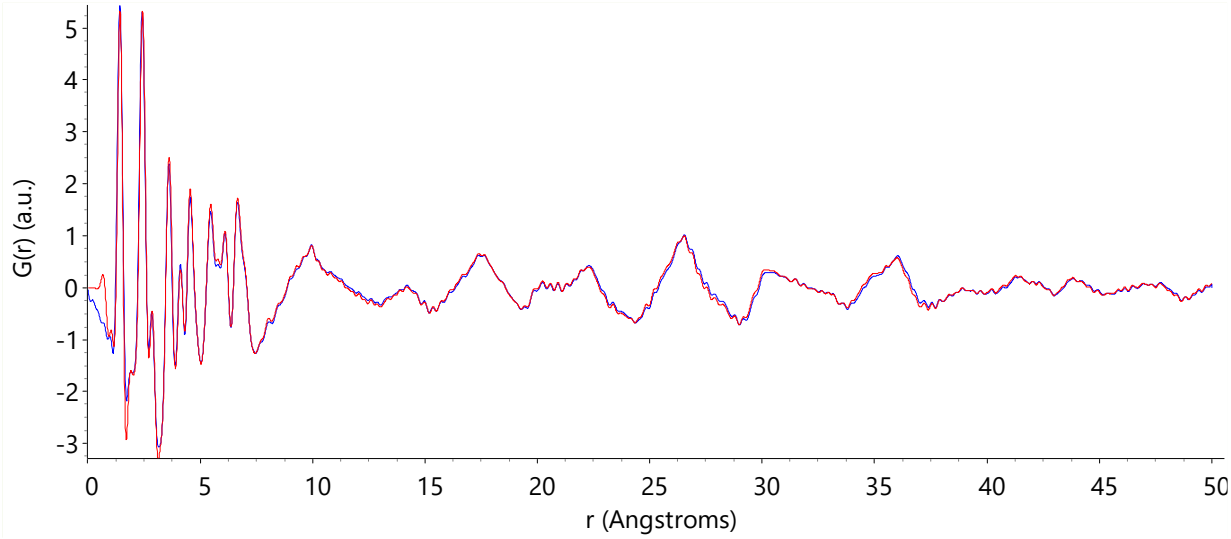
7.2 ..... PDF-Generating - Fullerene

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

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

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

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

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

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



7.3 ..... Levenberg-Marquardt constant determination

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

8 ... MISCELLANEOUS

8.1 ..... Capillary convolution for a focusing convergent beam

The capillary convolution has been extended to include a focusing convergent beam (Coelho & Rowles, 2017); syntax is as follows:

[*capillary_diameter_mm* E]
  capillary_u_cm_inv E
  [*capillary_convergent_beam*][*capillary_divergent_beam*][*capillary_parallel_beam*]
  [*capillary_focal_length_mm* E]
  [*capillary_xy_n* #]

