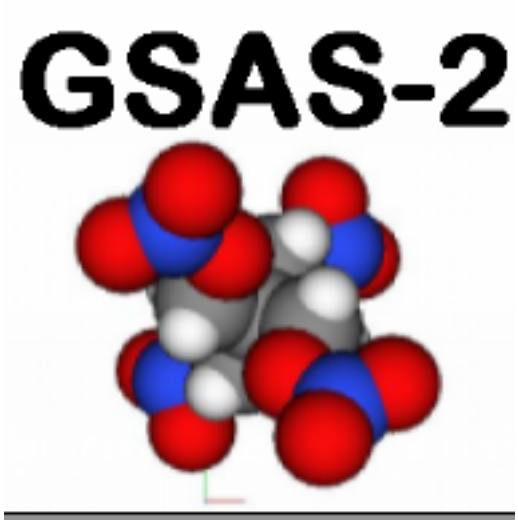

GSAS-II Developers Documentation

Release 0.2.0

Robert B. Von Dreele and Brian H. Toby

October 17, 2013

CONTENTS

CHAPTER

ONE

GSAS-II MAIN MODULE

Main routines for the GSAS-II program

class GSASII.**GSASII**(*parent*)
Define the main GSAS-II frame and its associated menu items

CheckNotebook()
Make sure the data tree has the minimally expected controls. (BHT) correct?

class **ConstraintDialog**(*parent*, *title*, *text*, *data*, *separator='*'*)
Window to edit Constraint values

class GSASII.**CopyDialog**(*parent*, *title*, *text*, *data*)
Creates a dialog for copying control settings between data tree items

GSASII.**ErrorDialog**(*title*, *message*, *parent=None*, *wtype=4*)
Display an error message

GSASII.**ExitMain**(*event*)
Called if the main window is closed

GSASII.**FillMainMenu**(*menubar*)
Define contents of the main GSAS-II menu for the (main) data tree window in the mac, used also for the data item windows as well.

GSASII.**GetFileList**(*fileType*, *skip=None*)
Appears unused. Note routine of same name in GSASIIpwdGUI

GSASII.**GetHKLFdatafromTree**(*HKLFname*)
Returns single crystal data from GSASII tree

Parameters **HKLFname** (*str*) – a single crystal histogram name as obtained from GSASIIstruct.GetHistogramNames()

Returns HKLFdata = single crystal data list of reflections

GSASII.**GetPWDRdatafromTree**(*PWDRname*)
Returns powder data from GSASII tree

Parameters **PWDRname** (*str*) – a powder histogram name as obtained from GSASIIstruct.GetHistogramNames()

Returns PWDRdata = powder data dictionary with Powder data arrays, Limits, Instrument Parameters, Sample Parameters

GSASII.**GetPhaseData**()
Returns a list of defined phases. Used only in GSASIIgrid Note routine GSASIIstruct.GetPhaseData() also exists.

GSASII.**GetPowderIparm**(*rd*, *prevIparm*, *lastIparmfile*, *lastdatafile*)
Open and read an instrument parameter file for a data file Returns the list of parameters used in the data tree

Parameters

- **rd** (*obj*) – the raw data (histogram) data object.
- **prevIparm** (*str*) – not used
- **lastIparmfile** (*str*) – Name of last instrument parameter file that was read, or a empty string.
- **lastdatafile** (*str*) – Name of last data file that was read.

Returns a list of two dicts, the first containing instrument parameters and the second used for future TOF datasets (timemaps?)

GSASII.**GetUsedHistogramsAndPhasesfromTree**()
Returns all histograms that are found in any phase and any phase that uses a histogram :returns: two dicts:

•Histograms = dictionary of histograms as {name:data,...}

•Phases = dictionary of phases that use histograms

GSASII.**OnAddPhase**(*event*)
Add a new, empty phase to the tree. Called by Data/Add Phase menu

GSASII.**OnDataDelete**(*event*)
Delete one or more histograms from data tree. Called by the Data/DeleteData menu

GSASII.**OnDeletePhase**(*event*)
Delete a phase from the tree. Called by Data/Delete Phase menu

GSASII.**OnDummyPowder**(*event*)
Called in response to Import/Powder Data/Simulate menu item to create a Dummy powder diffraction data set.

Reads an instrument parameter file and then gets input from the user

GSASII.**OnFileClose**(*event*)
Clears the data tree in response to the File/Close Project menu button. User is given option to save the project.

GSASII.**OnFileExit**(*event*)
Called in response to the File/Quit menu button

GSASII.**OnFileOpen**(*event*, *filename=None*)
Reads in a GSAS-II .gpx project file in response to the File/Open Project menu button

GSASII.**OnFileSave**(*event*)
Save the current project in response to the File/Save Project menu button

GSASII.**OnFileSaveas**(*event*)
Save the current project in response to the File/Save as menu button

GSASII.**OnImageRead**(*event*)
Called to read in an image in any known format

GSASII.**OnImageSum**(*event*)
Sum together image data(?)

GSASII.**OnImportGeneric**(*reader*, *readerlist*, *label*, *multiple=False*)
Used to import Phases, powder dataset or single crystal datasets (structure factor tables) using

reader objects subclassed from GSASIIIO.ImportPhase, GSASIIIO.ImportStructFactor or GSASIIIO.ImportPowderData. If a reader is specified, only that will be attempted, but if no reader is specified, every one that is potentially compatible (by file extension) will be tried on the selected file(s).

Parameters

- **reader** (*readerobject*) – This will be a reference to a particular object to be used to read a file or None, if every appropriate reader should be used.
- **readerlist** (*list*) – a list of reader objects appropriate for the current read attempt. At present, this will be either self.ImportPhaseReaderlist, self.ImportSfactReaderlist or self.ImportPowderReaderlist (defined in _init_Imports from the files found in the path), but in theory this list could be tailored. Used only when reader is None.
- **label** (*str*) – string to place on the open file dialog: Open *label* input file
- **multiple** (*bool*) – True if multiple files can be selected in the file dialog. False is default. At present True is used only for reading of powder data.

Returns a list of reader objects (rd_list) that were able to read the specified file(s). This list may be empty.

GSASII.**OnImportPhase**(*event*)
Called in response to an Import/Phase/... menu item to read phase information. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

GSASII.**OnImportPowder**(*event*)
Called in response to an Import/Powder Data/... menu item to read a powder diffraction data set. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

Also reads an instrument parameter file for each dataset.

GSASII.**OnImportSfact**(*event*)
Called in response to an Import/Structure Factor/... menu item to read single crystal datasets. dict self.ImportMenuId is used to look up the specific reader item associated with the menu item, which will be None for the last menu item, which is the "guess" option where all appropriate formats will be tried.

GSASII.**OnMakePDFs**(*event*)
Calculates PDFs

GSASII.**OnPatternTreeItemActivated**(*event*)
Called when a tree item is activated

GSASII.**OnPatternTreeItemCollapsed**(*event*)
Called when a tree item is collapsed

GSASII.**OnPatternTreeItemDelete**(*event*)
Called when a tree item is deleted – not sure what this does

GSASII.**OnPatternTreeItemExpanded**(*event*)
Called when a tree item is expanded

GSASII.**OnPatternTreeKeyDown**(*event*)
Not sure what this does

GSASII.**OnPatternTreeSelChanged**(*event*)
Called when a data tree item is selected

GSASII.**OnPwdrSum**(*event*)
Sum together powder data(?)

GSASII.**OnReadPowderPeaks**(*event*)
Bound to menu Data/Read Powder Peaks – still needed?

GSASII.**OnRefine**(*event*)
Perform a refinement. Called from the Calculate/Refine menu.

GSASII.**OnRenameData**(*event*)
Renames an existing phase. Called by Data/Rename Phase menu

GSASII.**OnSeqRefine**(*event*)
Perform a sequential refinement. Called from the Calculate/Sequential refine menu.

GSASII.**OnSize**(*event*)
Called when the main window is resized. Not sure why

GSASII.**OnViewLSParms**(*event*)
Displays a window showing all parameters in the refinement. Called from the Calculate/View LS Parms menu.

GSASII.**ReadPowderInstprm**(*instfile*)
Read a GSAS-II (new) instrument parameter file

Parameters **instfile** (*str*) – name of instrument parameter file

GSASII.**ReadPowderIparm**(*instfile*, *bank*, *databanks*, *rd*)
Read a GSAS (old) instrument parameter file

Parameters

- **instfile** (*str*) – name of instrument parameter file
- **bank** (*int*) – the bank number read in the raw data file
- **databanks** (*int*) – the number of banks in the raw data file. If the number of banks in the data and instrument parameter files agree, then the sets of banks are assumed to match up and bank is used to select the instrument parameter file. If not, the user is asked to make a selection.
- **rd** (*obj*) – the raw data (histogram) data object. This sets rd.instbank.

class GSASII.**SumDialog**(*parent*, *title*, *text*, *dataType*, *data*)
Allows user to supply scale factor(s) when summing data

class GSASII.**ViewParmDialog**(*parent*, *title*, *parmDict*)
Window to show all parameters in the refinement. Called from `OnViewLSParms()`

class GSASII.**GSASIImain**(*redirect=True*, *filename=None*, *useBestVisual=False*, *clearSigInt=True*)
Defines a wxApp for GSAS-II

Creates a wx frame (self.main) which contains the display of the data tree.

MacOpenFile(*filename*)
Called on Mac every time a file is dropped on the app when it is running, treat this like a File/Open project menu action. Should be ignored on other platforms

OnInit()
Called automatically when the app is created.

GSASII.**main**()
Start up the GSAS-II application

CHAPTER

TWO

GSASIIOBJ: DATA OBJECTS

This module defines and/or documents the data structures used in GSAS-II.

2.1 Constraints Tree Item

Constraints are stored in a dict, separated into groups. Note that parameter are named in the following pattern, p:h:<var>:n, where p is the phase number, h is the histogram number <var> is a variable name and n is the parameter number. If a parameter does not depend on a histogram or phase or is unnumbered, that number is omitted. Note that the contents of each dict item is a List where each element in the list is a *constraint definition objects*.

The keys in the Constraints dict are:

key	explanation
Hist	This specifies a list of constraints on histogram-related parameters, which will be of form :h:<var>:n.
HAP	This specifies a list of constraints on parameters that are defined for every histogram in each phase and are of form p:h:<var>:n.
Phase	This specifies a list of constraints on phase parameters, which will be of form p::<var>:n.
Global	This specifies a list of constraints on parameters that are not tied to a histogram or phase and are of form ::<var>:n

Each constraint is defined as a list using a series of terms of form

```
[[<mult1>, <var1>], [<mult2>, <var2>],..., <fixed val>, <vary flag>, <cons type>]
```

Where the variable pair list item containing two values [<mult>, <var>],

- <mult> is a multiplier for the constraint (float)
- <var> is the name of the variable (str) (or to be implemented a `VarName` object.)

Note that the last three items in the list play a special role:

- <fixed val> is the fixed value for a constraint equation or is None
- <vary flag> is True, False or None and is intended to use to indicate if new variables should be refined.
- <cons type> is one of four letters, 'e', 'c', 'h', 'f' that determines the type of constraint.
  - 'e' defines a set of equivalent variables. Only the first variable is refined (if the appropriate refine flag is set) and and all other equivalent variables in the list are generated from that variable. The vary flag for those variables is ignored.
  - 'c' defines a constraint equation of form, $m_1 \times var_1 + m_2 \times var_2 + ... = c$

- 'h' defines a variable to hold (not vary). Any variable on this list is not varied, even if its refinement flag is set. This is of particular value when needing to hold one or more variables in a set such as the reciprocal metric tensor or anisotropic displacement parameter.
- 'f' defines a relationship to define a new variable according to relationship $newvar = m_1 \times var_1 + m_2 \times var_2 + ...$

2.2 Covariance Tree Item

The Covariance tree item has results from the last least-squares run. They are stored in a dict with these keys:

key	sub-key	explanation
newCellDict		dict with lattice parameters computed by `GSASIIstrMath.GetNewCellParms()` (dict)
title		Name of gpx file(?) (str)
variables		Values for all N refined variables (list of float values, length N, ordered to match varyList)
sig		Uncertainty values for all N refined variables (list of float values, length N, ordered to match varyList)
varyList		List of directly refined variables (list of str values, length N)
newAtomDict		dict with atom position values computed in `GSASIIstrMath.ApplyXYZshifts()` (dict)
Rvals		R-factors, GOF, Marquardt value for last refinement cycle (dict)
	Nobs	Number of observed data points (int)
	Rwp	overall weighted profile R-factor (%, float)
	chisq	sum[w*(Iobs-Icalc)**2] for all data note this is not the reduced chi squared (float)
	lamMax	Marquardt value applied to Hessian diagonal (float)
	GOF	The goodness-of-fit, aka square root of the reduced chi squared. (float)
covMatrix		The (NxN) covVariance matrix (np.array)

2.3 Phase Tree Items

Phase information is stored in the GSAS-II data tree as children of the Phases item in a dict with keys:

key	sub-key	explanation
General		Overall information for the phase (dict)
	AtomPtrs	list of four locations to use to pull info from the atom records (list)
	F000X	x-ray F(000) intensity (float)
	F000N	neutron F(000) intensity (float)
	Mydir	directory of current .gpx file (str)
	MCSA controls	?
	Cell	List with 7 items: cell refinement flag (bool) a, b, c, (Angstrom, float) alpha, beta & gamma (degrees, float)
	Type	for now 'nuclear' (str)
	Map	dict of map parameters
	SH Texture	dict of spherical harmonic preferred orientation parameters
	Isotope	dict of isotopes for each atom type
	Isotopes	dict of scattering lengths for each isotope combination for each element in phase

Continued on next page

Table 2.1 – continued from previous page

key	sub-key	explanation
	Name	phase name (str)
	SGData	Space group details as a *space group (SGData) object* as defined in `GSASIIspc.SpcGroup()`.
	Pawley neg wt	Restraint value for negative Pawley intensities (float)
	Flip	Charge flip controls dict?
	Data plot type	?
	Mass	Mass of unit cell contents in g/mol
	POhkl	March-Dollase preferred orientation direction
	Z	?
	vdWRadii	?
	Color	Colors for atoms (list of (r,b,g) triplets)
	AtomTypes	List of atom types
	AtomMass	List of masses for atoms
	doPawley	Flag for Pawley intensity extraction (bool)
	NoAtoms	Number of atoms per unit cell of each type (dict)
	Pawley dmin	maximum Q (as d-space) to use for Pawley extraction (float)
	BondRadii	Radius for each atom used to compute interatomic distances (list of floats)
	AngleRadii	Radius for each atom used to compute interatomic angles (list of floats)
ranId		unique random number Id for phase (int)
pId		Phase Id number for current project (int).
Atoms		Atoms in phase as a list of lists. The outer list is for each atom, the inner list contains varying items depending on the type of phase, see the *Atom Records* description. (list of lists)
Drawing		Display parameters (dict)
	ballScale	Size of spheres in ball-and-stick display (float)
	bondList	dict with bonds
	contourLevel	? (float)
	showABC	Flag to show view point triplet (bool). True=show.
	viewDir	cartesian viewing direction (np.array with three elements)
	Zclip	clipping distance in A (float)
	backColor	background for plot as and R,G,B triplet (default = [0, 0, 0], black). (list with three atoms)
	selectedAtoms	List of selected atoms (list of int values)
	showRigidBodies	Flag to highlight rigid body placement
	sizeH	Size ratio for H atoms (float)
	bondRadius	Size of binds in A (float)
	atomPtrs	? (list)
	viewPoint	list of lists. First item in list is [x,y,z] in fractional coordinates for the center of the plot. Second item ?.
	showHydrogen	Flag to control plotting of H atoms.
	unitCellBox	Flag to control display of the unit cell.
	ellipseProb	Probability limit for display of thermal ellipsoids in % (float).
	vdwScale	Multiplier of van der Waals radius for display of vdW spheres.
	Atoms	A list of lists with an entry for each atom that is plotted.
	Zstep	Step to de/increase Z-clip (float)
	Quaternion	Viewing quaternion (4 element np.array)
	radiusFactor	Distance ratio for searching for bonds. ? Bonds are located that are within r(Ra+Rb) and (Ra+Rb)/r where Ra and Rb are the atomic radii.

