NICOS knowledge for Zebra (compiled from NICOS documentation and 'single_crystal' instructions of M. Koennecke)

COMMANDS FROM A UNIX TERMINAL:

startnicos starts NICOS from zebra terminal (**this command only if desperately needed, restarts nicos and epics**), related commands stopnicos, startsics, stopsics
monit restart nicos-daemon re-starts nicos
nicos-client starts command-line version of NICOS (for those who don't like GUI…).
nicos-gui starts GUI version of NICOS (**for USER**). host: localhost:1301 ; user: **user** ; password: **23lns1**
if nicos logs are needed they are on /home/zebra/nicos/log/nicos one file is created per day, and on restart

COMMANDS (python) to be used from NICOS command-line or from the GUI under 'Instrument interaction':

(use ListCommands() to see all available commands)

1) Configure Nicos mode (**USER should skip this part**)

SetMode(mode) sets the execution mode:
'master' for normal operation (only one Nicos instance at a time);
'slave' for secondary copies of the instrument control software (only read status);
'simulation' for complete simulation of the instrument (based on current status, without communication).

Sim(filename_or_code) runs a script from filename or the piece of code in the brackets in dry run mode (ranges of all devices moved are recorded and the required time estimated from device properties 'speed', 'ramp' and 'accel').

sync() synchronizes dry-run/simulation copy with master copy (applies all current setups to simulated devices).

2) Configure the instrument (**USER may skip this part**): **in NICOS GUI this can be done under 'Setup, Instrument'**

NewSetup('') loads the setup for the Eulerian cradle instead of the current one. Several setups may be loaded at the same time, e.g. NewSetup('zebraeuler', 'detector_2d') loads cradle with area detector.

AddSetup('') loads the given setups additional to the current one, e.g. AddSetup('euler').

RemoveSetup('zebraeuler') unloads a setup (setups are mutually exclusive).

zebraconf() reads the instrument configuration from hardware connection (SPS).

ListSetups() shows all available setups.

Structure of Zebra in Nicos
Setups (installs corresponding geometries and have **ZEBRA**, **h**, **k**, **l** as devices): **zebraeuler**, **zebrakappa**, **zebranb**, **zebratas**
Components and associated devices: **wagen1** (ana, detdist, nu) **wagen2** (a5, a6, ana, detdist, nu, s2anh, s2anv, s2anb, s2anl, s2anr, s2ant) **detector_single** (counts, elapsedtime, intensity, monitor1, monitorpreset, protoncount, timepreset, zebradet) **detector_2d** (area_detector, elapsedtime, histogrammer, monitorpreset, protoncount, timepreset, zebradet) **monochromator** (wavelength, cex1, cex2, mexz, and mcv*, mgp*, mgv*, mom*, mtp*, mtv* with * being u or l) **slits** (s1h, s1v, s1hl, s1hr, s1vb, s1vt + same for s2, and for nose: snh, snv, snhm, snvm) **sps** (coll2d, euler_present, mcu2, pgfilter, shutter) **sample** (sch, som, sph, stt, sx, sy, sz)

3) Configure the experiment (proposal, user, sample): **in NICOS GUI this can be done under 'Setup, Experiment'**

NewExperiment(proposalid, title, local_contact_email, user_email) sets up proposalid, etc…. Note that the previous experiment has to be finished to start a new one, done either in the GUI (may not work) or with the command below.
AddUser(user_email) to add other users.
ListUsers(user_email) lists all users.

SetMailReceivers(email, ...) sets a list of emails to be notified on unhandled errors or when the Notify('some text') command is used.

FinishExperiment() finishes experiment (**in NICOS GUI this can be done under 'Finish Experiment'**).