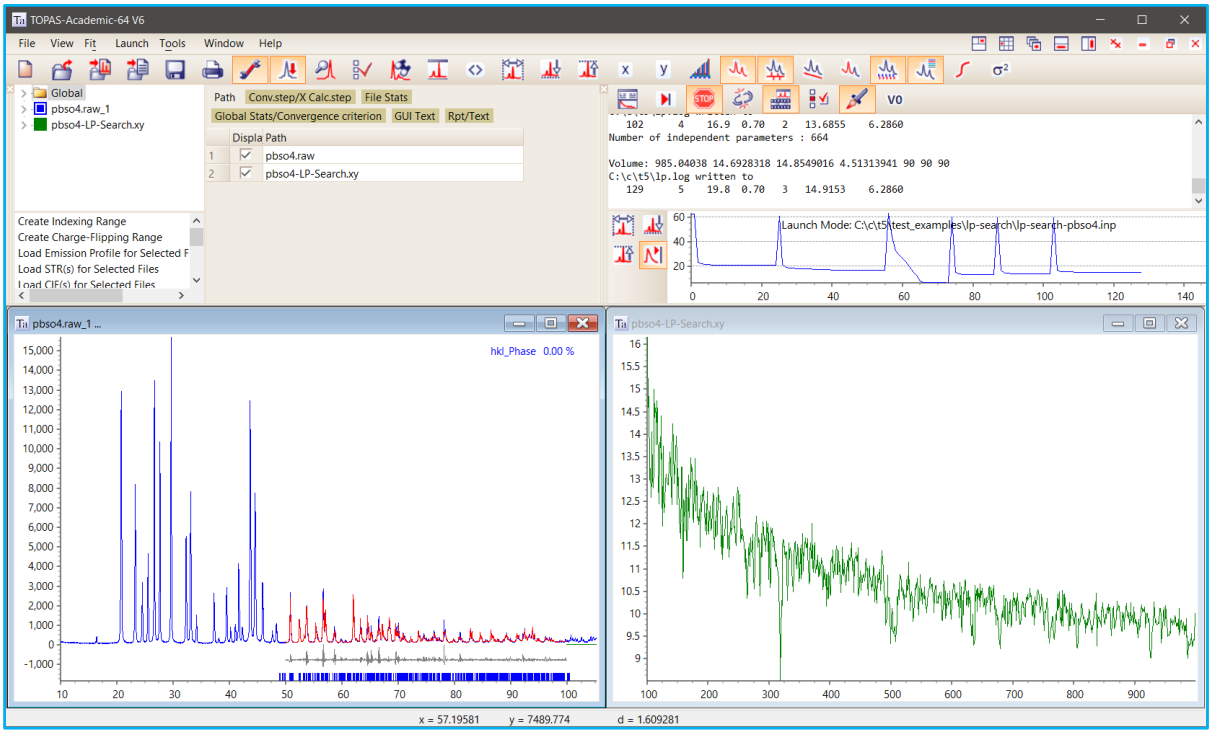
See examples LAB6-STOE.INP and LAB6-D8.INP in the directory TEST_EXAMPLES\CAPILLARY. If using a *str* phase then *capillary_u_cm_inv* can be set to the calculated linear absorption coefficient multiplied by a packing density, for example:

```
prm packing_density 0.31208
capillary_diameter_mm @ 0.57313
   capillary_u_cm_inv
      = Get(mixture_MAC) Get(mixture_density_g_on_cm3) packing_density;
   capillary_focal_length_mm @ 197.89657
   capillary_convergent_beam
```

If *capillary_focal_length_mm* is not defined, then it defaults to the diffractometer radius *Rs*.

8.2 ..... LP-Search - threaded, faster and more exhaustive

LP-Search (Coelho, 2017) uses a new figure of merit function, *G*, for indexing in a Monte Carlo search process for finding lattice parameters without the need for peak position extraction, see TEST_EXAMPLES\LP-SEARCH\LP-SEARCH-PBSO4.INP. LP-Search performs a Pawley refinement after each LP-Search Monte Carlo cycle. It also displays G_{min} as a function of Volume. This G_{min} plot is useful as it shows contrast between the minimum of G_{min}, corresponding to the most probable solution, and the rest of G_{min}. Solutions obtained at the end of each Pawley refinement is saved to the file LP.LOG. The example LP-SEARCH-PBSO4.INP demonstrates LP-Search applied to a $PbSO_4$ pattern whilst using only the high angle part of the pattern where peaks are heavily overlapped. The output, with the G_{min} versus volume plot on the right, is as follows:



Comment out '*#define* USE_HIGH_ANGLE_DATA_ONLY' to use the low angle region to see the difference. The LP-Search figure of merit function, *G*, keyword *lp_search_fom*, can be separately used in a penalty function, for example:

```
prm !Np 3
penalties_weighting_K1 = If(Cycle_Iter < Np, 20, 0);
lp_search_fom fom  6.69033248`
penalty = 1000 If(Cycle_Iter < Np, fom, 0);
```

Here the penalty is applied in the first three iterations of a Pawley refinement (see LP-SEARCH-PBSO4.INP).