Continued on next page

Table 2.1 – continued from previous page

key	sub-key	explanation
	oldxy	? (list with two floats)
	cameraPos	Viewing position in A for plot (float)
	depthFog	? (bool)
RBModels		Rigid body assignments (note Rigid body definitions are stored in their own main top-level tree entry.)
Pawley ref		Pawley reflections
Histograms		A dict of dicts. The key for the outer dict is the histograms tied to this phase. The inner dict contains the combined phase/histogram parameters for items such as scale factors, size and strain parameters. (dict)
MCSA		Monte-Carlo simulated annealing parameters

2.4 Space Group Objects

Space groups are interpreted by `GSASIIspc.SpcGroup()` and the information is placed in a SGdata object, which is a dict with these keys:

key	explanation
SpGrp	space group symbol (str)
Laue	one of the following 14 Laue classes: -1, 2/m, mmm, 4/m, 4/mmm, 3R, 3mR, 3, 3m1, 31m, 6/m, 6/mmm, m3, m3m (str)
SGInv	True if centrosymmetric, False if not (bool)
SGLatt	Lattice centering type. Will be one of P, A, B, C, I, F, R (str)
SGUniq	unique axis if monoclinic. Will be a, b, or c for monoclinic space groups. Will be blank for non-monoclinic. (str)
SGCen	Symmetry cell centering vectors. A (n,3) np.array of centers. Will always have at least one row: `np.array([[0, 0, 0]])`
SGOps	symmetry operations as a list of form `[[M1,T1], [M2,T2],...]` where M_n is a 3x3 np.array and T_n is a length 3 np.array. Atom coordinates are transformed where the Asymmetric unit coordinates [X is (x,y,z)] are transformed using $X' = M_n * X + T_n$
SGSys	symmetry unit cell: type one of 'triclinic', 'monoclinic', 'orthorhombic', 'tetragonal', 'rhombohedral', 'trigonal', 'hexagonal', 'cubic' (str)
SGPolax	Axes for space group polarity. Will be one of '', 'x', 'y', 'x y', 'z', 'x z', 'y z', 'xyz'. In the case where axes are arbitrary '111' is used (P 1, and ?).

2.5 Atom Records

If `phasedict` points to the phase information in the data tree, then atoms are contained in a list of atom records (list) in `phasedict['Atoms']`. Also needed to read atom information are four pointers, `cx,ct,cs,cia = phasedict['General']['AtomPtrs']`, which define locations in the atom record, as shown below.

location	explanation
cx,cx+1,cx+2	the x,y and z coordinates
cx+3	fractional occupancy (also cs-1)
ct-1	atom label
ct	atom type
ct+1	refinement flags
cs	site symmetry string
cs+1	site multiplicity
cia	ADP flag: Isotropic ('I') or Anisotropic ('A')
cia+1	Uiso
cia+2...cia+6	U11, U22, U33, U12, U13, U23

2.6 Powder Diffraction Tree Items

Every powder diffraction histogram is stored in the GSAS-II data tree with a top-level entry named beginning with the string "PWDR ". The diffraction data for that information are directly associated with that tree item and there are a series of children to that item. The routine `GetUsedHistogramsAndPhasesfromTree()` will load this information into a dictionary where the child tree name is used as a key, and the information in the main entry is assigned a key of `Data`, as outlined below.

key	sub-key	explanation
Limits		A list of two two element lists, as [[Ld,Hd],[L,H]] whe the current and default lowest two-theta value to be use and Hd are the current and default highest two-theta valu
Reflection Lists		A dict with an entry for each phase in the histogram. The dict item is a list of reflections as described in the *Pow* description.
Instrument Parameters		A list containing two dicts where the possible keys in eac below. The value for each item is a list containing three v value, the current value and a refinement flag which can True, False or 0 where 0 indicates a value that cannot first and second values are floats unless otherwise note first dict are noted as [1]
	Lam	Specifies a wavelength in Angstroms [1]
	Lam1	Specifies the primary wavelength in Angstrom, when an source is used [1]
	Lam2 I(L2)/I(L1)	Specifies the secondary wavelength in Angstrom, when an source is used [1] Ratio of Lam2 to Lam1 [1]
	Type	**Histogram type (str) [1]:** • 'PXC' for constant wavelength x-ray • 'PNC' for constant wavelength neutron • 'PNT' for time of flight neutron
	Zero	Two-theta zero correction in *degrees* [1]
	Azimuth	Azimuthal setting angle for data recorded with differin [1]
	U, V, W	Cagliotti profile coefficients for Gaussian instrumen where the FWHM goes as $U\tan^2\theta + V\tan\theta + W$ [1]
	X, Y	Cauchy (Lorentzian) instrumental broadening coefficient
	SH/L	Variant of the Finger-Cox-Jephcoat asymmetric peak b Note that this is the average between S/L and H/L wh height, H is the slit height and L is the goniometer diame
	Polariz.	Polarization coefficient. [1]
wtFactor		A weighting factor to increase or decrease the leverage o togram (float). A value of 1.0 weights the data with their tainties and a larger value increases the weighting of the to decreasing the uncertainties).
Sample Parameters		Specifies a dict with parameters that describe how the lected, as listed below. Refinable parameters are a list c and a bool, where the second value specifies if the value erwise the value is a float unless otherwise noted.
	Scale	The histogram scale factor (refinable)
	Absorption	The sample absorption coefficient as μr where r is the ra
	DisplaceX, DisplaceY	Sample displacement from goniometer center where Y is direction and X is perpendicular. Units are μm (refinable
	Phi, Chi, Omega	Goniometer sample setting angles, in degrees.
	Gonio. radius	Radius of the diffractometer in mm
	InstrName	A name for the instrument, used in preparing a CIF (str).
	Force, Temperature, Humidity, Pressure, Voltage	Variables that describe how the measurement was perfo directly in any computations.
	ranId	The random-number Id for the histogram (same value as key is ranId)
	Type	Type of diffraction data, may be 'Debye-Scherrer' or 'B (str).
	Diffuse	not in use?
hId		The number assigned to the histogram when the proje edited (can change)
ranId		A random number id for the histogram that does not cha
Background		The background is stored as a list with where the first

2.7 Powder Reflection Data Structure

For every phase in a histogram, the `Reflection Lists` value is a list of reflections. The items in that list are documented below.

index	explanation
0,1,2	h,k,l (float)
3	multiplicity
4	d-space, Angstrom
5	pos, two-theta
6	sig, Gaussian width
7	gam, Lorenzian width
8	F^2_{obs}
9	F^2_{calc}
10	reflection phase, in degrees
11	the equivalent reflections as a (m x 3) np.array, where m is 0.5 * multiplicity. Note that Freidel pairs, (-h,-k-,l), are not included.
12	phase shift for each of the equivalent reflections as a length (m) array
13	intensity correction for reflection, this times F^2_{obs} or F^2_{calc} gives Iobs or Icalc
14	dict with the form factor (f or b) by atom type symbol at the reflection position.

2.8 Single Crystal Tree Items

Every single crystal diffraction histogram is stored in the GSAS-II data tree with a top-level entry named beginning with the string "HKLF ". The diffraction data for that information are directly associated with that tree item and there are a series of children to that item. The routine `GetUsedHistogramsAndPhasesfromTree()` will load this information into a dictionary where the child tree name is used as a key, and the information in the main entry is assigned a key of `Data`, as outlined below.

key	sub-key	explanation
Data		A list of lists, where each inner item is an individual reflection as described in the *Single Crystal Reflections* description.
Instrument Parameters		A list containing two dicts where the possible keys in each dict are listed below. The value for most items is a list containing two values: the initial value, the current value. The first and second values are floats unless otherwise noted.
	Lam	Specifies a wavelength in Angstroms (two floats)
	Type	**Histogram type (two str values):** • 'SXC' for constant wavelength x-ray • 'SNC' for constant wavelength neutron • 'SNT' for time of flight neutron
	InstrName	A name for the instrument, used in preparing a CIF (str).
wtFactor		A weighting factor to increase or decrease the leverage of data in the histogram (float). A value of 1.0 weights the data with their standard uncertainties and a larger value increases the weighting of the data (equivalent to decreasing the uncertainties).
hId		The number assigned to the histogram when the project is loaded or edited (can change)

2.9 Single Crystal Reflection Data Structure

For every phase in a histogram, the `Reflection Lists` value is a list of reflections. The items in that list are documented below.

index	explanation
0,1,2	h,k,l (float)
3	multiplicity
4	d-space, Angstrom
5	F^2_{obs}
6	$\sigma(F^2_{obs})$
7	F^2_{calc}
8	$F^2_{obs}T$
9	$F^2_{calc}T$
10	reflection phase, in degrees
11	the equivalent reflections as a (m x 3) np.array.
12	phase shift for each of the equivalent reflections as a length (m) array
13	intensity correction for reflection, this times F^2_{obs} or F^2_{calc} gives Iobs or Icalc (not used in single crystals?)
14	dict with the form factor (f or b) by atom type symbol at the reflection position.

2.10 *Classes and routines*

`GSASIIobj.`**`LoadHistogramIDs`**(*histList*, *idList*)
Save the Id values for a series of histograms

class `GSASIIobj.`**`VarName`**(**args*)
Defines a GSAS-II variable either using the phase/atom/histogram unique Id numbers or using a character string that specifies variables by phase/atom/histogram number (which can change). Note that `LoadID()` should be used to (re)load the current Ids before creating or later using the VarName object.

A `VarName` object can be created with a single parameter:

Parameters **varname** (*str*) –

a single value can be used to create a `VarName` object. The string must be of form "p:h:var" or "p:h:var:a", where

- p is the phase number (which may be left blank);
- h is the histogram number (which may be left blank);
- a is the atom number (which may be left blank in which case the third colon is omitted).

Alternately, a `VarName` object can be created with exactly four positional parameters:

Parameters

- **phasenum** (*int*) – The number for the phase
- **histnum** (*int*) – The number for the histogram
- **varname** (*str*) – a single value can be used to create a `VarName`
- **atomnum** (*int*) – The number for the atom

`fullDescr`()
Return a longer description for a GSAS-II variable

Returns a short description or ‘no definition’ if not found

getDescr()
Return a short description for a GSAS-II variable

Returns a short description or ‘no definition’ if not found

name()
Formats the GSAS-II variable name as a “traditional” string (p:h:<var>:a)

Returns the variable name as a str

re = <module ‘re’ from ‘/Library/Frameworks/Python.framework/Versions/7.1/lib/python2.7/re.pyc’>

CHAPTER

THREE

GSAS-II UTILITY MODULES

3.1 GSASIIdata: Data for computations

At present this module defines one dict, `ramachandranDist`, which contains arrays for All and specific amino acids

3.2 ElementTable: Periodic Table Data

Element table data for building periodic table with valences & JMOL colors. Need these in case we go back to this periodic table coloring scheme.

Defines list `ElTable` which contains all defined oxidation states for each element, the location in the table, an element name, a color, a size and a second color.

3.3 FormFactors: Scattering Data

Contains atomic scattering factors from "New Analytical Scattering Factor Functions for Free Atoms and Ions for Free Atoms and Ions", D. Waasmaier & A. Kirfel, *Acta Cryst.* **(1995).** A51, 416-413.

Also, tabulated coefficients for calculation of Compton Cross Section as a function of sin(theta)/lambda from "Analytic Approximations to Incoherently Scattered X-Ray Intensities", H. H. M. Balyuzi, *Acta Cryst.* **(1975).** A31, 600.

3.4 ImageCalibrants: Calibration Standards

GSASII powder calibrants as a dictionary `ImageCalibrants.Calibrants` with substances commonly used for powder calibrations for image data.

Each entry in `ImageCalibrants` consists of:

```
'key':([Bravais num,],[(a,b,c,alpha,beta,gamma),],no. lines skipped,(dmin,pixLimit,cutOff))
```

as an example:

```
'LaB6  SRM660a':([2,],[(4.1569162,4.1569162,4.1569162,90,90,90),],0,(1.0,10,10)),
```

or where "Bravais num" and "(a,b,...)" are repeated in the case of mixtures:

```
'LaB6 & CeO2':([2,0], [(4.1569,4.1569,4.1569,90,90,90),(5.4117,5.4117,5.4117,90,90,90)], 0, (1.0,2,1)
```

To expand this list with locally needed additions, do not modify this file, because you may lose these changes during a software update. Instead duplicate the format of this file in a file named *UserCalibrants.py* and there define the material(s) you want:

```
Calibrants={
  'LaB6 skip 2 lines':([2,],[(4.1569162,4.1569162,4.1569162,90,90,90),],2,(1.0,10,10)),
}
```

New key values will be added to the list of options. If a key is duplicated, the information in *UserCalibrants.py* will override the information in this file.

Note, some useful Bravais numbers are: F-cubic=0, I-cubic=1, P-cubic=2, R3/m (hex)=3, P6=4, P4mmm=6

3.5 *GSASIIpath: locations & updates*

Routines for dealing with file locations, etc.

Determines the location of the compiled (.pyd or .so) libraries.

Interfaces with subversion (svn): Determine the subversion release number by determining the highest version number where `SetVersionNumber()` is called (best done in every GSASII file). Other routines will update GSASII from the subversion server if svn can be found.

`GSASIIpath.`**`GetVersionNumber`**`()`
Return the maximum version number seen in `SetVersionNumber()`

`GSASIIpath.`**`SetVersionNumber`**`(`*RevString*`)`
Set the subversion version number

Parameters **RevString** (*str*) – something like "$Revision: 1047 $" that is set by subversion when the file is retrieved from subversion.

Place `GSASIIpath.SetVersionNumber("$Revision: 1047 $")` in every python file.

`GSASIIpath.`**`svnFindLocalChanges`**`(`*fpath='/Users/toby/software/G2/GSASII'*`)`
Returns a list of files that were changed locally. If no files are changed, the list has length 0

Parameters **fpath** – path to repository dictionary, defaults to directory where the current file is located

Returns None if there is a subversion error (likely because the path is not a repository or svn is not found)

`GSASIIpath.`**`svnGetLog`**`(`*fpath='/Users/toby/software/G2/GSASII'*, *version=None*`)`
Get the revision log information for a specific version of the

Parameters

- **fpath** (*str*) – path to repository dictionary, defaults to directory where the current file is located.
- **version** (*int*) – the version number to be looked up or None (default) for the latest version.

Returns a dictionary with keys (one hopes) 'author', 'date', 'msg', and 'revision'

`GSASIIpath.`**`svnGetRev`**`(`*fpath='/Users/toby/software/G2/GSASII'*, *local=True*`)`
Obtain the version number for the either the last update of the local version or contacts the subversion server to get the latest update version (# of Head).