4) Samples

Sample.new({'name': 'CHO', 'a': 10.7, 'bravais': 'F', 'laue': 'm3m'}) clears all crystallographic information, initializes the diffractometer and sets up an initial B matrix.
The parameter for Sample.new() is a dictionary with entries:
a, b, c the lattice constants
alpha, beta, gamma the lattice angles, defaulted to 90 when not given
lattice an alternative to give cell constants individually, expects a list of [a, b, c] as argument
angles an alternative to give cell angles individually, expects a list of [alpha, beta, gamma] as argument
bravais the lattice type of the sample
laue the point group of the sample

SetSample(number, name, ...) updates sample name and parameters for given sample number.
ListSamples() lists all information about defined samples.
SelectSample(number_or_name) selects the sample with the given number or name.
ClearSamples() clears current sample information and delete all stored samples.

5) Manual entries in data files and electronic logbook

Remark(remark) changes the data file remark about instrument configuration (to record sections of an experiment or changes to instrumental setup that are not otherwise visible in the NICOS devices), e.g. Remark('search for peaks'). Use Remark('') to remove the current remark.

LogEntry(entry) makes a free-form entry in the electronic logbook, e.g. LogEntry('search for peaks').

6) Run the instrument (device, measuring, scanning and on-line analysis commands)

a. Device commands (**mostly also doable in GUI 'Instrument interaction': double-click on a device from right panel**)

ListParams(*dev*) lists all parameters of the device.

dev.param queries the value of a parameter of a device
dev.param=newvalue sets a newvalue of a parameter

ListMethods(dev) lists the methods of a device
dev.method(par,…) calls a method of a device using specified parameters

---

read([*dev*, ...]) reads the value of devices (if no device is given, read all readable devices). The output is readable for humans but may not be convenient to use to assign a motor value to a variable, in which case use **device.read(0)**, e.g. in a script, center('om', 10, 0.1, 10, m=1000) then x = om.read(0) then cscan('om', x, 0.1, 8, m=1000000).

move(*dev1, pos1, ...*) moves to a position without waiting for movement to finish, unless wait(dev1, ...) is added.
maw(*dev*, *pos*, ...) moves devices to new positions and wait for them.

rmove(*dev, delta, ...*) moves devices by a relative amount.
rmaw(*dev*, *delta*, ...) moves devices by relative amounts and wait for them.

stop([*dev*, ...]) stops devices (if no device is given, stop all stoppable devices in parallel)

---

status([*dev*, ...]) reads status of devices (if no device is given, read the status of all readable devices)

reset(*dev*, ...) resets devices, e.g. restores communication, re-sets positioning fault.

reference(*dev*, **args*) does a reference drive of the device, if possible.

info([*dev*, ...]) prints general information of devices (value, status and any parameter marked as "interesting").

---

limits([*dev*, ...]) prints limits of devices, both absolute limits (dev.abslimits) and user limits (dev.userlimits).
To set userlimits, use either *dev*.userlimits = (low, high) or set(*dev*, 'userlimits', (low, high)).

resetlimits(*dev*, ...) resets userlimits to the maximum allowed range.

---

fix(*dev*[, *reason*]) fixes a device (a reason can be displayed when movement is attempted).
release(*dev*, ...) releases one or more devices,

disable(*dev*, ...) disables one or more devices (WAIT A FEW SECONDS).
enable(*dev*, ...) enables one or more devices => DOES A RESTART OF CORRESPONDING MCU (WAIT A FEW SECONDS).
***dev*.isEnabled** queries whether or not a device is enabled

---

adjust(*dev*, *value*[, *newvalue*]) adjusts the offset of the device so that either the current value or a given value becomes newvalue ("dev" must be a device that supports the "offset" parameter).

---

history(*dev[, key][, fromtime]*) prints history of a device parameter "key".
fromtime and *totime* are either numbers giving **hours** in the past, or otherwise strings with a time specification:

```
>>> history(mth)           # show value of mth in the last hour
>>> history(mth, 48)       # show value of mth in the last two days
>>> history(mtt, 'offset')  # show offset of mth in the last day

>>> history(mth, '1 day')              # allowed: d/day/days
>>> history(mth, 'offset', '1 week')   # allowed: w/week/weeks
>>> history(mth, 'speed', '30 minutes')  # allowed: m/min/minutes
>>> history(mth, 'speed', '2012-05-04 14:00')  # from that date/time on
>>> history(mth, 'speed', '14:00', '17:00')    # between 14h and 17h today
>>> history(mth, 'speed', '2012-05-04', '2012-05-08')  # between two days
```