8.3 ..... Sine and cosine transforms

[*ft_conv_re_im*] ... [*ft_conv_re* E] [*ft_conv_im* E] [*WPPM_ft_conv_re_im* E]... [*WPPM_ft_conv_re* E] [*WPPM_ft_conv_im* E]	

Sine and cosine transforms. *ft_conv* is equal to the two keywords [*ft_conv_re_im ft_conv_re*] and similarly *WPPM_ft_conv* is equal to [*WPPM_ft_conv_re_im WPPM_ft_conv_re*]. More than one transform can be defined. See *ft_conv* and *WPPM_ft_conv* for details.

8.4 ..... **.SST data files*

Data files with an SST extension implies an equal x-axis and has the following format:

- start, step, data points....

SST files can be loaded wherever *.XY files are loaded. They save space on creation as well as on loading as x-axis values are not needed. For equal x-axis data then the macro *Out_XDD_SST* can be used in the following manner:

```
xdd …
   Out_XDD_SST(filename.sst) = Ycalc;        ' outut Ycalc
   Out_XDD_SST(filename.sst) = Yobs – Ycalc; ' output difference plot
```

8.5 ..... *xdd_array* and nested *xdd_sum*

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

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

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

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

xdd_sum is similarto *xdd_array* except an array is not stored; instead, the sum of the values of the array are calculated and stored. Similar-to *xdd_array*, the summed value of *xdd_sum* is only recalculated when necessary. In Version 7, *xdd_sum* can be nested, for example, to normalize the intensities between *Yobs* and *Ycalc* the following is now possible:

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

8.6 ..... String_To_Variable and Double_To_String functions

The function *Double_To_String* converts a number to a string. The macro *String_To_Variable* converts a string to a variable. Together, these macros provide flexibility in the creation of INP files where the number of data files and phases are large; example usage is as follows:

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

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

8.7 ..... Restraining background using the Bkg_at function

The Chebyshev background function, *bkg*, can sometimes misbehave during Pawley, Le Bail or deconvolution refinements. In the case of *xdd* deconvolution refinements, the *Deconvolution_Bkg_Penalty* stabilizes *bkg* in most cases. In cases of instability, however, the Bkg_at(x) function can be used in penalty functions to guide the shape of *bkg*. Bkg_at(x) returns the value of *bkg* at the x-axis position of x. Example use of Bkg_at as applied to TOF data is:

```
bkg  @  0.443519294`  0.0200324829`  0.0113774736`
penalty = 1000000 (Bkg_at(2036)  - 0.43)^2;
penalty = 1000000 (Bkg_at(9000)  - 0.50)^2;
penalty = 1000000 (Bkg_at(14600) - 0.50)^2;
```

The first penalty restrains the value of *bkg* at x=2036 to 0.43. Typically, only two to three Bkg_at penalties are necessary. The values of 0.43, 0.5 and 0.5 can be determined graphically.

8.8 ..... phase_out_X

[*phase_out_X* $file [*append*]]...

Phase dependent keyword that writes phase Ycalc details to a file. The *out_eqn* can contain reserved parameter names occurring in *xdd_out* as well as Get(phase_ycalc); for example:

```
    phase_out_X Phase.txt load out_record out_fmt out_eqn {
        " %9.0f" = Xi;
        " %11.5f" = X;
        " %11.5f" = Get(phase_ycalc);
        " %11.5f" = Ycalc;
        " %11.5f" = Yobs;
        " %11.5f\n" = Get(weighting);
    }
```

The x-axis extent of the output corresponds to the x-axis range of the phase. If *conserve_memeory* is used, then the message "phase_out_X: No data" is outputted.

8.9 ..... Functions allowing access to rigid-body fractional atomic coordinates

The standard macro *Point*(site_name, rx), see TOPAS.INC, returns the x Cartesian coordinate of the point called site_name; y and z Cartesian coordinates are returned by ry and rz objects respectively. These functions can only to be used in equations of the rigid body which encompass the keywords and their dependents of *point_for_site*, *z_matrix*, *translate* and *rotate*. The actual value return by *Point* depends on where it is used in the rigid-body, for example, in the following:

```
rigid
    point_for_site O1
    translate tx 1
    point_for_site O2 ux = Point(O1, rx); ' Point here returns 1
    translate tx 2
    point_for_site O3 ux = Point(O1, rx); ' Point here returns 3
```

the final x Cartesian coordinate of site O3 becomes 3. To instead return fractional coordinates of points, the functions Point_rx_ua, Point_rx_ub and and Point_rx_ua can be used. These functions are passed the address of the point in question using the *Point* macro with one argument. Accompanying macros simplifying the call, as defined in TOPAS.INC, are:

```
macro Point_ua(site_name) { Point_rx_to_ua(Point(site_name)) }
macro Point_ub(site_name) { Point_ry_to_ub(Point(site_name)) }
macro Point_uc(site_name) { Point_rz_to_uc(Point(site_name)) }
```

These macros can return many different values for the same point in question depending on where they are called during the rigid body calculation.