Parameters

- **fpath** (*str*) – path to repository dictionary, defaults to directory where the current file is located
- **local** (*bool*) – determines the type of version number, where True (default): returns the latest installed update False: returns the version number of Head on the server

Returns the version number as an str or None if there is a subversion error (likely because the path is not a repository or svn is not found)

GSASIIpath.**svnUpdateDir**(*fpath='/Users/toby/software/G2/GSASII'*, *version=None*)
This performs an update of the files in a local directory from a server.

Parameters

- **fpath** (*str*) – path to repository dictionary, defaults to directory where the current file is located
- **version** – the number of the version to be loaded. Used only cast as a string, but should be an integer or something that corresponds to a string representation of an integer value when cast. A value of None (default) causes the latest version on the server to be used.

GSASIIpath.**svnUpdateProcess**(*version=None*, *projectfile=None*)
perform an update of GSAS-II in a separate python process

GSASIIpath.**whichsvn**()
Returns a path to the subversion exe file, if any is found. Searches the current path as well as subdirectory "svn" and "svn/bin" in the location of the GSASII source files.

Returns None if svn is not found or an absolute path to the subversion executable file.

3.6 *GSASIIElem: functions for element types*

GSASIIElem.**CheckElement**(*El*)
Check if element El is in the periodic table

Parameters **El** (*str*) – One or two letter element symbol, capitaliztion ignored

Returns True if the element is found

GSASIIElem.**ComptonFac**(*El*, *SQ*)
compute Compton scattering factor

Parameters

- **El** – element dictionary
- **SQ** – (sin-theta/lambda)**2

Returns compton scattering factor

GSASIIElem.**FPcalc**(*Orbs*, *KEv*)
Compute real & imaginary resonant X-ray scattering factors

Parameters

- **Orbs** – list of orbital dictionaries as defined in GetXsectionCoeff
- **KEv** – x-ray energy in keV

Returns C: (f',f",mu): real, imaginary parts of resonant scattering & atomic absorption coeff.

GSASIIElem.**FixValence**(*El*)
Returns the element symbol, even when a valence is present

GSASIIElem.**GetAtomInfo**(*El*)
reads element information from file atmdata.dat

GSASIIElem.**GetBLtable**(*General*)
returns a dictionary of neutron scattering length data for atom types & isotopes found in General

Parameters **General** (*dict*) – dictionary of phase info.; includes AtomTypes & Isotopes

Returns BLtable, dictionary of scattering length data; key is atom type

GSASIIElem.**GetFFC5**(*ElSym*)
Get 5 term form factor and Compton scattering data

Parameters **ElSym** – str(1-2 character element symbol with proper case);

Return El dictionary with 5 term form factor & compton coefficients

GSASIIElem.**GetFFtable**(*atomTypes*)
returns a dictionary of form factor data for atom types found in atomTypes

Parameters **atomTypes** (*list*) – list of atom types

Returns FFtable, dictionary of form factor data; key is atom type

GSASIIElem.**GetFormFactorCoeff**(*El*)
Read X-ray form factor coefficients from *atomdata.asc* file

Parameters **El** (*str*) – element 1-2 character symbol, case irrevelant

Returns *FormFactors*: list of form factor dictionaries

Each X-ray form factor dictionary is:

- *Symbol*: 4 character element symbol with valence (e.g. 'NI+2')
- *Z*: atomic number
- *fa*: 4 A coefficients
- *fb*: 4 B coefficients
- *fc*: C coefficient

GSASIIElem.**GetMagFormFacCoeff**(*El*)
Read magnetic form factor data from atomdata.asc file

Parameters **El** – 2 character element symbol

Returns MagFormFactors: list of all magnetic form factors dictionaries for element El.

each dictionary contains:

- 'Symbol':Symbol
- 'Z':Z
- 'mfa': 4 MA coefficients
- 'nfa': 4 NA coefficients
- 'mfb': 4 MB coefficients
- 'nfb': 4 NB coefficients
- 'mfc': MC coefficient

- 'nfc': NC coefficient

GSASIIElem.**GetXsectionCoeff**(*El*)
Read atom orbital scattering cross sections for fprime calculations via Cromer-Lieberman algorithm

Parameters **El** – 2 character element symbol

Returns Orbs: list of orbitals each a dictionary with detailed orbital information used by FPcalc

each dictionary is:

- 'OrbName': Orbital name read from file
- 'IfBe' 0/2 depending on orbital
- 'BindEn': binding energy
- 'BB': BindEn/0.02721
- 'XSectIP': 5 cross section inflection points
- 'ElEterm': energy correction term
- 'SEdge': absorption edge for orbital
- 'Nval': 10/11 depending on IfBe
- 'LEner': 10/11 values of log(energy)
- 'LXSect': 10/11 values of log(cross section)

GSASIIElem.**ScatFac**(*El*, *SQ*)
compute value of form factor

Parameters

- **El** – element dictionary defined in GetFormFactorCoeff
- **SQ** – (sin-theta/lambda)**2

Returns real part of form factor

GSASIIElem.**getBLvalues**(*BLtables*)
Needs a doc string

GSASIIElem.**getFFvalues**(*FFtables*, *SQ*, *ifList=False*)
Needs a doc string

3.7 GSASIIlattice: Unit cells

Perform lattice-related computations

Note that g is the reciprocal lattice tensor, and G is its inverse, $G = g^{-1}$, where

$$G = \begin{pmatrix} a^2 & ab\cos\gamma & ac\cos\beta \\ ab\cos\gamma & b^2 & bc\cos\alpha \\ ac\cos\beta & bc\cos\alpha & c^2 \end{pmatrix}$$

The "A tensor" terms are defined as $A = (G_{11} \quad G_{22} \quad G_{33} \quad 2G_{12} \quad 2G_{13} \quad 2G_{23})$ and A can be used in this fashion: $d^* = \sqrt{A_1h^2 + A_2k^2 + A_3l^2 + A_4hk + A_5hl + A_6kl}$, where d is the d-spacing, and d^* is the reciprocal lattice spacing, $Q = 2\pi d^* = 2\pi/d$

GSASIIlattice.**A2Gmat**(*A*, *inverse=True*)
Fill real & reciprocal metric tensor (G) from A.

Parameters

- **A** – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]
- **inverse** (*bool*) – if True return both G and g; else just G

Returns reciprocal (G) & real (g) metric tensors (list of two numpy 3x3 arrays)

GSASIIlattice.**A2cell**(*A*)
Compute unit cell constants from A

Parameters **A** – [G11,G22,G33,2*G12,2*G13,2*G23] G - reciprocal metric tensor

Returns a,b,c,alpha, beta, gamma (degrees) - lattice parameters

GSASIIlattice.**A2invcell**(*A*)
Compute reciprocal unit cell constants from A returns tuple with a*,b*,c*,alpha*, beta*, gamma* (degrees)

GSASIIlattice.**CellAbsorption**(*ElList*, *Volume*)
Compute unit cell absorption

Parameters

- **ElList** (*dict*) – dictionary of element contents including mu and number of atoms be cell
- **Volume** (*float*) – unit cell volume

Returns mu-total/Volume

GSASIIlattice.**CellBlock**(*nCells*)
Generate block of unit cells n*n*n on a side; [0,0,0] centered, n = 2*nCells+1 currently only works for nCells = 0 or 1 (not >1)

GSASIIlattice.**CentCheck**(*Cent*, *H*)
needs doc string

GSASIIlattice.**CosAngle**(*U*, *V*, *G*)
calculate cos of angle between U & V in generalized coordinates defined by metric tensor G

Parameters

- **U** – 3-vectors assume numpy arrays, can be multiple reflections as (N,3) array
- **V** – 3-vectors assume numpy arrays, only as (3) vector
- **G** – metric tensor for U & V defined space assume numpy array

Returns cos(phi)

GSASIIlattice.**CosSinAngle**(*U*, *V*, *G*)
calculate sin & cos of angle between U & V in generalized coordinates defined by metric tensor G

Parameters

- **U** – 3-vectors assume numpy arrays
- **V** – 3-vectors assume numpy arrays
- **G** – metric tensor for U & V defined space assume numpy array

Returns cos(phi) & sin(phi)

GSASIIlattice.**CrsAng**(*H*, *cell*, *SGData*)
needs doc string

GSASIIlattice.**Flnh**(*Start*, *SHCoef*, *phi*, *beta*, *SGData*)
needs doc string

GSASIIlattice.**GenHBravais**(*dmin*, *Bravais*, *A*)
Generate the positionally unique powder diffraction reflections

Parameters

- **dmin** – minimum d-spacing in A
- **Bravais** – lattice type (see GetBraviasNum). Bravais is one of:: 0 F cubic 1 I cubic 2 P cubic 3 R hexagonal (trigonal not rhombohedral) 4 P hexagonal 5 I tetragonal 6 P tetragonal 7 F orthorhombic 8 I orthorhombic 9 C orthorhombic 10 P orthorhombic 11 C monoclinic 12 P monoclinic 13 P triclinic
- **A** – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]

Returns HKL unique d list of [h,k,l,d,-1] sorted with largest d first

GSASIIlattice.**GenHLaue**(*dmin*, *SGData*, *A*)
Generate the crystallographically unique powder diffraction reflections for a lattice and Bravais type

Parameters

- **dmin** – minimum d-spacing
- **SGData** – space group dictionary with at least
  - 'SGLaue': Laue group symbol: one of '-1','2/m','mmm','4/m','6/m','4/mmm','6/mmm', '3m1', '31m', '3', '3R', '3mR', 'm3', 'm3m'
  - 'SGLatt': lattice centering: one of 'P','A','B','C','I','F'
  - 'SGUniq': code for unique monoclinic axis one of 'a','b','c' (only if 'SGLaue' is '2/m') otherwise an empty string
- **A** – reciprocal metric tensor elements as [G11,G22,G33,2*G12,2*G13,2*G23]

Returns HKL = list of [h,k,l,d] sorted with largest d first and is unique part of reciprocal space ignoring anomalous dispersion

GSASIIlattice.**GenSHCoeff**(*SGLaue*, *SamSym*, *L*, *IfLMN=True*)
needs doc string

GSASIIlattice.**GetBraviasNum**(*center*, *system*)
Determine the Bravais lattice number, as used in GenHBravais

Parameters

- **center** – one of: 'P', 'C', 'I', 'F', 'R' (see SGLatt from GSASIIspc.SpcGroup)
- **system** – one of 'cubic', 'hexagonal', 'tetragonal', 'orthorhombic', 'trigonal' (for R) 'monoclinic', 'triclinic' (see SGSys from GSASIIspc.SpcGroup)

Returns a number between 0 and 13 or throws a ValueError exception if the combination of center, system is not found (i.e. non-standard)

GSASIIlattice.**GetKcl**(*L*, *N*, *SGLaue*, *phi*, *beta*)
needs doc string

GSASIIlattice.**GetKclKsl**(*L*, *N*, *SGLaue*, *psi*, *phi*, *beta*)
This is used for spherical harmonics description of preferred orientation; cylindrical symmetry only (M=0) and no sample angle derivatives returned

GSASIIlattice.**GetKsl**(*L*, *M*, *SamSym*, *psi*, *gam*)
needs doc string

GSASIIlattice.**Glnh**(*Start*, *SHCoef*, *psi*, *gam*, *SamSym*)
needs doc string

GSASIIlattice.**Gmat2A**(*G*)
Extract A from reciprocal metric tensor (G)

Parameters **G** – reciprocal maetric tensor (3x3 numpy array

Returns A = [G11,G22,G33,2*G12,2*G13,2*G23]

GSASIIlattice.**Gmat2AB**(*G*)
Computes orthogonalization matrix from reciprocal metric tensor G

Returns

tuple of two 3x3 numpy arrays (A,B)

- A for crystal to Cartesian transformations A*x = np.inner(A,x) = X
- B (= inverse of A) for Cartesian to crystal transformation B*X = np.inner(B,X) = x

GSASIIlattice.**Gmat2cell**(*g*)
Compute real/reciprocal lattice parameters from real/reciprocal metric tensor (g/G) The math works the same either way.

Parameters **(or G)** (*g*) – real (or reciprocal) metric tensor 3x3 array

Returns a,b,c,alpha, beta, gamma (degrees) (or a*,b*,c*,alpha*,beta*,gamma* degrees)

GSASIIlattice.**Hx2Rh**(*Hx*)
needs doc string

GSASIIlattice.**MaxIndex**(*dmin*, *A*)
needs doc string

GSASIIlattice.**OdfChk**(*SGLaue*, *L*, *M*)
needs doc string

GSASIIlattice.**Rh2Hx**(*Rh*)
needs doc string

GSASIIlattice.**SamAng**(*Tth*, *Gangls*, *Sangl*, *IFCoup*)
Compute sample orientation angles vs laboratory coord. system

Parameters

- **Tth** – Signed theta
- **Gangls** – Sample goniometer angles phi,chi,omega,azmuth
- **Sangl** – Sample angle zeros om-0, chi-0, phi-0
- **IFCoup** – True if omega & 2-theta coupled in CW scan

Returns psi,gam: Sample odf angles dPSdA,dGMdA: Angle zero derivatives

GSASIIlattice.**SwapIndx**(*Axis*, *H*)
needs doc string

GSASIIlattice.**U6toUij**(*U6*)
Fill matrix (Uij) from U6 = [U11,U22,U33,U12,U13,U23] NB: there is a non numpy version in GSASIIspc: U2Uij

Parameters **U6** (*list*) – 6 terms of u11,u22,...