---

getall(*parameter, ...*) lists the given parameters for all existing devices that have them.
Example: **>>>** getall('offset') lists the offset for all devices with an "offset" parameter.

setall(*param, value*) sets the given parameter to the given value for all devices that have it.
Example: **>>>** setall('offset', 0) sets the offset for all devices to zero.

b. Measuring commands

count([detectors][, presets]) performs a single counting with given detector/preset e.g.: **count(t=10)** or **count(m=10000)**, or simply **count()** using default detector/preset.

preset(preset) sets a new default preset for the current detector, e.g.: **preset(t=10) or preset(m=10000)**.

c. Scanning commands

scan(dev, start, step, numpoints) scans over device(s) and count detector(s) from start to stop with numpoints.

cscan(dev, center, step, numperside, ...) scans around a center with step size and number of points on each side.

scan(dev, [, , ,]) scans over device(s) and count detector(s) through a list of comma separated values.

contscan(dev, start, end, [speed, timedelta,]...) scans a device continuously with low speed (default is 1/5th of the current device speed) while detectors are read out every timedelta seconds (default is 1 second).

center(dev, center, step, numperside, ycol) performs a scan around center, and moves the given device to the maximum of a Gaussian (default) fit of ycol through a scan, e.g.: **center('om', 10.34, 0.1, 20, m=1000,ycol='counts')**.

Additional parameters for presets in all types of scans:
>>> scan(dev, ..., t=5) # at each scan point count for 5 seconds
>>> scan(dev, ..., mon1=1000) # at each scan point count until mon1 is 1000

d. On-line analysis commands

center_of_mass([[xcol], ycol]) calculates the center of mass x-coordinate of the last scan.
gauss([[xcol], ycol]) fits a Gaussian through the data of the last scan.

root_mean_square([ycol]) calculates the root-mean-square of the last scan.
fwhm([[xcol], ycol]) calculates an estimate of full width at half maximum.
findpeaks([[xcol,]ycol][, npoints=n]) finds peaks in the data of the last scan.

integrate(*columns) is to be used after a scan and calculates the integrated intensity of a peak in the scan.

In all the above commands it is possible to give columns by name instead of giving column numbers.
>>> center_of_mass() # uses first X column and first counter Y column (default)
>>> center_of_mass(2, 3) # uses second X column and third Y column
>>> center_of_mass('om', 'ctr1'). # uses column names

7) Crystal orientation

a. Centering reflections

In general, the following commands should list all available parameters of an experiment:
ListParams(Sample)
ListParams(ZEBRA)
ListParams(Exp)

The Sample object holds the cell parameters, the UB matrix, and a **reflist**. The **reflist** of the sample can be either of the two system lists: **ublist** (for peak search and UB refinement) or **messlist** (for reflections to be measured).
The diffractometer object holds some interesting parameters for centering reflections in **ublist**:
ZEBRA.center_counter is the counter channel to be used for centering.
ZEBRA.center_maxpts is the maximum number of points to consider for centering.
ZEBRA.center_steps is a list of step widths to use for each angle (stt, om, chi, phi) when centering reflections.

Max(dev, step, counter, maxpts, preset) finds a maximum for a peak with respect to the device given (hill climbing algorithm) using given or default preset.

Center(idx,reflist= ublist, preset) centers a reflection, preset is either given or default.