8.10 .... set_top_peak_area

[*set_top_peak_area* E]...

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

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

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

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

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

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

8.11 .... bring_nth_peak_to_top

[*bring_nth_peak_to_top* !E]...

A peak stack operation (see *push_peak*) where the n^{th} peak from the top of the stack is placed at the top of the stack; n=0 corresponds to the top of the stack; the following two entries are equal:

```
bring_nth_peak_to_top 1
bring_2nd_peak_to_top
```

8.12 .... scale_occ keyword

[*scale_occ* E] is *occ* dependent and it scales *occ*. It and can be a function of *H*, *K*, *L*, *D_spacing*, *Xo* and *Th*. The *occ* keyword remains single valued for QUANT purposes and thus cannot be a function of *H*, *K*, *L* etc. The following is valid:

```
occ Pb+2 1
   prm q1 1 min 1e-6
   prm q2 1 min 1e-6
   prm q3 1 min 1e-6
   prm q4 1 min 1e-6
   scale_occ = q1 / D_spacing + 1 / (q2 H^2 + q3 K^2 + q4 L^2);
```

scale_occ is a child of *occ*, the keyword therefore needs to occur after the *occ* keyword. The following two definitions will produce identical structure factors but different QUANT results:

```
site Pb occ Pb+2 1   beq 1
site Pb occ Pb+2 0.5 beq 1 scale_occ 2
```

scale_occ works with magnetic data, neutron data, x-ray data etc... but not PDF data.

Symmetry: The user is responsible for obeying symmetry. If not working in *P*1 then the *Multiplicities_Sum* macro could be used. The *spherical_harmonics_hkl* keyword can also be used, for example:

```
spherical_harmonics_hkl sh sh_order 6
site Pb occ Pb+2 1 beq 1
   prm q 1 min 1e-6
   scale_occ = q sh;
```

8.13 .... p1_fractional_to_file

[*p1_fractional_to_file* $file][*in_str_format*]

Structure dependent. Saves atomic positions corresponding to space group *P*1 to the file $file. The original space group can be any space group. If *in_str_format* is defined, then the structural data is saved in INP format.

8.14 .... Determining the orientation of a known fragment using a Rigid-Body

Determines rotation and translation parameters for a known fragment, see TEST_EXAMPLES\RIGID\MATCH.INP. The known fragment is in fractional coordinates. To do the same for a fragment in Cartesian coordinates then change the lattice angles to 90 degrees and adjust the lattice parameter lengths. Also, see:

http://topas.dur.ac.uk/topaswiki/doku.php?id=rigid_body_-_matching_to_a_known_fragment

8.15 .... user_defined_starting_transition

[*generate_stack_sequences*]{ ...
  [*user_defined_starting_transition* $transition_name]
}

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

8.16 .... Using a user defined table to input f0 values via user_y

Atomic scattering factors *f0* can be defined in a *.XY file and used via the *user_y* keyword as follows:

```
xdd ...
   user_y C_f0_table C_f0_table.xy
   str
```

```
load f0_f1_f11_atom f0 f1 f11 { C = C_f0_table; 0 0 }
...
```

Here the C_F0_TABLE.XY file comprises D_spacing and f0 value pairs which is used to describe *f0* values for the C atoms within the structure. In the above example, *f1* and *f11* are set to zero.