Returns Uij - numpy [3][3] array of uij

GSASIIlattice.**Uij2Ueqv**(*Uij*, *GS*, *Amat*)
returns 1/3 trace of diagonalized U matrix

GSASIIlattice.**Uij2betaij**(*Uij*, *G*)
Convert Uij to beta-ij tensors – stub for eventual completion

Parameters

- **Uij** – numpy array [Uij]
- **G** – reciprocal metric tensor

Returns beta-ij - numpy array [beta-ij]

GSASIIlattice.**UijtoU6**(*U*)
Fill vector [U11,U22,U33,U12,U13,U23] from Uij NB: there is a non numpy version in GSASIIspc: Uij2U

GSASIIlattice.**calc_V**(*A*)
Compute the real lattice volume (V) from A

GSASIIlattice.**calc_rDsq**(*H*, *A*)
needs doc string

GSASIIlattice.**calc_rDsq2**(*H*, *G*)
needs doc string

GSASIIlattice.**calc_rDsqZ**(*H*, *A*, *Z*, *tth*, *lam*)
needs doc string

GSASIIlattice.**calc_rV**(*A*)
Compute the reciprocal lattice volume (V*) from A

GSASIIlattice.**calc_rVsq**(*A*)
Compute the square of the reciprocal lattice volume (1/V**2) from A'

GSASIIlattice.**cell2A**(*cell*)
Obtain A = [G11,G22,G33,2*G12,2*G13,2*G23] from lattice parameters

Parameters **cell** – [a,b,c,alpha,beta,gamma] (degrees)

Returns G reciprocal metric tensor as 3x3 numpy array

GSASIIlattice.**cell2AB**(*cell*)
Computes orthogonalization matrix from unit cell constants

Parameters **cell** (*tuple*) – a,b,c, alpha, beta, gamma (degrees)

Returns tuple of two 3x3 numpy arrays (A,B) A for crystal to Cartesian transformations A*x = np.inner(A,x) = X B (= inverse of A) for Cartesian to crystal transformation B*X = np.inner(B,X) = x

GSASIIlattice.**cell2GS**(*cell*)
returns Uij to betaij conversion matrix

GSASIIlattice.**cell2Gmat**(*cell*)
Compute real and reciprocal lattice metric tensor from unit cell constants

Parameters **cell** – tuple with a,b,c,alpha, beta, gamma (degrees)

Returns reciprocal (G) & real (g) metric tensors (list of two numpy 3x3 arrays)

GSASIIlattice.**combinations**(*items*, *n*)
take n distinct items, order matters

GSASIIlattice.**criticalEllipse**(*prob*)
Calculate critical values for probability ellipsoids from probability

GSASIIlattice.**fillgmat**(*cell*)
Compute lattice metric tensor from unit cell constants

Parameters **cell** – tuple with a,b,c,alpha, beta, gamma (degrees)

Returns 3x3 numpy array

GSASIIlattice.**getHKLmax**(*dmin*, *SGData*, *A*)
finds maximum allowed hkl for given A within dmin

GSASIIlattice.**invcell2Gmat**(*invcell*)
Compute real and reciprocal lattice metric tensor from reciprocal unit cell constants

Parameters **invcell** – [a*,b*,c*,alpha*, beta*, gamma*] (degrees)

Returns reciprocal (G) & real (g) metric tensors (list of two 3x3 arrays)

GSASIIlattice.**invpolfcal**(*ODFln*, *SGData*, *phi*, *beta*)
needs doc string

GSASIIlattice.**permutations**(*items*)
take all items, order matters

GSASIIlattice.**polfcal**(*ODFln*, *SamSym*, *psi*, *gam*)
needs doc string

GSASIIlattice.**rotdMat**(*angle*, *axis=0*)
Prepare rotation matrix for angle in degrees about axis(=0,1,2)

Parameters

- **angle** – angle in degrees
- **axis** – axis (0,1,2 = x,y,z) about which for the rotation

Returns rotation matrix - 3x3 numpy array

GSASIIlattice.**rotdMat4**(*angle*, *axis=0*)
Prepare rotation matrix for angle in degrees about axis(=0,1,2) with scaling for OpenGL

Parameters

- **angle** – angle in degrees
- **axis** – axis (0,1,2 = x,y,z) about which for the rotation

Returns rotation matrix - 4x4 numpy array (last row/column for openGL scaling)

GSASIIlattice.**sec2HMS**(*sec*)
Convert time in sec to H:M:S string

Parameters **sec** – time in seconds

Returns H:M:S string (to nearest 100th second)

GSASIIlattice.**selections**(*items*, *n*)
take n (not necessarily distinct) items, order matters

GSASIIlattice.**selftestlist = [<function test0 at 0x209e930>, <function test1 at 0x209e970>, <function test2 at 0x209e**
Defines a list of self-tests

GSASIIlattice.**sortHKLd**(*HKLd*, *ifreverse*, *ifdup*)
needs doc string

Parameters

- **HKLd** – a list of [h,k,l,d,...];

- **ifreverse** – True for largest d first
- **ifdup** – True if duplicate d-spacings allowed

GSASIIlattice.**test1**()
test cell2A and A2Gmat

GSASIIlattice.**test2**()
test Gmat2A, A2cell, A2Gmat, Gmat2cell

GSASIIlattice.**test3**()
test invcell2Gmat

GSASIIlattice.**test4**()
test calc_rVsq, calc_rV, calc_V

GSASIIlattice.**test5**()
test A2invcell

GSASIIlattice.**test6**()
test cell2AB

GSASIIlattice.**test7**()
test GetBraviasNum(...) and GenHBravais(...)

GSASIIlattice.**test8**()
test GenHLaue

GSASIIlattice.**test9**()
test GenHLaue

GSASIIlattice.**textureIndex**(*SHCoef*)
needs doc string

GSASIIlattice.**uniqueCombinations**(*items*, *n*)
take n distinct items, order is irrelevant

3.8 *GSASIIspc: Space group module*

Space group interpretation routines. Note that space group information is stored in a *Space Group (SGData)* object.

GSASIIspc.**AllOps**(*SGData*)
Returns a list of all operators for a space group, including those for centering and a center of symmetry

Parameters **SGData** – from `SpcGroup()`

Returns

(SGTextList,offsetList,symOpList,G2oprList) where

- SGTextList: a list of strings with formatted and normalized symmetry operators.
- offsetList: a tuple of (dx,dy,dz) offsets that relate the GSAS-II symmetry operation to the operator in SGTextList and symOpList. these dx (etc.) values are added to the GSAS-II generated positions to provide the positions that are generated by the normalized symmetry operators.
- symOpList: a list of tuples with the normalized symmetry operations as (M,T) values (see `SGOps` in the *Space Group object*)
- G2oprList: The GSAS-II operations for each symmetry operation as a tuple with (center,mult,opnum), where center is (0,0,0), (0.5,0,0), (0.5,0.5,0.5),...; where mult is 1 or -1 for

the center of symmetry and opnum is the number for the symmetry operation, in SGOps (starting with 0).

GSASIIspc.**ApplyStringOps**(*A*, *SGData*, *X*, *Uij*=[ ])
Needs a doc string

GSASIIspc.**ElemPosition**(*SGData*)
Under development. Object here is to return a list of symmetry element types and locations suitable for say drawing them. So far I have the element type... getting all possible locations without lookup may be impossible!

GSASIIspc.**GenAtom**(*XYZ*, *SGData*, *All=False*, *Uij*=[ ], *Move=True*)
Generates the equivalent positions for a specified coordinate and space group

Parameters

- **XYZ** – an array, tuple or list containing 3 elements: x, y & z
- **SGData** – from SpcGroup()
- **All** – True return all equivalent positions including duplicates; False return only unique positions
- **Uij** – [U11,U22,U33,U12,U13,U23] or [] if no Uij
- **Move** – True move generated atom positions to be inside cell False do not move atoms

Returns

[[XYZEquiv],Idup,[UijEquiv]]

- [XYZEquiv] is list of equivalent positions (XYZ is first entry)
- Idup = [-][C]SS where SS is the symmetry operator number (1-24), C (if not 0,0,0)
- is centering operator number (1-4) and - is for inversion Cell = unit cell translations needed to put new positions inside cell [UijEquiv] - equivalent Uij; absent if no Uij given

GSASIIspc.**GenHKLf**(*HKL*, *SGData*)
Uses old GSAS Fortran routine genhkl.for

Parameters

- **HKL** – [h,k,l]
- **SGData** – space group data obtained from SpcGroup

Returns

iabsnt,mulp,Uniq,phi

- iabsnt = True if reflection is forbidden by symmetry
- mulp = reflection multiplicity including Friedel pairs
- Uniq = numpy array of equivalent hkl in descending order of h,k,l

GSASIIspc.**GetCSuinel**(*siteSym*)
returns Uij terms, multipliers, GUI flags & Uiso2Uij multipliers

GSASIIspc.**GetCSxinel**(*siteSym*)
Needs a doc string

GSASIIspc.**GetKNsym**(*key*)
Needs a doc string

GSASIIspc.**GetNXUPQsym**(*siteSym*)
Needs a doc string

GSASIIspc.**GetOprPtrName**(*key*)
Needs a doc string

GSASIIspc.**HStrainNames**(*SGData*)
Needs a doc string

GSASIIspc.**Latt2text**(*Latt*)
From lattice type ('P',A', etc.) returns ';' delimited cell centering vectors

GSASIIspc.**MT2text**(*M*, *T*)
From space group matrix/translation operator returns text version

GSASIIspc.**MoveToUnitCell**(*xyz*)
Translates a set of coordinates so that all values are >=0 and < 1

Parameters **xyz** – a list or numpy array of fractional coordinates

Returns XYZ - numpy array of new coordinates now 0 or greater and less than 1

GSASIIspc.**Muiso2Shkl**(*muiso*, *SGData*, *cell*)
this is to convert isotropic mustrain to generalized Shkls - doesn't work just now

GSASIIspc.**MustrainCoeff**(*HKL*, *SGData*)
Needs a doc string

GSASIIspc.**MustrainNames**(*SGData*)
Needs a doc string

GSASIIspc.**Opposite**(*XYZ*, *toler=0.0002*)

Gives opposite corner, edge or face of unit cell for position within tolerance. Result may be just outside the cell within tolerance

Parameters

- **XYZ** – 0 >= np.array[x,y,z] > 1 as by MoveToUnitCell
- **toler** – unit cell fraction tolerance making opposite

Returns XYZ: array of opposite positions; always contains XYZ

GSASIIspc.**SGErrors**(*IErr*)
Interprets the error message code from SpcGroup. Used in SpaceGroup.

Parameters **IErr** – see SGError in `SpcGroup()`

Returns ErrString - a string with the error message or "Unknown error"

GSASIIspc.**SGPrint**(*SGData*)
Print the output of SpcGroup in a nicely formatted way. Used in SpaceGroup

Parameters **SGData** – from `SpcGroup()`

Returns SGText - list of strings with the space group details

GSASIIspc.**SGpolar**(*SGData*)
Determine identity of polar axes if any

GSASIIspc.**SpaceGroup**(*SGSymbol*)
Print the output of SpcGroup in a nicely formatted way.

Parameters **SGSymbol** – space group symbol (string) with spaces between axial fields

Returns nothing

GSASIIspc.**SpcGroup**(*SGSymbol*)
Determines cell and symmetry information from a short H-M space group name

Parameters **SGSymbol** – space group symbol (string) with spaces between axial fields

Returns

(SGError,SGData) * SGError = 0 for no errors; >0 for errors (see SGErrors below for details) * SGData - is a dict (see *Space Group object*) with entries:

- 'SpGrp': space group symbol, slightly cleaned up
- 'Laue': one of '-1', '2/m', 'mmm', '4/m', '4/mmm', '3R', '3mR', '3', '3m1', '31m', '6/m', '6/mmm', 'm3', 'm3m'
- 'SGInv': boolean; True if centrosymmetric, False if not
- 'SGLatt': one of 'P', 'A', 'B', 'C', 'I', 'F', 'R'
- 'SGUniq': one of 'a', 'b', 'c' if monoclinic, '' otherwise
- 'SGCen': cell centering vectors [0,0,0] at least
- 'SGOps': symmetry operations as [M,T] so that M*x+T = x'
- 'SGSys': one of 'triclinic', 'monoclinic', 'orthorhombic', 'tetragonal', 'rhombohedral', 'trigonal', 'hexagonal', 'cubic'
- 'SGPolax': one of '', 'x', 'y', 'x y', 'z', 'x z', 'y z', 'xyz', '111' for arbitrary axes

GSASIIspc.**StandardizeSpcName**(*spcgroup*)
Accept a spacegroup name where spaces may have not been used in the names according to the GSAS convention (spaces between symmetry for each axis) and return the space group name as used in GSAS

GSASIIspc.**StringOpsProd**(*A*, *B*, *SGData*)
Find A*B where A & B are in strings '-' + '100*c+n' + '+ijk' where '-' indicates inversion, c(>0) is the cell centering operator, n is operator number from SgOps and ijk are unit cell translations (each may be <0). Should return resultant string - C. SGData - dictionary using entries:

- 'SGCen': cell centering vectors [0,0,0] at least
- 'SGOps': symmetry operations as [M,T] so that M*x+T = x'

GSASIIspc.**SytSym**(*XYZ*, *SGData*)
Generates the number of equivalent positions and a site symmetry code for a specified coordinate and space group

Parameters

- **XYZ** – an array, tuple or list containing 3 elements: x, y & z
- **SGData** – from SpcGroup

Returns a two element tuple:

- The 1st element is a code for the site symmetry (see GetKNsym)
- The 2nd element is the site multiplicity

GSASIIspc.**selftestlist = [<function test0 at 0x5bda4f0>, <function test1 at 0x5bda530>, <function test2 at 0x5bda570>,**
Defines a list of self-tests

GSASIIspc.**test0**()
self-test #0: exercise MoveToUnitCell

GSASIIspc.**test1**()
self-test #1: SpcGroup and SGPrint against previous results

GSASIIspc.**test2**()
self-test #2: SpcGroup against cctbx (sgtbx) computations

GSASIIspc.**test3**()
self-test #3: exercise SytSym (includes GetOprPtrName, GenAtom, GetKNsym) for selected space groups against info in IT Volume A

3.9 *gltext: draw OpenGL text*

Routines that render text on OpenGL without use of GLUT.

Code written by Christian Brugger & Stefan Hacker and distributed under GNU General Public License.

class gltext.**Text**(*text='Text'*, *font=None*, *font_size=8*, *foreground=wx.Colour()*, *centered=False*)
A simple class for using System Fonts to display text in an OpenGL scene. The Text adds a global Cache of already created text elements to TextElement's base functionality so you can save some memory and increase speed

centered
Display the text centered

draw_text(*position=wx.Point(0, 0)*, *scale=1.0*, *rotation=0*)
position (wx.Point) - x/y Position to draw in scene scale (float) - Scale rotation (int) - Rotation in degree

Draws the text to the scene

font
Font of the object

font_size
Font size

foreground
Color/Overlay bitmap of the text

getTextElement()
Returns the text element bound to the Text class

getTexture()
Returns the texture of the bound TextElement

getTexture_size()
Returns a texture size tuple

setCentered(*value*, *reinit=True*)
value (bool) - New centered value reinit (bool) - Create a new texture

Sets a new value for 'centered'

setFont(*value*, *reinit=True*)
value (bool) - New Font reinit (bool) - Create a new texture

Sets a new font

setFont_size(*value*, *reinit=True*)
value (bool) - New font size reinit (bool) - Create a new texture

Sets a new font size

setForeground(*value*, *reinit=True*)
value (bool) - New centered value reinit (bool) - Create a new texture

Sets a new value for 'centered'

setText(*value*, *reinit=True*)
value (bool) - New Text reinit (bool) - Create a new texture

Sets a new text

text
Text of the object

text_element
TextElement bound to this class

texture
Texture of bound TextElement

texture_size
Size of the used texture

class gltext.**TextElement**(*text=''*, *font=None*, *foreground=wx.Colour()*, *centered=False*)
A simple class for using system Fonts to display text in an OpenGL scene

bind()
Increase refcount

centered
Is text centered

createTexture()
Creates a texture from the settings saved in TextElement, to be able to use normal system fonts conviently a wx.MemoryDC is used to draw on a wx.Bitmap. As wxwidgets device contexts don't support alpha at all it is necessary to apply a little hack to preserve antialiasing without sticking to a fixed background color:

We draw the bmp in b/w mode so we can use its data as a alpha channel for a solid color bitmap which after GL_ALPHA_TEST and GL_BLEND will show a nicely antialiased text on any surface.

To access the raw pixel data the bmp gets converted to a wx.Image. Now we just have to merge our foreground color with the alpha data we just created and push it all into a OpenGL texture and we are DONE *inhalesdelpy*

DRAWBACK of the whole conversion thing is a really long time for creating the texture. If you see any optimizations that could save time PLEASE CREATE A PATCH!!!

deleteTexture()
Deletes the OpenGL texture object

draw_text(*position=wx.Point(0, 0)*, *scale=1.0*, *rotation=0*)
position (wx.Point) - x/y Position to draw in scene scale (float) - Scale rotation (int) - Rotation in degree

Draws the text to the scene

font
Font of the object

foreground
Color of the text

isBound()
Return refcount

owner_cnt
Owner count

release()
Decrease refcount

text
Text of the object

texture
Used texture

texture_size
Size of the used texture

CHAPTER

FOUR

GSAS-II GUI ROUTINES

4.1 GSASIIgrid: Basic GUI routines

class GSASIIgrid.**ASCIIValidator**(*result=None*, *key=None*)
A validator to be used with a TextCtrl to prevent entering characters other than ASCII characters.