Dealing with reflection lists (**reflist = ublist in commands below may not be needed if reflist is already defined as ublist**):
ListRef(reflist= ublist) lists a reflection list.
ClearRef(reflist= ublist) removes all entries from a reflection list
LoadRef(filename, reflist= ublist) loads a reflection list from filename (format: "*h k l stt om chi phi*", angles are optional).
SaveRef(filename, reflist= ublist) saves a reflection list to filename.
AddRef(hkl, angles, reflist= ublist) adds a reflection to a reflection list,
e.g. **AddRef((2,-1,0), (stt, om, chi, phi), reflist=ublist)** or **AddRef((2,-1,0))** to use current diffractometer angles.
SetRef(idx, hkl, angles, reflist= ublist) modifies the reflection with the index idx in the reflection list.
SetAngRef(idx, angles, reflist=ublist) modifies the angles for the reflection with the index idx in the reflection list.
DelRef(idx, reflist= ublist) deletes the reflection with index idx from the reflection list.

b. Indexing reflections

IndexTH(idx, reflist= ublist, hkllim=10) suggests possible indices for the reflection with index idx in the reflection list based on twotheta (hkllim defines the range of Miller indices to consider).

SaveRef(filename, reflist= ublist, fmt='dirax') save the reflection list for use the program dirax outside of NICOS.

SetRef(idx, (h, k, l)) assigns indices to reflections.

c. UB matrix

CalcUB(idx1, idx2, replace=False) calculates a UB matrix using idx1 and idx2 reflections and eventually replace default UB matrix by the new calculated one.

CenterList(reflist, preset) centers all reflections of a list using given or default preset.

SaveRef(filename, fmt='rafin') saves the reflection list in a file.

Sample.ubmatrix=[ub11, ub12, ub13, ub21, ub22, ub23, ub31, ub32, ub33] to enter the refined UB.
RefineMatrix() uses **ublist** in order to refine the orientation matrix (by default uses all parameters – i.e. 3 orientation angles for the UB, 6 cell constants, wavelength and offsets for all used motors, and therefore may typically leads to an underconstrained fit returning TypeError M=XX N=YY). **NOT YET FUNCTIONAL, please USE RAFIN for the moment!**

Fixing some of the parameters can be done by specifying these, e.g.:
RefineMatrix(wavelength=1.178, a=5.23, b='a', c='a', alpha=90, beta=90, gamma=90, delta_stt=0, delta_om=0, delta_chi=0, delta_phi=0)
If the resulting UB consists of NaNs, something is probably wrong (lattice constants, starting UB, constraints…).

AcceptRefinement() loads the refined matrix.

d. Useful commands once UB is correct

Once a UB matrix or at least cell constants have been defined, it is possible **to drive directly in reciprocal space**:

maw(ZEBRA,(2,2,0))

CalcAng((h,k,l)) calculates the angles for a reflection assuming current UB and without driving to it.
CalcPos((stt, om, chi, phi)) calculates the reciprocal space position from the angles provided (current angles if not given).

ShowAng() shows the position of all diffractometer motors.

ScanOmega((h, k, l), preset) performs a omega scan of the reflection (h, k, l) using given or default preset
ScanT2T((h, k, l), preset) performs an omega-twotheta scan of the reflection (h, k, l) using given or default preset

8) Measuring reflection lists

The Sample object holds the cell parameters, the UB matrix, and a **reflist**. The **reflist** of the sample can be either of the two system lists: **ublist** (for peak search and UB refinement) or **messlist** (for reflections to be measured).

The diffractometer object holds some interesting parameters for centering reflections in **messlist**:
ZEBRA.scanmode is the scan mode to use for measuring reflections, i.e. omega or t2t.
ZEBRA.scansteps is the number of steps to measure for each reflection.
ZEBRA.scan_uvw are Cagliotti parameters to calculate scan widths, e.g. type **ZEBRA.scan_uvw = [0.1506, -0.4452, 0.4639]**

!! All reflection list commands take a reflist parameter. If this parameter is missing, then the command is applied to the default reflection list configured in sample.reflist.

GenerateList(dmin, dmax, reflist) generates a list of reflections with dmin and dmax denoting the limits in d for the reflections, applying systematic extinctions and symmetrical equivalents filtered according to space group.

Measure(scanmode=None, skip=0, reflist=None, preset) measures a reflection list using given or default preset.
- scanmode: either omega or t2t (or default mode in parameter to Sample)
- skip to start processing the reflection list at a certain index.
- Preset is the count preset, e.g. t=.5
- scan step and number of scan points for calculated from the parameters of the instrument object.