8.17 .... Extending user_y

[*xdd*]...
  [*user_y* $name { *#include* $file }]... | [*user_y* $name $file]...
    [[1]*xye_format*]
    [[1]*rebin_with_dx_of* !E]
    [[1]*user_y_hat* E]...
    [[1]*user_y_gauss_fwhm* E]...
    [[1]*user_y_lor_fwhm* E]...
    [[1]*user_y_exp_conv_const* E [*user_y_exp_limit* E]...

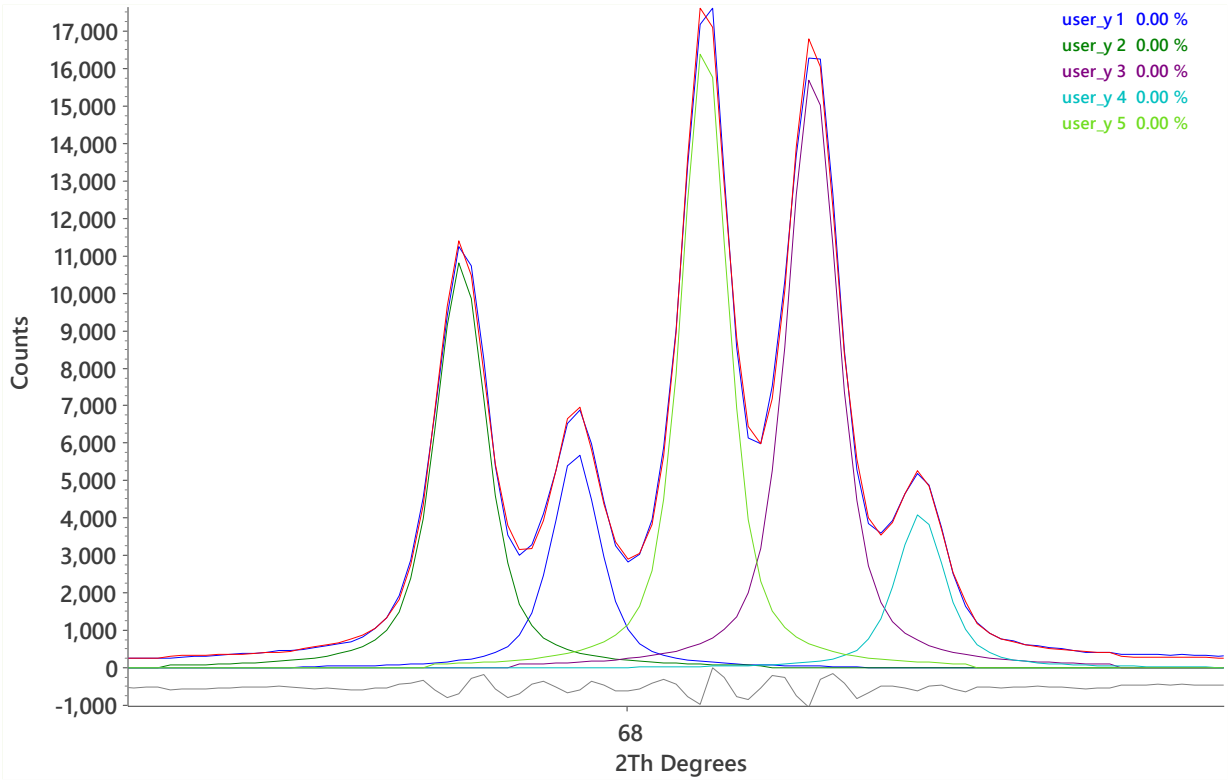
---

[1]New *user_y* dependents. *user_y_hat*, *user_y_gauss_fwhm*, *user_y_lor_fwhm* and *user_y_exp_conv_const* are identical to the *hat*, *gauss_fwhm* and *lor_fwhm* and *exp_conv_const* convolutions except they are applied to the *user_y* data.

user_y can be used to add, multiply and in general manipulate data files of different x-axis steps. For example, to add two data files, square the result and then multiply by the x-axis reserved parameter *X*, the following can be used:

```
user_y f1 file1.xy
user_y f2 file2.xy
yobs_eqn result.sst = X (f1 + f2)^2; min 10 max 100 del 0.01
```

The test example USER_Y\USER_Y.INP fits five fit objects to the quartz triplet using a learnt peak shape defined using *user_y*; the fit with the individual *fit_obj*'s displayed (using the macro *Plot_Fit_Obj*) looks like:



The test example USER_Y\USER_Y_CONVOLUTION.INP fits five fit objects to a simulated pattern using a learnt peak shape defined using *user_y* with the two *user_y* convolutions of *user_y_exp_conv_const* and *user_y_gauss_fwhm*; the INP file looks like:

```
'#define CREATE_SIMULATED_
continue_after_convergence

macro FO_Peak(& p, & pe, & a, & x, & s)
   {
      fit_obj = a p;
         min_X = -pe s + x; max_X = pe s + x;
          fo_transform_X = (X - x) / s;
   }

prm !peak_extent 2

#ifdef CREATE_SIMULATED_
    iters 0
    user_y peak { _xy -0.01 0 0 100 0.01 0  }
       user_y_exp_conv_const @ 1    min 0.5 max 2
       user_y_gauss_fwhm     @ 0.1  min 0.1 max 2
   yobs_eqn = 1; min 66 max 70 del 0.01
       gui_ignore ' don't load data file into GUI
       Out_X_Ycalc(user_y_convolution.xy)
#else
    ' Fit to the simulated peak
    user_y peak { _xy -0.01 0 0 100 0.01 0  }
       user_y_exp_conv_const @ 1   min 0.5 max 2 val_on_continue = Rand(0.5, 2);
       user_y_gauss_fwhm     @ 0.1 min 0.1 max 1 val_on_continue = Rand(0.1, 1);
    xdd user_y_convolution.xy
#endif
      start_X  66
      finish_X 70
```

```
bkg @ 100
prm a1 1000 min 1.0e-6 val_on_continue = Rand(1, 100);
prm a2 2000 min 1.0e-6 val_on_continue = Rand(1, 100);
prm a3 3000 min 1.0e-6 val_on_continue = Rand(1, 100);
prm a4 2000 min 1.0e-6 val_on_continue = Rand(1, 100);
prm a5 1500 min 1.0e-6 val_on_continue = Rand(1, 100);
prm x1 67.7 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
prm x2 67.9 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
prm x3 68.1 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
prm x4 68.3 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
prm x5 68.5 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
prm s1 0.7 val_on_continue = Rand(0.5, 2); min 0.5 max 2
prm s2 0.9 val_on_continue = Rand(0.5, 2); min 0.5 max 2
prm s3 1.1 val_on_continue = Rand(0.5, 2); min 0.5 max 2
prm s4 1.0 val_on_continue = Rand(0.5, 2); min 0.5 max 2
prm s5 0.8 val_on_continue = Rand(0.5, 2); min 0.5 max 2

FO_Peak(peak, peak_extent, a1, x1, s1) Plot_Fit_Obj("user_y 1")
FO_Peak(peak, peak_extent, a2, x2, s2) Plot_Fit_Obj("user_y 2")
FO_Peak(peak, peak_extent, a3, x3, s3) Plot_Fit_Obj("user_y 3")
FO_Peak(peak, peak_extent, a4, x4, s4) Plot_Fit_Obj("user_y 4")
FO_Peak(peak, peak_extent, a5, x5, s5) Plot_Fit_Obj("user_y 5")
```

8.18 .... New Keywords

[*num_cycles* #]

Determines the number of cycles to process when *continue_after_convergence* is defined. The number of iterations, defined using *iters*, is still adhered to. Thus, to set number of cycles to 100 then using something like:

```
continue_after_convergence
iters 1000000000
num_cycles 100
```

[*suspend_writing_to_log_file* #1]

When *num_runs* > 0, then, by default, output to TOPAS.LOG (or TC.LOG if running TC.EXE) is suspended after the first run (Run_Number == 0). *suspend_writing_to_log_file* changes this behaviour.

xdd...
[*gui_reload*]
[*gui_ignore*]

When in Launch mode; data files by default are not reloaded if already loaded. *gui_reload* forces the reload of the data file. Data files are loaded/reloaded into the GUI under the following circumstances:

- The data file is not loaded into the GUI
- Any of the following keywords have been used at the *xdd* level:
  gui_reload, *rebin_with_dx_of*, *smooth*, *yobs_eqn*, *yobs_to_xo_posn_yobs*

gui_reload can be used in cases where the data file has been changed by a process not listed. *gui_ignore* informs the GUI to ignore the *xdd* data file; *Ycalc*, difference and other items associated with the data file is not retrieved from the Kernel.