The value is checked for validity after every keystroke If an invalid number is entered, the box is highlighted. If the number is valid, it is saved in result[key]

Parameters

- **result** (*dict/list*) – List or dict where value should be placed when valid
- **key** (*any*) – key to use for result (int for list)

Clone()
Create a copy of the validator, a strange, but required component

OnChar(*event*)
Called each type a key is pressed ignores keys that are not allowed for int and float types

TestValid(*tc*)
Check if the value is valid by casting the input string into ASCII.

Save it in the dict/list where the initial value was stored

Parameters **tc** (*wx.TextCtrl*) – A reference to the TextCtrl that the validator is associated with.

TransferFromWindow()
Needed by validator, strange, but required component

TransferToWindow()
Needed by validator, strange, but required component

class GSASIIgrid.**AddHelp**(*frame*, *helpType*, *helpLbl=None*, *title=''*)
For the Mac: creates an entry to the help menu of type 'Help on <helpType>': where helpType is a reference to an HTML page to be opened.

NOTE: when appending this menu (menu.Append) be sure to set the title to '&Help' so that wx handles it correctly.

OnHelpById(*event*)
Called when Help on... is pressed in a menu. Brings up a web page for documentation.

GSASIIgrid.**CallScrolledMultiEditor**(*parent*, *dictlst*, *elemlst*, *prelbl=[]*, *postlbl=[]*, *title='Edit items'*, *header=''*, *size=(300, 250)*, *CopyButton=False*)
Shell routine to call a ScrolledMultiEditor dialog. See ScrolledMultiEditor for parameter definitions.

Returns True if the OK button is pressed; False if the window is closed with the system menu or the Cancel button.

class GSASIIgrid.**DataFrame**(*parent*, *frame*, *data=None*, *name=None*, *size=None*, *pos=None*)
Create the data item window and all the entries in menus used in that window. For Linux and windows, the menu entries are created for the current data item window, but in the Mac the menu is accessed from all windows. This means that a different menu is posted depending on which data item is posted. On the Mac, all the menus contain the data tree menu items, but additional menus are added specific to the data item.

Note that while the menus are created here, the binding for the menus is done later in various GSASII*GUI modules, where the functions to be called are defined.

Bind(**args*, ***kwargs*)
Override the Bind() function: on the Mac the binding is to the main window, so that menus operate with any window on top. For other platforms, call the default wx.Frame Bind()

PostfillDataMenu(*empty=False*)
Create the "standard" part of data frame menus. Note that on Linux and Windows, this is the standard help Menu. On Mac, this menu duplicates the tree menu, but adds an extra help command for the data item and a separator.

PrefillDataMenu(*menu*, *helpType*, *helpLbl=None*, *empty=False*)
Create the "standard" part of data frame menus. Note that on Linux and Windows nothing happens here. On Mac, this menu duplicates the tree menu, but adds an extra help command for the data item and a separator.

class GSASIIgrid.**DisAglDialog**(*parent*, *data*, *default*)
Distance Angle Controls dialog

class GSASIIgrid.**EnumSelector**(*parent*, *dct*, *item*, *choices*, *values=None*, ***kw*)
A customized `wxpython.ComboBox` that selects items from a list of choices, but sets a dict (list) entry to the corresponding entry from the input list of values.

Parameters

- **parent** (*wx.Panel*) – the parent to the `ComboBox` (usually a frame or panel)
- **dct** (*dict*) – a dict (or list) to contain the value set for the `ComboBox`.
- **item** – the dict key (or list index) where `dct[item]` will be set to the value selected in the `ComboBox`. Also, dct[item] contains the starting value shown in the widget. If the value does not match an entry in `values`, the first value in `choices` is used as the default, but `dct[item]` is not changed.
- **choices** (*list*) – a list of choices to be displayed to the user such as

```
["default","option 1","option 2",]
```

  Note that these options will correspond to the entries in `values` (if specified) item by item.
- **values** (*list*) – a list of values that correspond to the options in `choices`, such as

```
[0,1,2]
```

  The default for `values` is to use the same list as specified for `choices`.
- **(other)** – additional keyword arguments accepted by `ComboBox` can be specified.

class GSASIIgrid.**G2HtmlWindow**(*parent*, **args*, ***kwargs*)
Displays help information in a primitive HTML browser type window

class GSASIIgrid.**GSGrid**(*parent*, *name=''*)
Basic wx.Grid implementation

class GSASIIgrid.**GSNoteBook**(*parent*, *name=''*, *size=None*)
Notebook used in various locations; implemented with wx.aui extension

GSASIIgrid.**GetPatternTreeDataNames**(*G2frame*, *dataTypes*)
Needs a doc string

GSASIIgrid.**GetPatternTreeItemId**(*G2frame*, *parentId*, *itemText*)
Needs a doc string

class GSASIIgrid.**GridFractionEditor**(*grid*)
A grid cell editor class that allows entry of values as fractions as well as sine and cosine values [as s() and c()]

GSASIIgrid.**HorizontalLine**(*sizer*, *parent*)
Draws a horizontal line as wide as the window. This shows up on the Mac as a very thin line, no matter what I do

GSASIIgrid.**ItemSelector**(*ChoiceList*, *ParentFrame=None*, *title='Select an item'*, *size=None*, *header='Item Selector'*, *useCancel=True*, *multiple=False*)
Provide a wx dialog to select a single item from list of choices

Parameters

- **ChoiceList** (*list*) – a list of choices where one will be selected
- **ParentFrame** (*wx.Frame*) – Name of parent frame (default None)
- **title** (*str*) – heading above list of choices (default 'Select an item')
- **size** (*wx.Size*) – Size for dialog to be created (default None – size as needed)
- **header** (*str*) – Title to place on window frame (default 'Item Selector')
- **useCancel** (*bool*) – If True (default) both the OK and Cancel buttons are offered
- **multiple** (*bool*) – If True then multiple items can be selected (default False)

Returns the selection index or None or a selection list if multiple is true

GSASIIgrid.**MovePatternTreeToGrid**(*G2frame*, *item*)
Needs a doc string

class GSASIIgrid.**MyHelp**(*frame*, *helpType=None*, *helpLbl=None*, *morehelpitems=[]*, *title=''*)
A class that creates the contents of a help menu. The menu will start with two entries:

•'Help on <helpType>': where helpType is a reference to an HTML page to be opened

•About: opens an About dialog using OnHelpAbout. N.B. on the Mac this gets moved to the App menu to be consistent with Apple style.

NOTE: for this to work properly with respect to system menus, the title for the menu must be &Help, or it will not be processed properly:

```
menu.Append(menu=MyHelp(self,...),title="&Help")
```

OnCheckUpdates(*event*)
Check if the GSAS-II repository has an update for the current source files and perform that update if requested.

OnHelpAbout(*event*)
Display an 'About GSAS-II' box

OnHelpById(*event*)
Called when Help on... is pressed in a menu. Brings up a web page for documentation.

OnSelectVersion(*event*)
Allow the user to select a specific version of GSAS-II

class GSASIIgrid.**MyHtmlPanel**(*frame*, *id*)
Defines a panel to display HTML help information, as an alternative to displaying help information in a web browser.

class GSASIIgrid.**NumberValidator**(*typ*, *positiveonly=False*, *min=None*, *max=None*, *result=None*, *key=None*, *OKcontrol=None*, *CIFinput=False*)
A validator to be used with a TextCtrl to prevent entering characters other than digits, signs, and for float input, a period and exponents.

The value is checked for validity after every keystroke If an invalid number is entered, the box is highlighted. If the number is valid, it is saved in result[key]

Parameters

- **typ** (*type*) – the base data type. Must be int or float.
- **positiveonly** (*bool*) – If True, negative integers are not allowed (default False). This prevents the + or - keys from being pressed. Used with typ=int; ignored for typ=float.
- **min** (*number*) – Minimum allowed value. If None (default) the lower limit is unbounded
- **max** (*number*) – Maximum allowed value. If None (default) the upper limit is unbounded
- **result** (*dict/list*) – List or dict where value should be placed when valid
- **key** (*any*) – key to use for result (int for list)
- **OKcontrol** (*function*) – function or class method to control an OK button for a window. Ignored if None (default)
- **CIFinput** (*bool*) – allows use of a single '?' or '.' character as valid input.

CheckInput(*previousInvalid*)
called to test every change to the TextCtrl for validity and to change the appearance of the TextCtrl

Anytime the input is invalid, call self.OKcontrol (if defined) because it is fast. If valid, check for any other invalid entries only when changing from invalid to valid, since that is slower.

Parameters **previousInvalid** (*bool*) – True if the TextCtrl contents were invalid prior to the current change.

Clone()
Create a copy of the validator, a strange, but required component

OnChar(*event*)
Called each type a key is pressed ignores keys that are not allowed for int and float types

ShowValidity(*tc*)
Set the control colors to show invalid input

Parameters **tc** (*wx.TextCtrl*) – A reference to the TextCtrl that the validator is associated with.

TestValid(*tc*)
Check if the value is valid by casting the input string into the current type.

Set the invalid variable in the TextCtrl object accordingly.

If the value is valid, save it in the dict/list where the initial value was stored, if appropriate.

Parameters **tc** (*wx.TextCtrl*) – A reference to the TextCtrl that the validator is associated with.

TransferFromWindow()
Needed by validator, strange, but required component

TransferToWindow()
Needed by validator, strange, but required component

class GSASIIgrid.**PickTwoDialog**(*parent*, *title*, *prompt*, *names*, *choices*)
This does not seem to be in use

class GSASIIgrid.**ScrolledMultiEditor**(*parent*, *dictlst*, *elemlst*, *prelbl=[]*, *postlbl=[]*, *title='Edit items'*, *header=''*, *size=(300, 250)*, *CopyButton=False*, *minvals=[]*, *maxvals=[]*, *sizevals=[]*)
Define a window for editing a potentially large number of dict- or list-contained values with validation for each item. Edited values are automatically placed in their source location. If invalid entries are provided, the TextCtrl is turned yellow and the OK button is disabled.

The type for each TextCtrl validation is determined by the initial value of the entry (int, float or string). Float values can be entered in the TextCtrl as numbers or also as algebraic expressions using operators + - / * () and **, in addition pi, sind(), cosd(), tand(), and sqrt() can be used, as well as appreviations s(), sin(), c(), cos(), t(), tan() and sq().

Parameters

- **parent** (*wx.Frame*) – name of parent window, or may be None
- **dictlst** (*tuple*) – a list of dicts or lists containing values to edit
- **elemlst** (*tuple*) – a list of keys for each item in a dictlst. Must have the same length as dictlst.
- **parent** – name of parent window, or may be None
- **prelbl** (*tuple*) – a list of labels placed before the TextCtrl for each item (optional)
- **postlbl** (*tuple*) – a list of labels placed after the TextCtrl for each item (optional)
- **title** (*str*) – a title to place in the frame of the dialog
- **header** (*str*) – text to place at the top of the window. May contain new line characters.
- **size** (*wx.Size*) – a size parameter that dictates the size for the scrolled region of the dialog. The default is (300,250).
- **CopyButton** (*bool*) – if True adds a small button that copies the value for the current row to all fields below (default is False)
- **minvals** (*list*) – optional list of minimum values for validation of float or int values. Ignored if value is None.
- **maxvals** (*list*) – optional list of maximum values for validation of float or int values. Ignored if value is None.
- **sizevals** (*list*) – optional list of wx.Size values for each input widget. Ignored if value is None.

Returns the wx.Dialog created here. Use method .ShowModal() to display it.

Example for use of ScrolledMultiEditor:

```
dlg = <pkg>.ScrolledMultiEditor(frame,dictlst,elemlst,prelbl,postlbl,
                                header=header)
if dlg.ShowModal() == wx.ID_OK:
     for d,k in zip(dictlst,elemlst):
         print d[k]
```

Example definitions for dictlst and elemlst:

```
        dictlst = (dict1,list1,dict1,list1)
        elemlst = ('a', 1, 2, 3)

    This causes items dict1['a'], list1[1], dict1[2] and list1[3] to be edited.
```

Note that these items must have int, float or str values assigned to them. The dialog will force these types to be retained. String values that are blank are marked as invalid.

ControlOKButton(*setvalue*)
Enable or Disable the OK button for the dialog. Note that this is passed into the ValidatedTxtCtrl for use by validators.

Parameters **setvalue** (*bool*) – if True, all entries in the dialog are checked for validity. if False then the OK button is disabled.

GSASIIgrid.**SetDataMenuBar**(*G2frame*, *menu=None*)
Set the menu for the data frame. On the Mac put this menu for the data tree window instead.

Note that data frame items do not have menus, for these (menu=None) display a blank menu or on the Mac display the standard menu for the data tree window.

GSASIIgrid.**ShowHelp**(*helpType*, *frame*)
Called to bring up a web page for documentation.

class GSASIIgrid.**SingleFloatDialog**(*parent, title, prompt, value, limits=[0.0, 1.0], format='%.5g'*)
Dialog to obtain a single float value from user

class GSASIIgrid.**SingleStringDialog**(*parent*, *title*, *prompt*, *value=''*, *size=(200*, *-1)*)
Dialog to obtain a single string value from user

Parameters

- **parent** (*wx.Frame*) – name of parent frame
- **title** (*str*) – title string for dialog
- **prompt** (*str*) – string to tell use what they are inputting
- **value** (*str*) – default input value, if any

GetValue()
Use this method to get the value entered by the user :returns: string entered by user

Show()
Use this method after creating the dialog to post it :returns: True if the user pressed OK; False if the User pressed Cancel

class GSASIIgrid.**SymOpDialog**(*parent*, *SGData*, *New=True*, *ForceUnit=False*)
Class to select a symmetry operator

class GSASIIgrid.**Table**(*data=[]*, *rowLabels=None*, *colLabels=None*, *types=None*)
Basic data table for use with GSgrid

GSASIIgrid.**UpdateControls**(*G2frame*, *data*)
Edit overall GSAS-II controls in main Controls data tree entry

GSASIIgrid.**UpdateHKLControls**(*G2frame*, *data*)
Needs a doc string

GSASIIgrid.**UpdateNotebook**(*G2frame*, *data*)
Called when the data tree notebook entry is selected. Allows for editing of the text in that tree entry

GSASIIgrid.**UpdatePWHKPlot**(*G2frame*, *kind*, *item*)
Called when the histogram main tree entry is called. Displays the histogram weight factor, refinement statistics for the histogram and the range of data for a simulation.

Also invokes a plot of the histogram.

GSASIIgrid.**UpdateSeqResults**(*G2frame*, *data*)
Called when the Sequential Results data tree entry is selected to show results from a sequential refinement.