Use the LoadRef() command to load the list to be measured *(see above, in 7)* ***a.****, under 'Dealing with reflection lists').*

GenerateSuper(targetlist, vector, srclist) to generate incommensurate reflection lists.

9) Q scans

qscan(Q, dQ, numpoints, ...) performs a single-sided Q scan where Q and dQ are lists of 3 arguments.

example: qscan((1, 0, 0), (0.1, 0, 0), 11, monitor1=10000)

qcscan(Q, dQ, numperside, ...) performs a centered Q scan where Q and dQ are lists of 3 arguments.
example: qcscan((1, 0, 0), (0.01, 0, 0), 20, monitor1=10000)

10) Scripts

Exp.scriptpath sets the script directory (use ListParams(Exp) to list all available parameters of the experiment).

run(filename) runs a script file given by filename (absolute path, or relative to the experiment script directory).
/stop stops a running script (I ignore, H stop after current scan point, L stop after current command, S immediate stop)
pause(prompt='Script paused by pause() command.') pauses the script until the user confirms continuation. The prompt text is shown to the user.

abort(message=None) aborts any running script with a given message.

wait(*dev*, ...) waits until motion/action of one or more devices is complete.
The next example waits for the T (typically the sample temperature) and B (typically the magnetic field):
>>> wait(T, 60) *# wait for the T device, then another 60 seconds*
>>> wait(T)
>>> sleep(60)

waitfor(*dev*, *condition*[, *timeout*]) waits for a device until a condition is fulfilled.
>>> waitfor(motor, '< 10')
The supported conditions are 1:

- '<', '<=', '==', '!=', '>=', '>' with a value
- 'is False', 'is True', 'is None'
- 'in list', where list could be a Python list or tuple

An optional timeout value can be added, which denominates the maximum time the command will wait (in seconds). If the timeout value is reached, an error will be raised. The default timeout value is 86400 seconds (1 day).
Example: waitfor(T, '< 10', timeout=3600) waits a maximum of 1 hour for T to get below 10.

11) Sample environment

SetEnvironment([dev, ...])[source] selects device(s) to read during scans as "experiment environment".

AddEnvironment(dev, ...)[source] adds the specified environment device to the standard environment.

ListEnvironment()[source] lists the standard environment devices.

avg(dev)[source] creates a "statistics device" that calculates the scan-point average in order to calculate the average of a samenv device over the whole scan point, as opposed to the value at the end of the scan point.

minmax(dev)[source] creates a "statistics device" that calculates the scan-point min/maximum in order to calculate the minimum and maximum of a samenv device over the whole scan point.

Note to change the motor used as omega, type **om.alias = 'dom'** (stick motor) or **'om_raw'** (sample table).
To load a sample environment configuration in Nicos, type e.g. **frappy_main('ccr3')**. Use empty brackets to remove a sample environment configuration. For stick motors the command is e.g. **frappy_stick('')**.

Separate notes:

- If some scan data are written in *.ccl instead of *.dat files, type ZEBRA.ccl_file = False

- A threshold can be set so that the counting is stopped below a certain monitor threshold, using zebradet.threshold (use zebradet.thresholdcounter for setting the monitor number on which the threshold is applied – usually 1).

- In Eulerian geometry, Nicos works with phi limits 0 – 360 degrees while our motors have -180 – 180 degrees, which can result in certain issues. I would recommend, before starting the centering of reflections, to introduce an offset on phi so that the motor values match the range 0 – 360 degrees (e.g. by typing **phi.offset=180**).

- Exp._setROParam('forcescandata', False) is the command that fixes the problem of the failure of Center() command.

- use e.g. device.fmtstr='%.6f' to change the format of values saved in ascii files

- to communicate with sea from nicos, e.g. to switch off ccr compressor when using the JTCCR:
se_sea_main.communicate('epc port3 0')

Getting started with reflections centering, matrix calculation, matrix refinement and motor offsets (!):

Here is a quick summary of the procedure and commands needed to get an experiment started.