[*inp_text* $name] ...
[*inp_text_insert* $name { ... }]...

inp_text provides a means of defining INP text at one place in a file and having that text inserted at another place in the INP file, or in an #include file, using *inp_text_insert*. The *inp_text* is updated on refinement termination. *inp_text* is useful in simplifying complicated INP files where placing control parameters at the top of the file is of benefit; see test_example INP-TEXT.INP. An example is as follows:

```
inp_text back_ground {
   bkg @  17.365576`  14.5555883`  14.038067`
}
xdd …
   inp_text_insert back_ground
```

More than one *inp_text* can be of the same name; in such cases *inp_text_insert* will use the most recent *inp_text*.

8.19 .... New functions

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

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

Yobs_Min(x1, x2) : Returns the minimum value of *Yobs* between x1 and x2.

Yobs_at(x) : Returns the value of *Yobs* at x. Zero is returned of x < *X1* or x > *X2*.

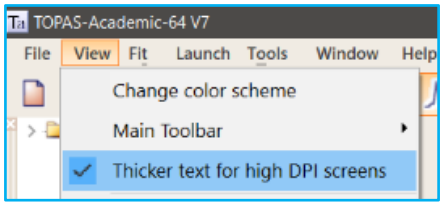
Ycalc_at(x) : Returns the value of *Ycalc* at x. Zero is returned if x < *X1* or x > *X2*.

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

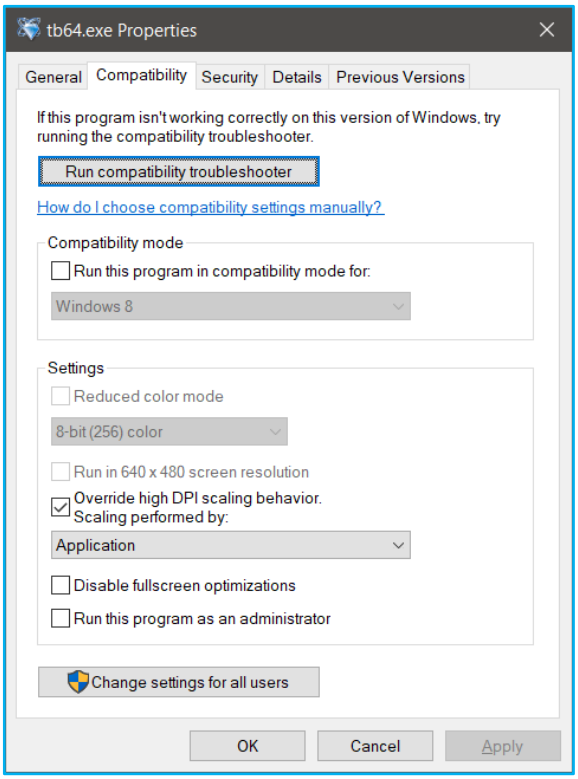
9 ... NEW GUI FUNCTIONALITY

9.1 ..... TOPAS is DPI aware

Monitors with a high number of Dots Per Inch (DPI), often display text that are too small. Windows can scale fonts using Windows font scaling to enlarge text. This scaling is carried through to TOPAS where fonts and bitmaps scale to the required size. Additionally, a thicker-text option ("Segoe UI Semibold") can be enabled if the TOPAS text appears too thin. The option is saved for subsequent TOPAS loads and is enabled/disabled from the View menu:



After applying Windows scaling, TOPAS needs a Sign-Out and a Sign-In to display all text correctly scaled. Also, for TOPAS to use its DPI capabilities, the properties of the executable needs to be set to "Application", i.e.



9.2 ..... Antialiasing and OpenGL

Enable Antialiasing on your graphics card to display smooth lines in OpenGL; this affects all OpenGL displays. Depending on the graphics card Antialiasing can also be enabled on a program specific manner.