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II data tree windows
- **data** (*dict*) – a dictionary containing the following items:
  - 'histNames' - list of histogram names in order as processed by Sequential Refinement
  - 'varyList' - list of variables - identical over all refinements in sequence
  - 'histName' - dictionaries for all data sets processed, which contains:
    - 'variables'- result[0] from leastsq call
    - 'varyList' - list of variables; same as above
    - 'sig' - esds for variables
    - 'covMatrix' - covariance matrix from individual refinement
    - 'title' - histogram name; same as dict item name
    - 'newAtomDict' - new atom parameters after shifts applied
    - 'newCellDict' - new cell parameters after shifts to A0-A5 applied'

class GSASIIgrid.**ValidatedTxtCtrl**(*parent*, *loc*, *key*, *notBlank=True*, *min=None*, *max=None*, *OKcontrol=None*, *OnLeave=None*, *typeHint=None*, *CIFinput=False*, ***kw*)
Create a TextCtrl widget that uses a validator to prevent the entry of inappropriate characters and changes color to highlight when invalid input is supplied. As valid values are typed, they are placed into the dict or list where the initial value came from. The type of the initial value must be int, float or str or None (see `key` and `typeHint`); this type (or the one in `typeHint`) is preserved.

Float values can be entered in the TextCtrl as numbers or also as algebraic expressions using operators + - / * () and **, in addition pi, sind(), cosd(), tand(), and sqrt() can be used, as well as appreviations s, sin, c, cos, t, tan and sq.

Parameters

- **parent** (*wx.Panel*) – name of panel or frame that will be the parent to the TextCtrl. Can be None.
- **loc** (*dict/list*) – the dict or list with the initial value to be placed in the TextCtrl.
- **key** (*int/str*) – the dict key or the list index for the value to be edited by the TextCtrl. The `loc[key]` element must exist, but may have value None. If None, the type for the element is taken from `typeHint` and the value for the control is set initially blank (and thus invalid.) This is a way to specify a field without a default value: a user must set a valid value. If the value is not None, it must have a base type of int, float, str or unicode; the TextCrtl will be initialized from this value.
- **notBlank** (*bool*) – if True (default) blank values are invalid for str inputs.
- **min** (*number*) – minimum allowed valid value. If None (default) the lower limit is unbounded.

- **max** (*number*) – maximum allowed valid value. If None (default) the upper limit is unbounded
- **OKcontrol** (*function*) – specifies a function or method that will be called when the input is validated. The called function is supplied with one argument which is False if the TextCtrl contains an invalid value and True if the value is valid. Note that this function should check all values in the dialog when True, since other entries might be invalid. The default for this is None, which indicates no function should be called.
- **OnLeave** (*function*) – specifies a function or method that will be called when the focus for the control is lost. The called function is supplied with (at present) three keyword arguments:
  - invalid: (*bool*) True if the value for the TextCtrl is invalid
  - value: (*int/float/str*) the value contained in the TextCtrl
  - tc: (*wx.TextCtrl*) the TextCtrl name

  The number of keyword arguments may be increased in the future, if needs arise, so it is best to code these functions with a **kwargs argument so they will continue to run without errors

  The default for OnLeave is None, which indicates no function should be called.
- **typeHint** (*type*) – the value of typeHint is overrides the initial value for the dict/list element `loc[key]`, if set to int or float, which specifies the type for input to the TextCtrl. Defaults as None, which is ignored.
- **CIFinput** (*bool*) – for str input, indicates that only printable ASCII characters may be entered into the TextCtrl. Forces output to be ASCII rather than Unicode. For float and int input, allows use of a single '?' or '.' character as valid input.
- **(other)** – other optional keyword parameters for the wx.TextCtrl widget such as Size or Style may be specified.

EvaluateExpression()
Show the computed value when an expression is entered to the TextCtrl Make sure that the number fits by truncating decimal places and switching to scientific notation, as needed. Called on loss of focus.

ShowStringValidity(*previousInvalid=True*)
Check if input is valid. Anytime the input is invalid, call self.OKcontrol (if defined) because it is fast. If valid, check for any other invalid entries only when changing from invalid to valid, since that is slower.

Parameters previousInvalid (*bool*) – True if the TextCtrl contents were invalid prior to the current change.

class GSASIIgrid.**downdate**(*parent=None*)
Dialog to allow a user to select a version of GSAS-II to install

getVersion()
Get the version number in the dialog

4.2 GSASIIIO: Misc I/O routines

Module with miscellaneous routines for input and output. Many are GUI routines to interact with user.

Includes support for image reading.

Also includes base classes for data import routines.

GSASIIIO.**CheckImageFile**(*G2frame*, *imagefile*)
Get an new image file name if the specified one does not exist

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II Frame and data object
- **imagefile** (*str*) – name of image file

Returns imagefile, if it exists, or the name of a file that does exist or False if the user presses Cancel

class GSASIIIO.**ExportBaseclass**(*G2frame*, *formatName*, *extension*, *longFormatName=None*)
Defines a base class for the exporting of GSAS-II results

CloseFile(*fp=None*)
Close a file opened in OpenFile

Parameters **fp** (*file*) – the file object to be closed. If None (default) file object self.fp is closed.

GetAtoms(*phasenam*)
Gets the atoms associated with a phase. Can be used with standard or macromolecular phases

Parameters **phasenam** (*str*) – the name for the selected phase

Returns

a list of items for eac atom where each item is a list containing: label, typ, mult, xyz, and td, where

- label and typ are the atom label and the scattering factor type (str)
- mult is the site multiplicity (int)
- xyz is contains a list with four pairs of numbers: x, y, z and fractional occupancy and their standard uncertainty (or a negative value)
- td is contains a list with either one or six pairs of numbers: if one number it is U_{iso} and with six it is U_{11}, U_{22}, U_{33}, U_{12}, U_{13} & U_{23} paired with their standard uncertainty (or a negative value)

GetCell(*phasenam*)
Gets the unit cell parameters and their s.u.'s for a selected phase

Parameters **phasenam** (*str*) – the name for the selected phase

Returns *cellList,cellSig* where each is a 7 element list corresponding to a, b, c, alpha, beta, gamma, volume where *cellList* has the cell values and *cellSig* has their uncertainties.

OpenFile(*fil=None*)
Open the output file

Parameters **fil** (*str*) – The name of the file to open. If None (default) the name defaults to self.filename.

Returns the file object opened by the routine which is also saved as self.fp

SetupExport(*event*, *AskFile=True*)
Determines the type of menu that called the Exporter. Selects histograms or phases when needed.

Parameters **AskFile** (*bool*) – if AskFile is True (default) get the name of the file in a dialog

Returns True in case of an error

Write(*line*)
write a line of output, attaching a line-end character

Parameters **line** (*str*) – the text to be written.

askSaveFile()
Ask the user to supply a file name

Returns a file name (str)

dumpTree(*mode='type'*)
Print out information on the data tree dicts loaded in loadTree

loadParmDict()
Load the GSAS-II refinable parameters from the tree into a dict (self.parmDict). Update refined values to those from the last cycle and set the uncertainties for the refined parameters in another dict (self.sigDict).

Expands the parm & sig dicts to include values derived from constraints.

loadTree(*histType=None*)
Load the contents of the data tree into a set of dicts (self.OverallParms, self.Phases and self.Histogram as well as self.powderDict & self.xtalDict)

- The childrenless data tree items are overall parameters/controls for the entire project and are placed in self.OverallParms
- Phase items are placed in self.Phases
- Data items are placed in self.Histogram. The key for these data items begin with a keyword, such as PWDR, IMG, HKLF,... that identifies the data type.

GSASIIIO.**ExtractFileFromZip**(*filename*, *selection=None*, *confirmread=True*, *confirmoverwrite=True*, *parent=None*, *multipleselect=False*)
If the filename is a zip file, extract a file from that archive.

Parameters

- **Selection** (*list*) – used to predefine the name of the file to be extracted. Filename case and zip directory name are ignored in selection; the first matching file is used.
- **confirmread** (*bool*) – if True asks the user to confirm before expanding the only file in a zip
- **confirmoverwrite** (*bool*) – if True asks the user to confirm before overwriting if the extracted file already exists
- **multipleselect** (*bool*) – if True allows more than one zip file to be extracted, a list of file(s) is returned. If only one file is present, do not ask which one, otherwise offer a list of choices (unless selection is used).

Returns the name of the file that has been created or a list of files (see multipleselect)

If the file is not a zipfile, return the name of the input file. If the zipfile is empty or no file has been selected, return None

GSASIIIO.**FileDlgFixExt**(*dlg*, *file*)
this is needed to fix a problem in linux wx.FileDialog

GSASIIIO.**GetEdfData**(*filename*, *imageOnly=False*)
Read European detector data edf file

GSASIIIO.**GetG2Image**(*filename*)
Read an image as a python pickle

GSASIIIO.**GetGEsumData**(*filename*, *imageOnly=False*)
Read SUM file as produced at 1-ID from G.E. images

GSASIIIO.**GetImageData**(*G2frame*, *imagefile*, *imageOnly=False*)
Read an image with the file reader keyed by the file extension

Parameters

- **G2frame** (*wx.Frame*) – main GSAS-II Frame and data object
- **imagefile** (*str*) – name of image file
- **imageOnly** (*bool*) – If True return only the image, otherwise (default) return more (see below)

Returns an image as a numpy array or a list of four items: Comments, Data, Npix and the Image, as selected by imageOnly

GSASIIIO.**GetImgData**(*filename*, *imageOnly=False*)
Read an ADSC image file

GSASIIIO.**GetMAR345Data**(*filename*, *imageOnly=False*)
Read a MAR-345 image plate image

GSASIIIO.**GetPowderPeaks**(*fileName*)
Read powder peaks from a file

GSASIIIO.**GetTifData**(*filename*, *imageOnly=False*)
Read an image in a pseudo-tif format, as produced by a wide variety of software, almost always incorrectly in some way.

class GSASIIIO.**ImportBaseclass**(*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)
Defines a base class for the reading of input files (diffraction data, coordinates,...

BlockSelector(*ChoiceList*, *ParentFrame=None*, *title='Select a block'*, *size=None*, *header='Block Selector'*, *useCancel=True*)
Provide a wx dialog to select a block if the file contains more than one set of data and one must be selected

ContentsValidator(*filepointer*)
This routine will attempt to determine if the file can be read with the current format. This will typically be overridden with a method that takes a quick scan of [some of] the file contents to do a "sanity" check if the file appears to match the selected format. Expected to be called via self.Validator()

ExtensionValidator(*filename*)
This methods checks if the file has the correct extension Return False if this filename will not be supported by this reader Return True if the extension matches the list supplied by the reader Return None if the reader allows un-registered extensions

MultipleBlockSelector(*ChoiceList*, *ParentFrame=None*, *title='Select a block'*, *size=None*, *header='Block Selector'*)
Provide a wx dialog to select a block of data if the file contains more than one set of data and one must be selected.

Returns a list of the selected blocks

MultipleChoicesDialog(*choicelist*, *headinglist*, *ParentFrame=None*, ***kwargs*)
A modal dialog that offers a series of choices, each with a title and a wx.Choice widget. Typical input:

•choicelist=[('a','b','c'), ('test1','test2'),('no choice',)]

•headinglist = ['select a, b or c', 'select 1 of 2', 'No option here']

optional keyword parameters are: head (window title) and title returns a list of selected indicies for each choice (or None)

class GSASIIIO.**ImportPhase**(*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)
Defines a base class for the reading of files with coordinates

PhaseSelector(*ChoiceList*, *ParentFrame=None*, *title='Select a phase'*, *size=None*, *header='Phase Selector'*)
Provide a wx dialog to select a phase if the file contains more than one phase

class GSASIIIO.**ImportPowderData**(*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)
Defines a base class for the reading of files with powder data

powderdata = None
A powder data set is a list with items [x,y,w,yc,yb,yd]: np.array(x), # x-axis values np.array(y), # powder pattern intensities np.array(w), # 1/sig(intensity)^2 values (weights) np.array(yc), # calc. intensities (zero) np.array(yb), # calc. background (zero) np.array(yd), # obs-calc profiles

class GSASIIIO.**ImportStructFactor**(*formatName*, *longFormatName=None*, *extensionlist=[]*, *strictExtension=False*)
Defines a base class for the reading of files with tables of structure factors

Note that the default controls are stored in self.Controls and the default instrument parameters are stored in self.Parameters. These can be changed, but any changes will be the defaults for all subsequent uses of the ImportStructFactor derived classes until InitControls() and InitParameters() are called. Probably better to use UpdateControls() and UpdateControls() (adding new args if needed) to change values.

InitControls()
initialize the controls structure

InitParameters()
initialize the instrument parameters structure

UpdateControls(*Type='Fosq'*, *FcalcPresent=False*)
Scan through the reflections to update the Controls dictionary

UpdateParameters(*Type=None*, *Wave=None*)
Revise the instrument parameters

GSASIIIO.**IndexPeakListSave**(*G2frame*, *peaks*)
Save powder peaks from the indexing list

class GSASIIIO.**MultipleChoicesDialog**(*choicelist*, *headinglist*, *head='Select options'*, *title='Please select from options below'*, *parent=None*)
A dialog that offers a series of choices, each with a title and a wx.Choice widget. Intended to be used Modally. typical input:

- choicelist=[('a','b','c'), ('test1','test2'),('no choice',)]
- headinglist = ['select a, b or c', 'select 1 of 2', 'No option here']

selections are placed in self.chosen when OK is pressed

GSASIIIO.**PDFSave**(*G2frame*, *exports*)
Save a PDF G(r) and S(Q) in column formats

GSASIIIO.**PeakListSave**(*G2frame*, *file*, *peaks*)
Save powder peaks to a data file

GSASIIIO.**ProjFileOpen**(*G2frame*)
Read a GSAS-II project file

GSASIIIO.**ProjFileSave**(*G2frame*)
Save a GSAS-II project file

GSASIIIO.**PutG2Image**(*filename*, *Comments*, *Data*, *Npix*, *image*)
Write an image as a python pickle

GSASIIIO.**ReadCIF**(*URLorFile*)
Open a CIF, which may be specified as a file name or as a URL using PyCifRW (from James Hester). The open routine gets confused with DOS names that begin with a letter and colon "C:dir" so this routine will try to open the passed name as a file and if that fails, try it as a URL

Parameters **URLorFile** (*str*) – string containing a URL or a file name. Code will try first to open it as a file and then as a URL.

Returns a PyCifRW CIF object.

GSASIIIO.**ReadEXPPhase**(*G2frame*, *filename*)
Read a phase from a GSAS .EXP file. Called in the GSAS phase import routine (see imports/G2phase.py)

GSASIIIO.**ReadPDBPhase**(*filename*)
Read a phase from a PDB file. Called in the PDB phase import routine (see imports/G2phase.py)

GSASIIIO.**SaveIntegration**(*G2frame*, *PickId*, *data*)
Save image integration results as powder pattern(s)

GSASIIIO.**SetNewPhase**(*Name='New Phase'*, *SGData=None*, *cell=None*)
Create a new phase with default values for various parameters

Parameters

- **Name** (*str*) – Name for new Phase
- **SGData** (*dict*) – space group data from `GSASIIspc.SpcGroup()`; defaults to data for P 1
- **cell** (*list*) – unit cell parameter list; defaults to [1.0,1.0,1.0,90.,90,90.,1.]

GSASIIIO.**powderFxyeSave**(*G2frame*, *exports*, *powderfile*)
Save a powder histogram as a GSAS FXYE file

GSASIIIO.**powderXyeSave**(*G2frame*, *exports*, *powderfile*)
Save a powder histogram as a Topas XYE file

GSASIIIO.**sfloat**(*S*)
Convert a string to float. An empty field is treated as zero

GSASIIIO.**sint**(*S*)
Convert a string to int. An empty field is treated as zero

GSASIIIO.**trim**(*val*)
Simplify a string containing leading and trailing spaces as well as newlines, tabs, repeated spaces etc. into a shorter and more simple string, by replacing all ranges of whitespace characters with a single space.