1) Adjust diffractometer angles until you find a reflection: double-click on a motor name in the GUI and click on 'move' to set a new target value, or in command line use e.g. maw('om',23) to move omega to 23 degrees. Stop when you are roughly on a reflection according to the voltmeter of the detector. Type ShowAng() if you want to see the diffractometer angles.

2) Clear the list of reflections if you start a new experiment, typing ClearRef(reflist=ublist), then add to the ublist the reflection you think the instrument is on, using e.g. AddRef((0,0,2)) (the current diffractometer angles are taken if they are not given in the AddRef command). The reflection (0,0,2) has index 0 in ublist.

3) Center all motors in appropriate order (om, stt, chi, phi in Eulerian or om, stt, nu in normal beam) and with appropriate steps (respectively 0.1, 0.3, 1, 0.1 in Eulerian or 0.1, 0.3, 1 in normal beam). This is done using either Center() or center():
- all angles sequentially using e.g. Center(0, m=10000) after setting the steps e.g. ZEBRA.center_steps = [0.1, 0.3, 0.5, 0.1];
- one angle after the other, using e.g. center('om', 14.5, 0.1, 8, m=10000, ycol='counts').

4) Once a reflection is centered, adjust its angles to the current diffractometer values using e.g. SetAng(0), or alternatively display the angles with ShowAng() and then copy values to SetRef(0, (0,0,2), (28.199, 14.455, 179.647, 82.676)). The last command is interesting if you want to change the Miller indices of a reflection in the ublist. Repeat steps 1-4 with a second and non-equivalent direction.

5) Make sure Nicos knows what your sample is, e.g. ClearSamples() and then:
Sample.new({'name': 'ZrMnGaSn', 'a': 9.19, 'a': b, 'c': 5.62, 'bravais': 'P', 'laue': '6/mmm'})
so that you can calculate a first UB matrix based e.g. on the first two reflections using CalcUB(0, 1, replace=True).

6) You may now be able to drive to reflections, e.g. maw(ZEBRA, (0,1,0)) and repeat the centering procedure with AddRef(), Center() and then SetRef() or SetAng() for each reflection, until at least about 10 reflections are in the ublist:

```
[13:48:45] >>> [user 2022-05-27 13:48:45]  ListRef()
[13:48:45]   #      H      K      L     STT     OM    CHI     PHI
[13:48:45]   0  0.0000  0.0000  2.0000   28.199  14.455 179.647  82.676
[13:48:45]   1  1.0000  1.0000  0.0000   17.110   8.581 175.401 172.763
[13:48:45]   2  0.0000  1.0000  0.0000    9.763   5.198 147.360 173.414
[13:48:45]   3 -1.0000  0.0000  0.0000    9.786   4.845 157.635  -7.899
[13:48:45]   4  1.0000  0.0000  0.0000   10.023   4.840 204.246 172.124
[13:48:45]   5 -5.0000  0.0000  0.0000   51.354  25.570 155.577  -7.845
[13:48:45]   6  0.0000  0.0000  4.0000   58.572  29.325 179.567  82.687
[13:48:45]   7 -2.0000  2.0000  0.0000   19.681  11.133  96.374 -13.278
[13:48:45]   8 -3.0000  2.0000  0.0000   26.345  14.027 115.177  -8.588
```

7) Copy the lines of the ublist above into a file rafin.dat (Eulerian geometry) or rafnb.dat (normal beam geometry), in your directory (make sure the wavelength and lattice parameters are set properly, and free parameters with flag 1 on lines 4-5):

```
HKL: 0 0 2,  ZrMnGaSn
2 1 0 0 45 3 4 1 .5 0
0  1.383
1 .0 1 .0 1 .0
1 9.19 2 9.19 1 5.62 0 90 0 90 0 120
  0.0000   0.0000   2.0000   28.199   14.455  179.647   82.676
  1.0000   1.0000   0.0000   17.110    8.581  175.401  172.763
  0.0000   1.0000   0.0000    9.763    5.198  147.360  173.414
 -1.0000   0.0000   0.0000    9.786    4.845  157.635   -7.899
  1.0000   0.0000   0.0000   10.023    4.840  204.246  172.124
 -5.0000   0.0000   0.0000   51.354   25.570  155.577   -7.845
  0.0000   0.0000   4.0000   58.572   29.325  179.567   82.687
 -2.0000  2.0000  0.0000   19.681  11.133  96.374 -13.278
 -3.0000  2.0000  0.0000   26.345  14.027 115.177  -8.588
 -2.0000  3.0000  0.0000   26.221  13.648 104.147 174.620

-1
```

The above file may also be written by Nicos using SaveRef(filename, fmt='rafin') (not tested…).