9.3 ..... Displaying a phase with and without background

Phases can be plotted with or without background by cycling through the three states of the phase-display icon.

9.4 ..... How atoms are displayed in OpenGL

Atoms colours and radii are defined in the files ATOM_COLORS.DEF and ATOM_RADIUS.DEF respectively. A site defined as:

```
site S1 occ Al+3 1 beq 1
```

will be displayed as a Sulphur atom. If the Site Name, minus the numbers, is not found in ATOM_COLOURS.DEF then the atom type defined at the first site occupancy is used. Thus, a site defined as:

```
site _S1 occ Al+3 1 beq 1
```

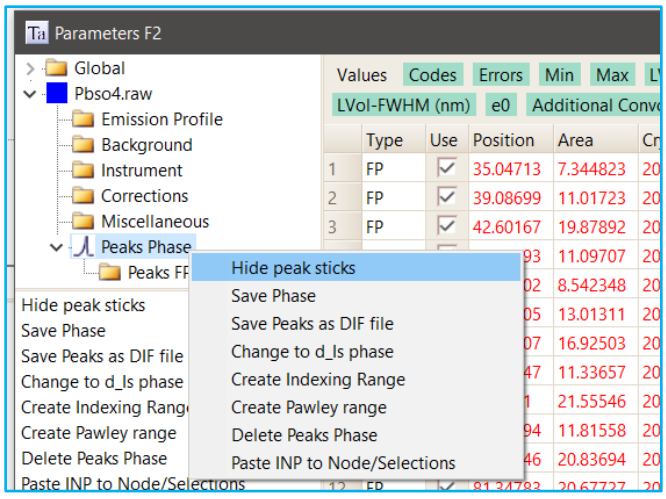
will be displayed as an Aluminium atom.

9.5 ..... x_calculation_step deleted when constant x-axis step size detected

*.XY and *.XYE data files are converted to a constant x-axis step size when a constant step size is detected. When this occurs Version 7 removes the "Calc.Step" item from the GUI menus for the corresponding data file. A small calculation step size can still be used by increasing "Conv. Steps". PRO files containing an *x_calculation_step* will still show an entry of *x_calculation_step*.

9.6 ..... hide_peak_sticks

A GUI option that toggles the display of peak sticks in the scan window; the option can be found at the Peaks phase level as follows:



10 . REFERENCES

Burla, C.B; Carrozzini, B.; Cascarano, G. L.; Giacovazzo C. & Polidori, G. (2011). *J. Appl. Cryst.* 44, 1143–1151

Coelho, A. A. (2007). *Acta Cryst.* **A36**, 400–406. "*A charge-flipping algorithm incorporating the tangent formula for solving difficult structures*"

Coelho, A. A. & Rowles, M. R. (2017). *J. Appl. Cryst.* 50, 1331-1340. https://doi.org/10.1107/S160057671701130X. "*A capillary specimen aberration for describing X-ray powder diffraction line profiles for convergent, divergent and parallel beam geometries*".

Coelho, A. A. (2017). *J. Appl. Cryst.* 50, 1323-1330. https://doi.org/10.1107/S1600576717011359 "*An indexing algorithm independent of peak position extraction for X-ray powder diffraction patterns*".

[a]Coelho, A. A. (2018). *J. Appl. Cryst.* 51, 112-123. https://doi.org/10.1107/S1600576717017988, "*Deconvolution of instrument and Kα contributions from X-ray powder diffraction patterns using least squares with penalties*".

[b]Coelho, A. A. (2018). *J. Appl. Cryst.* 51, 210-218. https://doi.org/10.1107/S1600576718000183, "*TOPAS & TOPAS-Academic: An optimization program integrating computer algebra and crystallographic objects written in c++*".

[c]Coelho, A. A. (2018). *J. Appl. Cryst.* 51, 428-435.*"Optimum Levenberg-Marquardt constant determination for nonlinear least-squares*".

Elser, Veit; Lan, Ti-Yen and Bendory Tamir (2017). arXiv:1706.00399 [cs.IT]. "*Benchmark problems for phase retrieval*".

Mooers, B. H. M. (2016). Acta Cryst. D72, 477–487