Parameters **val** (*str*) – the string to be simplified

Returns the (usually) shortened version of the string

4.3 *ReadMarCCDFrame: Read Mar Files*

class ReadMarCCDFrame.**marFrame**(*File*, *byteOrd='<'*, *IFD={}*)
A class to extract correct mar header and image info from a MarCCD file

Parameters

- **File** (*str*) – file object [from open()]
- **byteOrd** – '<' (default) or '>'
- **IFD** (*dict*) – ?

4.4 GSASIIpy3: Python 3.x Routines

Module to hold python 3-compatible code, to keep it separate from code that will break with __future__ options.

GSASIIpy3.**FormatValue**(*val*, *maxdigits=10*)

Format a float to fit in `maxdigits` spaces, showing as much precision as possible, more or less.

Parameters

- **val** (*float*) – number to be formatted.
- **maxdigits** (*int*) – the number of digits to be used for display of the number (defaults to 10).

Returns a string with <= maxdigits characters (I hope).

GSASIIpy3.**FormulaEval**(*string*)

Evaluates a algebraic formula into a float, if possible. Works properly on fractions e.g. 2/3 only with python 3.0+ division.

Expressions such as 2/3, 3*pi, sin(45)/2, 2*sqrt(2), 2**10 can all be evaluated.

Parameters **string** (*str*) – Character string containing a Python expression to be evaluated.

Returns the value for the expression as a float or None if the expression does not evaluate to a valid number.

CHAPTER

FIVE

GSAS-II GUI SUBMODULES

5.1 GSASIIphsGUI: Phase GUI

Module to create the GUI for display of phase information in the data display window when a phase is selected. Phase information is stored in one or more *Phase Tree Item* objects. Note that there are functions that respond to some tabs in the phase GUI in other modules (such as GSASIIddata).

GSASIIphsGUI.**UpdatePhaseData**(*G2frame*, *Item*, *data*, *oldPage*)

Create the data display window contents when a phase is clicked on in the man (data tree) window. Called only from `GSASIIgrid.MovePatternTreeToGrid()`, which in turn is called from `GSASII.GSASII.OnPatternTreeSelChanged()` when a tree item is selected.

Parameters

- **G2frame** (*wx.frame*) – the main GSAS-II frame object
- **Item** (*wx.TreeItemId*) – the tree item that was selected
- **data** (*dict*) – all the information on the phase in a dictionary
- **oldPage** (*int*) – This sets a tab to select when moving from one phase to another, in which case the same tab is selected to display first. This is set only when the previous data tree selection is a phase, if not the value is None. The default action is to bring up the General tab.

5.2 GSASIIddataGUI: Phase Diffraction Data GUI

Module to create the GUI for display of diffraction data * phase information that is shown in the data display window (when a phase is selected.)

GSASIIddataGUI.**UpdateDData**(*G2frame*, *DData*, *data*)

Display the Diffraction Data associated with a phase (items where there is a value for each histogram and phase)

Parameters

- **G2frame** (*wx.frame*) – the main GSAS-II frame object
- **DData** (*wx.ScrolledWindow*) – notebook page to be used for the display
- **data** (*dict*) – all the information on the phase in a dictionary

5.3 GSASIIElemGUI: GUI to select and delete element lists

Module to select elements from a periodic table and to delete an element from a list of selected elements.

class GSASIIElemGUI.**DeleteElement**(*parent*, *choice*)
Delete element from selected set widget

ElButton(*id*, *name*, *pos*)
Needs a doc string

class GSASIIElemGUI.**PickElement**(*parent*, *oneOnly=False*, *ifNone=False*)
Makes periodic table widget for picking element - caller maintains element list

ElButton(*name*, *pos*, *tip*, *color*)
Needs a doc string

5.4 GSASIIconstrGUI: Constraint GUI routines

Used to define constraints and rigid bodies.

class GSASIIconstrGUI.**MultiIntegerDialog**(*parent*, *title*, *prompts*, *values*)
Input a series of integers based on prompts

GSASIIconstrGUI.**UpdateConstraints**(*G2frame*, *data*)
Called when Constraints tree item is selected. Displays the constraints in the data window

GSASIIconstrGUI.**UpdateRigidBodies**(*G2frame*, *data*)
Called when Rigid bodies tree item is selected. Displays the rigid bodies in the data window

5.5 GSASIIimgGUI: Image GUI

Control image display and processing

GSASIIimgGUI.**UpdateImageControls**(*G2frame*, *data*, *masks*)
Shows and handles the controls on the "Image Controls" data tree entry

GSASIIimgGUI.**UpdateMasks**(*G2frame*, *data*)
Shows and handles the controls on the "Masks" data tree entry

GSASIIimgGUI.**UpdateStressStrain**(*G2frame*, *data*)
Shows and handles the controls on the "Stress/Strain" data tree entry

5.6 GSASIIpwdGUI: Powder Pattern GUI routines

Used to define GUI controls for the routines that interact with the powder histogram (PWDR) data tree items.

GSASIIpwdGUI.**IsHistogramInAnyPhase**(*G2frame*, *histoName*)
Needs a doc string

GSASIIpwdGUI.**SetDefaultSample**()
Needs a doc string

GSASIIpwdGUI.**UpdateBackground**(*G2frame*, *data*)
respond to selection of PWDR background data tree item.

GSASIIpwdGUI.**UpdateIndexPeaksGrid**(*G2frame*, *data*)
respond to selection of PWDR Index Peak List data tree item.

GSASIIpwdGUI.**UpdateInstrumentGrid**(*G2frame*, *data*)
respond to selection of PWDR Instrument Parameters data tree item.

GSASIIpwdGUI.**UpdateLimitsGrid**(*G2frame*, *data*)
respond to selection of PWDR Limits data tree item.

GSASIIpwdGUI.**UpdatePDFGrid**(*G2frame*, *data*)
respond to selection of PWDR PDF data tree item.

GSASIIpwdGUI.**UpdatePeakGrid**(*G2frame*, *data*)
respond to selection of PWDR powder peaks data tree item.

GSASIIpwdGUI.**UpdateReflectionGrid**(*G2frame*, *data*, *HKLF=False*, *Name=''*)
respond to selection of PWDR Reflections data tree item.

GSASIIpwdGUI.**UpdateSampleGrid**(*G2frame*, *data*)
respond to selection of PWDR Sample Parameters data tree item.

GSASIIpwdGUI.**UpdateUnitCellsGrid**(*G2frame*, *data*)
respond to selection of PWDR Unit Cells data tree item.

5.7 GSASIIrestrGUI: Restraint GUI routines

Used to define restraints.

GSASIIrestrGUI.**UpdateRestraints**(*G2frame*, *data*, *Phases*, *phaseName*)
Respond to selection of the Restraints item on the data tree

CHAPTER

SIX

GSAS-II STRUCTURE SUBMODULES

6.1 GSASIIstrMain: main structure routine

GSASIIstrMain.**BestPlane**(*PlaneData*)
Needs a doc string

GSASIIstrMain.**DisAglTor**(*DATData*)
Needs a doc string

GSASIIstrMain.**PrintDistAngle**(*DisAglCtls*, *DisAglData*, *out=<open file '<stdout>', mode 'w' at 0x240078>*)
Print distances and angles

Parameters

- **DisAglCtls** (*dict*) – contains distance/angle radii usually defined using GSASIIgrid.DisAglDialog()
- **DisAglData** (*dict*) – contains phase data: Items 'OrigAtoms' and 'TargAtoms' contain the atoms to be used for distance/angle origins and atoms to be used as targets. Item 'SGData' has the space group information (see *Space Group object*)
- **out** (*file*) – file object for output. Defaults to sys.stdout.

GSASIIstrMain.**Refine**(*GPXfile*, *dlg*)
Needs a doc string

GSASIIstrMain.**RetDistAngle**(*DisAglCtls*, *DisAglData*)
Compute and return distances and angles

Parameters

- **DisAglCtls** (*dict*) – contains distance/angle radii usually defined using GSASIIgrid.DisAglDialog()
- **DisAglData** (*dict*) – contains phase data: Items 'OrigAtoms' and 'TargAtoms' contain the atoms to be used for distance/angle origins and atoms to be used as targets. Item 'SGData' has the space group information (see *Space Group object*)

Returns

AtomLabels,DistArray,AngArray where:

AtomLabels is a dict of atom labels, keys are the atom number

DistArray is a dict keyed by the origin atom number where the value is a list of distance entries. The value for each distance is a list containing:

0. the target atom number (int);
1. the unit cell offsets added to x,y & z (tuple of int values)
2. the symmetry operator number (which may be modified to indicate centering and center of symmetry)
3. an interatomic distance in A (float)
4. an uncertainty on the distance in A or 0.0 (float)

AngArray is a dict keyed by the origin (central) atom number where the value is a list of angle entries. The value for each angle entry consists of three values:

0. a distance item reference for one neighbor (int)
1. a distance item reference for a second neighbor (int)
2. a angle, uncertainty pair; the s.u. may be zero (tuple of two floats)

The AngArray distance reference items refer directly to the index of the items in the DistArray item for the list of distances for the central atom.

GSASIIstrMain.**SeqRefine**(*GPXfile*, *dlg*)
Needs a doc string

GSASIIstrMain.**main**()
Needs a doc string

6.2 GSASIIstrMath - structure math routines

GSASIIstrMath.**ApplyRBModelDervs**(*dFdvDict*, *parmDict*, *rigidbodyDict*, *Phase*)
Needs a doc string

GSASIIstrMath.**ApplyRBModels**(*parmDict*, *Phases*, *rigidbodyDict*, *Update=False*)
Takes RB info from RBModels in Phase and RB data in rigidbodyDict along with current RB values in parmDict & modifies atom contents (xyz & Uij) of parmDict

GSASIIstrMath.**ApplyXYZshifts**(*parmDict*, *varyList*)
takes atom x,y,z shift and applies it to corresponding atom x,y,z value

Parameters

- **parmDict** (*dict*) – parameter dictionary
- **varyList** (*list*) – list of variables

Returns newAtomDict - dictionary of new atomic coordinate names & values; key is parameter shift name

GSASIIstrMath.**Dict2Values**(*parmdict*, *varylist*)
Use before call to leastsq to setup list of values for the parameters in parmdict, as selected by key in varylist

GSASIIstrMath.**GetAbsorb**(*refl*, *hfx*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetAbsorbDerv**(*refl*, *hfx*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetAtomFXU**(*pfx*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetFobsSq**(*Histograms*, *Phases*, *parmDict*, *calcControls*)
Needs a doc string

GSASIIstrMath.**GetHStrainShift**(*refl*, *SGData*, *phfx*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetHStrainShiftDerv**(*refl*, *SGData*, *phfx*)
Needs a doc string

GSASIIstrMath.**GetIntensityCorr**(*refl*, *uniq*, *G*, *g*, *pfx*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetIntensityDerv**(*refl*, *uniq*, *G*, *g*, *pfx*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetNewCellParms**(*parmDict*, *varyList*)
Needs a doc string

GSASIIstrMath.**GetPrefOri**(*refl*, *uniq*, *G*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetPrefOriDerv**(*refl*, *uniq*, *G*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetReflPos**(*refl*, *wave*, *G*, *hfx*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetReflPosDerv**(*refl*, *wave*, *A*, *hfx*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetSampleSigGam**(*refl*, *wave*, *G*, *GB*, *phfx*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**GetSampleSigGamDerv**(*refl*, *wave*, *G*, *GB*, *phfx*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**HessRefine**(*values*, *HistoPhases*, *parmDict*, *varylist*, *calcControls*, *pawleyLookup*, *dlg*)
Loop over histograms and compute derivatives of the fitting model (M) with respect to all parameters. For each histogram, the Jacobian matrix, dMdv, with dimensions (n by m) where n is the number of parameters and m is the number of data points *in the histogram*. The (n by n) Hessian is computed from each Jacobian and it is returned. This routine is used when refinement derivatives are selected as "analytic Hessian" in Controls.

Returns Vec,Hess where Vec is the least-squares vector and Hess is the Hessian

GSASIIstrMath.**SCExtinction**(*ref*, *phfx*, *hfx*, *pfx*, *calcControls*, *parmDict*, *varyList*)
Single crystal extinction function; puts correction in ref[13] and returns corrections needed for derivatives

GSASIIstrMath.**SHPOcal**(*refl*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
spherical harmonics preferred orientation (cylindrical symmetry only)

GSASIIstrMath.**SHPOcalDerv**(*refl*, *g*, *phfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
spherical harmonics preferred orientation derivatives (cylindrical symmetry only)

GSASIIstrMath.**SHTXcal**(*refl*, *g*, *pfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
Spherical harmonics texture

GSASIIstrMath.**SHTXcalDerv**(*refl*, *g*, *pfx*, *hfx*, *SGData*, *calcControls*, *parmDict*)
Spherical harmonics texture derivatives

GSASIIstrMath.**StructureFactor**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)
Compute structure factors for all h,k,l for phase puts the result, F^2, in each ref[8] in refList input:

Parameters

- **refDict** (*dict*) – where 'RefList' list where each ref = h,k,l,m,d,... 'FF' dict of form factors - filed in below
- **G** (*np.array*) – reciprocal metric tensor
- **pfx** (*str*) – phase id string
- **SGData** (*dict*) – space group info. dictionary output from SpcGroup
- **calcControls** (*dict*) –
- **ParmDict** (*dict*) –

GSASIIstrMath.**StructureFactorDerv**(*refDict*, *G*, *hfx*, *pfx*, *SGData*, *calcControls*, *parmDict*)
Needs a doc string

GSASIIstrMath.**Values2Dict**(*parmdict*, *varylist*, *values*)
Use after call to leastsq to update the parameter dictionary with values corresponding to keys in varylist

GSASIIstrMath.**dervRefine**(*values*, *HistoPhases*, *parmDict*, *varylist*, *calcControls*, *pawleyLookup*, *dlg*)
Loop over histograms and compute derivatives of the fitting model (M) with respect to all parameters. Results are returned in a Jacobian matrix (aka design matrix) of dimensions (n by m) where n is the number of parameters and m is the number of data points. This can exceed memory when m gets large. This routine is used when refinement derivatives are selected as "analtytic Jacobian" in Controls.

Returns Jacobian numpy.array dMdv for all histograms concatinated

GSASIIstrMath.**errRefine**(*values*, *HistoPhases*, *parmDict*, *varylist*, *calcControls*, *pawleyLookup*, *dlg*)
Needs a doc string

GSASIIstrMath.**getPowderProfile**(*parmDict*, *x*, *varylist*, *Histogram*, *Phases*, *calcControls*, *pawleyLookup*)
Needs a doc string

GSASIIstrMath.**getPowderProfileDerv**(*parmDict*, *x*, *varylist*, *Histogram*, *Phases*, *rigidbodyDict*, *calcControls*, *pawleyLookup*)
Needs a doc string

GSASIIstrMath.**penaltyDeriv**(*pNames*, *pVal*, *HistoPhases*, *parmDict*, *varyList*)
Needs a doc string

GSASIIstrMath.**penaltyFxn**(*HistoPhases*, *parmDict*, *varyList*)
Needs a doc string

6.3 GSASIIstrIO: structure I/O routines

GSASIIstrIO.**CheckConstraints**(*GPXfile*)
Load constraints and related info and return any error or warning messages

GSASIIstrIO.**GPXBackup**(*GPXfile*, *makeBack=True*)
makes a backup of the current .gpx file (?)