8) Run the rafin/rafnb program from a terminal from your directory, using e.g. rafin > rafin.lis, and inspect the refinement results in the output (rafin.lis or rafnb.lis):

```
0RESULTS AT THE END OF THE LAST CYCLE (no :  4)
 ------------------------------------------------------
   H   K   L          2THETA                  OMEGA                    CHI
                obs     cal     diff      obs     cal     diff      obs     cal     diff  Warning   PSI
 ------------------------------------------------------------------------------------------------------------------------------------------
  0.0  0.0  2.0    28.199   28.238   -0.039    14.455   14.177    0.278   179.647  179.880   -0.2
  1.0  1.0  0.0    17.110   17.113   -0.003     8.581    8.610   -0.029   175.401  175.436   -0.0
  0.0  1.0  0.0     9.763    9.788   -0.025     5.198    5.141    0.057   147.360  145.439    1.9
 -1.0  0.0  0.0     9.786    9.788   -0.002     4.845    4.925   -0.080   157.635  156.323    1.3
  1.0  0.0  0.0    10.023    9.788    0.235     4.840    4.946   -0.106  -155.754 -154.565   -1.1
 -5.0  0.0  0.0    51.354   51.213    0.141    25.570   25.687   -0.117   155.577  156.323   -0.7
  0.0  0.0  4.0    58.572   58.595   -0.023    29.325   29.366   -0.041   179.567  179.880   -0.3
 -2.0  2.0  0.0    19.681   19.810   -0.129    11.133   11.203   -0.070    96.374   96.402   -0.0
 -3.0  2.0  0.0    26.345   26.360   -0.015    14.027   14.148   -0.121   115.177  115.448   -0.2
 -2.0  3.0  0.0    26.221   26.360   -0.139    13.648   13.415    0.233   104.147  104.560   -0.4
 ------------------------------------------------------------------------------------------------------------------------------------------
 Mean abs(DEV.) (no *false* ref.):      0.0751                   0.1133                   0.6462
0-------------------------------------------------------------------------
0PARAMETER    VALUE  (ST.DEV.)      CORRELATIONS
    1 =    -0.1578(   0.3493)     1.00  0.24  0.01  0.80  0.61  0.12  0.51  0.03
    2 =     0.8805(   0.5057)     0.24  1.00 -0.03  0.32  0.11 -0.04  0.32 -0.00
    3 =     0.8792(   0.1976)     0.01 -0.03  1.00  0.02 -0.00 -0.38 -0.08  0.68
    4 =     9.2111(   0.1280)     0.80  0.32  0.02  1.00  0.52  0.07  0.68  0.03
    5 =     5.6386(   0.0621)     0.61  0.11 -0.00  0.52  1.00  0.27  0.21  0.01
    6 = -2.988E-01(  3.043E-01)   0.12 -0.04 -0.38  0.07  0.27  1.00  0.05 -0.25
    7 = -1.404E-01(  3.290E-01)   0.51  0.32 -0.08  0.68  0.21  0.05  1.00 -0.03
    8 =  7.419E-01(  2.912E-01)   0.03 -0.00  0.68  0.03  0.01 -0.25 -0.03  1.00
0-------------------------------------------------------------------------
0FINAL ORIENTATION [UB] MATRIX
   0.1132041    0.1016310   -0.0198335
  -0.0136560   -0.0101639   -0.1762079
  -0.0520916    0.0726844    0.0030918
 SUMMARY OF ZERO OFFSETS
   2THETA offset =    0.158 deg
   OMEGA  offset =   -0.881 deg
   CHI    offset =   -0.879 deg
0=========================================================================
0DIRECT CELL   9.2111   9.2111   5.6386     90.0000  90.0000 120.0000
0RECIP. CELL  0.125360 0.125360 0.177348     90.0000  90.0000  60.0000
0DIRECT VOLUME       414.3092
0=========================================================================
```

9) In the example above the refinement of the orientation matrix includes cell parameters and offsets of diffractometer angles. As this looks good we can then input the refinement results into Nicos:

Sample.ubmatrix=[0.1132041 , 0.1016310 , -0.0198335, -0.0136560, -0.0101639, -0.1762079, -0.0520916, 0.0726844, 0.0030918]
Sample.a=9.2111
Sample.b=9.2111
Sample.c=5.6386
stt.offset=0.1578
om.offset=-0.881
chi.offset=-0.879

Note that the correct signs for the offsets to be implemented in Nicos are those found under "SUMMARY OF ZERO OFFSETS" in the output of rafin and <u>not</u> those in the table below the last refinement cycle.

<u>Getting started with data collections (!)</u>:

Once the UB is correct, and Zebra being a diffractometer, you certainly want to collect datasets!

1) Generate relevant lists of reflections using SXtalRefGen (either command line in terminal – see explanations here, or using pyzebra.psi.ch under the 'ccl prepare' tab).

2) Define the parameters relevant for the scans, i.e. the type of scan (omega or t2t), the total number of points in each scan, the UVW parameters used to calculate the two-theta dependence of the instrument resolution, and the number of times the instrument resolution a scan should span. **The parameters below give a constant scan step of 0.1 degree (increase u 0.5 to 1 for a scan step of 0.2 degree for example)**.

```
ZEBRA.scanmode='omega'
ZEBRA.scansteps=20
ZEBRA.scan_uvw=[0.5, 0, 0]
ZEBRA.resolution_multiplier=4
```

3) Load a list of reflection and assign it to messlist, e.g.:

```
LoadRef('/home/zebralnsg/sibille/2022/ZrMnGaSn/Zr_1.mhkl', reflist='messlist')
```

4) Collect all reflections of the given messlist using a specific monitor for each point:

```
Measure(reflist='messlist', m=80000)
```

Getting started with scripts for parametric studies: just some examples of typical scripts with loops.

Example 1:

```
maw(magfield, 0)

LoadRef('/home/zebralnsg/willwater/UNi4B_may2022/nuclear_1.hkl',reflist='messlist')
Measure(reflist='messlist',m=10000)

LoadRef('/home/zebralnsg/willwater/UNi4B_may2022/magnetic_1.mhkl',reflist='messlist')
Measure(reflist='messlist',m=50000)

for i in range(0, 105, 5):
        maw(magfield, i/10)
        maw(ZEBRA,(2,1.333,0))
        cscan('om',-113.3,0.1,15,m=50000)

LoadRef('/home/zebralnsg/willwater/UNi4B_may2022/magnetic_1.mhkl',reflist='messlist')
Measure(reflist='messlist',m=50000)

LoadRef('/home/zebralnsg/willwater/UNi4B_may2022/nuclear_1.hkl',reflist='messlist')
Measure(reflist='messlist',m=10000)
```

Example 2:

```
maw('se_ts', 1.0)

for i in [9,8,7.75,7.5,7.25,7,6.75,6.5,6,5,4,3,2,1,0]:

        maw(magfield, i)

        maw(ZEBRA,(0,1.333,0))
        cscan('om',-46.72,0.1,10,m=50000)

        maw(ZEBRA,(2,1.333,0))
        cscan('om',-113.27,0.1,10,m=50000)
```

Alternatively, on can also import numpy in a script, for instance to use the arange and linspace functions to generate lists of fields or temperatures:

```
import numpy as np
```

```
print(np.arange(0,10.1,0.1))
print(np.linspace(0,10,101))
```

Example 3: a possible script for manually operating on a list of external reflections, e.g. to substitute Measure()