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **makeBack** (*bool*) – if True (default), the backup is written to a new file; if False, the last backup is overwritten

Returns the name of the backup file that was written

GSASIIstrIO.**GetAllPhaseData**(*GPXfile*, *PhaseName*)
Returns the entire dictionary for PhaseName from GSASII gpx file

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **PhaseName** (*str*) – phase name

Returns phase dictionary

GSASIIstrIO.**GetConstraints**(*GPXfile*)
Read the constraints from the GPX file and interpret them

GSASIIstrIO.**GetControls**(*GPXfile*)
Returns dictionary of control items found in GSASII gpx file

Parameters **GPXfile** (*str*) – full .gpx file name

Returns dictionary of control items

GSASIIstrIO.**GetFprime**(*controlDict*, *Histograms*)
Needs a doc string

GSASIIstrIO.**GetHistogramData**(*Histograms*, *Print=True*, *pFile=None*)
needs a doc string

GSASIIstrIO.**GetHistogramNames**(*GPXfile*, *hType*)
Returns a list of histogram names found in GSASII gpx file

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **hType** (*str*) – list of histogram types

Returns list of histogram names (types = PWDR & HKLF)

GSASIIstrIO.**GetHistogramPhaseData**(*Phases*, *Histograms*, *Print=True*, *pFile=None*, *resetRefList=True*)
Loads the HAP histogram/phase information into dicts

Parameters

- **Phases** (*dict*) – phase information
- **Histograms** (*dict*) – Histogram information
- **Print** (*bool*) – prints information as it is read
- **pFile** (*file*) – file object to print to (the default, None causes printing to the console)
- **resetRefList** (*bool*) – Should the contents of the Reflection List be initialized on loading. The default, True, initializes the Reflection List as it is loaded.

Returns (hapVary,hapDict,controlDict) * hapVary: list of refined variables * hapDict: dict with refined variables and their values * controlDict: dict with computation controls (?)

GSASIIstrIO.**GetHistograms**(*GPXfile*, *hNames*)
Returns a dictionary of histograms found in GSASII gpx file

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **hNames** (*str*) – list of histogram names

Returns dictionary of histograms (types = PWDR & HKLF)

GSASIIstrIO.**GetPawleyConstr**(*SGLaue*, *PawleyRef*, *pawleyVary*)
needs a doc string

GSASIIstrIO.**GetPhaseData**(*PhaseData*, *RestraintDict={}*, *rbIds={}*, *Print=True*, *pFile=None*)
needs a doc string

GSASIIstrIO.**GetPhaseNames**(*GPXfile*)
Returns a list of phase names found under 'Phases' in GSASII gpx file

Parameters **GPXfile** (*str*) – full .gpx file name

Returns list of phase names

GSASIIstrIO.**GetRestraints**(*GPXfile*)
Read the restraints from the GPX file. Throws an exception if not found in the .GPX file

GSASIIstrIO.**GetRigidBodies**(*GPXfile*)
Read the rigid body models from the GPX file

GSASIIstrIO.**GetRigidBodyModels**(*rigidbodyDict*, *Print=True*, *pFile=None*)
needs a doc string

GSASIIstrIO.**GetUsedHistogramsAndPhases**(*GPXfile*)
Returns all histograms that are found in any phase and any phase that uses a histogram

Parameters **GPXfile** (*str*) – full .gpx file name

Returns

(Histograms,Phases)

- Histograms = dictionary of histograms as {name:data,...}
- Phases = dictionary of phases that use histograms

GSASIIstrIO.**PrintRestraints**(*cell*, *SGData*, *AtPtrs*, *Atoms*, *AtLookup*, *textureData*, *phaseRest*, *pFile*)
needs a doc string

GSASIIstrIO.**ProcessConstraints**(*constList*)
Interpret the constraints in the constList input into a dictionary, etc.

Parameters **constList** (*list*) – a list of lists where each item in the outer list specifies a constraint of some form. The last item in each inner list determines which of the four constraints types has been input:

- h (hold): a single variable that will not be varied. It will be removed from the varyList later.
- c (constraint): specifies a linear relationship that can be varied as a new grouped variable a fixed value.
- f (fixed): specifies a linear relationship that is assigned a fixed value.
- e (equivalence): specifies a series of variables where the first variable in the last can be used to generate the values for all the remaining variables.

Returns

a tuple of (constDict,fixedList,ignored) where:

- constDict (list) contains the constraint relationships
- fixedList (list) contains the fixed values for type of constraint.
- ignored (int) counts the number of invalid constraint items (should always be zero!)

GSASIIstrIO.**SetHistogramData**(*parmDict*, *sigDict*, *Histograms*, *Print=True*, *pFile=None*)
needs a doc string

GSASIIstrIO.**SetHistogramPhaseData**(*parmDict*, *sigDict*, *Phases*, *Histograms*, *Print=True*, *pFile=None*)
needs a doc string

GSASIIstrIO.**SetPhaseData**(*parmDict*, *sigDict*, *Phases*, *RBIds*, *covData*, *RestraintDict=None*, *pFile=None*)
needs a doc string

GSASIIstrIO.**SetRigidBodyModels**(*parmDict*, *sigDict*, *rigidbodyDict*, *pFile=None*)
needs a doc string

GSASIIstrIO.**SetSeqResult**(*GPXfile*, *Histograms*, *SeqResult*)
Needs doc string

Parameters **GPXfile** (*str*) – full .gpx file name

GSASIIstrIO.**SetUsedHistogramsAndPhases**(*GPXfile*, *Histograms*, *Phases*, *RigidBodies*, *CovData*, *makeBack=True*)
Updates gpxfile from all histograms that are found in any phase and any phase that used a histogram. Also updates rigid body definitions.

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **Histograms** (*dict*) – dictionary of histograms as {name:data,...}
- **Phases** (*dict*) – dictionary of phases that use histograms
- **RigidBodies** (*dict*) – dictionary of rigid bodies
- **CovData** (*dict*) – dictionary of refined variables, varyList, & covariance matrix
- **makeBack** (*bool*) – True if new backup of .gpx file is to be made; else use the last one made

GSASIIstrIO.**ShowBanner**(*pFile=None*)
Print authorship, copyright and citation notice

GSASIIstrIO.**ShowControls**(*Controls*, *pFile=None*)
Print controls information

GSASIIstrIO.**cellFill**(*pfx*, *SGData*, *parmDict*, *sigDict*)
Returns the filled-out reciprocal cell (A) terms and their uncertainties from the parameter and sig dictionaries.

Parameters

- **pfx** (*str*) – parameter prefix ("n::", where n is a phase number)
- **SGdata** (*dict*) – a symmetry object
- **parmDict** (*dict*) – a dictionary of parameters
- **sigDict** (*dict*) – a dictionary of uncertainties on parameters

Returns A,sigA where each is a list of six terms with the A terms

GSASIIstrIO.**cellVary**(*pfx*, *SGData*)
needs a doc string

GSASIIstrIO.**getBackupName**(*GPXfile*, *makeBack*)
Get the name for the backup .gpx file name

Parameters

- **GPXfile** (*str*) – full .gpx file name
- **makeBack** (*bool*) – if True the name of a new file is returned, if False the name of the last file that exists is returned

Returns the name of a backup file

GSASIIstrIO.**getCellEsd**(*pfx*, *SGData*, *A*, *covData*)
needs a doc string

CHAPTER

SEVEN

GSASIIMAPVARS: PARAMETER CONSTRAINTS

Module to implements algebraic contraints, parameter redefinition and parameter simplification contraints.

Parameter redefinition (new vars) is done by creating one or more relationships between a set of parameters

```
Mx1 * Px + My1 * Py +...
Mx2 * Px + Mz2 * Pz + ...
```

where Pj is a parameter name and Mjk is a constant.

Constant constraint Relations can also be supplied in the form of an equation:

```
nx1 * Px + ny1 * Py +... = C1
```

where Cn is a constant. These equations define an algebraic constrant.

Parameters can also be "fixed" (held), which prevents them from being refined.

All of the above three cases are input using routines GroupConstraints and GenerateConstraints. The input consists of a list of relationship dictionaries:

```
constrDict = [
     {'0:12:Scale': 2.0, '0:14:Scale': 4.0, '0:13:Scale': 3.0, '0:0:Scale': 0.5},
     {'2::C(10,6,1)': 1.0, '1::C(10,6,1)': 1.0},
     {'0::A0': 0.0}]
fixedList = ['5.0', None, '0']
```

Where the dictionary defines the first part of an expression and the corresponding fixedList item is either None (for parameter redefinition) or the constant value, for a constant constraint equation. A dictionary that contains a single term defines a variable that will be fixed (held). The multiplier and the fixedList value in this case are ignored.

Parameters can also be equivalenced or "slaved" to another parameter, such that one (independent) parameter is equated to several (now dependent) parameters. In algebraic form this is:

```
P0 = M1 * P1 = M2 * P2 = ...
```

Thus parameters P0, P1 and P2,... are linearly equivalent. Routine StoreEquivalence is used to specify these equivalences.

Parameter redefinition (new vars) describes a new, independent, parameter, which is defined in terms of dependent parameters that are defined in the Model, while fixed constrained relations effectively reduce the complexity of the Model by removing a degree of freedom. It is possible for a parameter to appear in both a parameter redefinition expression and a fixed constraint equation, but a parameter cannot be used a parameter equivalance cannot be used elsewhere (not fixed, constrained or redefined). Likewise a fixed parameter cannot be used elsewhere (not equivalanced, constrained or redefined).

Relationships are grouped so that a set of dependent parameters appear in only one group (done in routine GroupConstraints.) Note that if a group contains relationships/equations that involve N dependent parameters, there must exist N-C new parameters, where C is the number of contraint equations in the group. Routine GenerateConstraints takes the output from GroupConstraints and generates the "missing" relationships and saves that information in the module's global variables. Each generated parameter is named sequentially using paramPrefix.

A list of parameters that will be varied is specified as input to GenerateConstraints (varyList). A fixed parameter will simply be removed from this list preventing that parameter from being varied. Note that all parameters in a relationship must specified as varied (appear in varyList) or none can be varied. This is checked in GenerateConstraints (as well as for generated relationships in SetVaryFlags).

- If all parameters in a parameter redefinition (new var) relationship are varied, the parameter assigned to this expression (::constr:n, see paramPrefix) newly generated parameter is varied. Note that any generated "missing" relations are not varied. Only the input relations are varied.
- If all parameters in a fixed constraint equation are varied, the generated "missing" relations in the group are all varied. This provides the N-C degrees of freedom.

7.1 *External Routines*

To define a set of constrained and unconstrained relations, one defines a list of dictionary defining constraint parameters and their values, a list of fixed values for each constraint and a list of parameters to be varied. In addition, one uses `StoreEquivalence()` to define parameters that are equivalent. One can then use `CheckConstraints()` to check that the input is internally consistent and finally `GroupConstraints()` and `GenerateConstraints()` to generate the internally used tables. Routines `Map2Dict()` is used to initialize the parameter dictionary and `Dict2Map()`, `Dict2Deriv()`, and `ComputeDepESD()` are used to apply constraints. Routine `VarRemapShow()` is used to print out the constraint information, as stored by `GenerateConstraints()`.

`InitVars()` This is optionally used to clear out all defined previously defined constraint information

`StoreEquivalence()` To implement parameter redefinition, one calls StoreEquivalence. This should be called for every set of equivalence relationships. There is no harm in using StoreEquivalence with the same independent variable:

```
StoreEquivalence('x',('y',))
StoreEquivalence('x',('z',))
```

or equivalently

```
StoreEquivalence('x',('y','z'))
```

The latter will run more efficiently. Note that mixing independent and dependent variables is problematic. This is not allowed:

```
StoreEquivalence('x',('y',))
StoreEquivalence('y',('z',))
```

Use StoreEquivalence before calling GenerateConstraints or CheckConstraints

`CheckConstraints()` To check that input in internally consistent, use CheckConstraints

`Map2Dict()` To determine values for the parameters created in this module, one calls Map2Dict. This will not apply contraints.

`Dict2Map()` To take values from the new independent parameters and constraints, one calls Dict2Map. This will apply contraints.

`Dict2Deriv()` Use Dict2Deriv to determine derivatives on independent parameters from those on dependent ones

ComputeDepESD() Use ComputeDepESD to compute uncertainties on dependent variables

VarRemapShow() To show a summary of the parameter remapping, one calls VarRemapShow

7.2 Global Variables

dependentParmList: contains a list by group of lists of parameters used in the group. Note that parameters listed in dependentParmList should not be refined as they will not affect the model

indParmList: a list of groups of Independent parameters defined in each group. This contains both parameters used in parameter redefinitions as well as names of generated new parameters.

fixedVarList: a list of variables that have been 'fixed' by defining them as equal to a constant (::var: = 0). Note that the constant value is ignored at present. These variables are later removed from varyList which prevents them from being refined. Unlikely to be used externally.

arrayList: a list by group of relationship matrices to relate parameters in dependentParmList to those in indParmList. Unlikely to be used externally.

invarrayList: a list by group of relationship matrices to relate parameters in indParmList to those in dependentParmList. Unlikely to be used externally.

fixedDict: a dictionary containing the fixed values corresponding to parameter equations. The dict key is an ascii string, but the dict value is a float. Unlikely to be used externally.

7.3 Routines

Note that parameter names in GSAS-II are strings of form `<ph>:<hst>:<nam>`

GSASIImapvars.**CheckConstraints**(*varyList*, *constrDict*, *fixedList*)

Takes a list of relationship entries comprising a group of constraints and checks for inconsistencies such as conflicts in parameter/variable definitions and or inconsistently varied parameters.

Parameters

- **varyList** (*list*) – a list of parameters names that will be varied
- **constrDict** (*dict*) – a list of dicts defining relationships/constraints (as defined in GroupConstraints())
- **fixedList** (*list*) – a list of values specifying a fixed value for each dict in constrDict. Values are either strings that can be converted to floats or None if the constraint defines a new parameter rather than a constant.

Returns

two strings:

- the first lists conflicts internal to the specified constraints
- the second lists conflicts where the varyList specifies some parameters in a constraint, but not all

If there are no errors, both strings will be empty

GSASIImapvars.**ComputeDepESD**(*covMatrix*, *varyList*, *parmDict*)

Compute uncertainties for dependent parameters from independent ones returns a dictionary containing the esd values for dependent parameters

GSASIImapvars.**Dict2Deriv**(*varyList*, *derivDict*, *dMdv*)
Compute derivatives for Independent Parameters from the derivatives for the original parameters

Parameters

- **varyList** (*list*) – a list of parameters names that will be varied
- **derivDict** (*dict*) – a dict containing derivatives for parameter values keyed by the parameter names.
- **dMdv** (*list*) – a Jacobian, as a list of np.array containing derivatives for dependent parameter computed from derivDict

GSASIImapvars.**Dict2Map**(*parmDict*, *varyList*)
Applies the constraints defined using `StoreEquivalence()`, `GroupConstraints()` and `GenerateConstraints()` by changing values in a dict containing the parameters. This should be done before the parameters are used for any computations

Parameters

- **parmDict** (*dict*) – a dict containing parameter values keyed by the parameter names. This will contain updated values for both dependent and independent parameters after Dict2Map is called. It will also contain some unexpected entries of every constant value {'0':0.0} & {'1.0':1.0}, which do not cause any problems.
- **varyList** (*list*) – a list of parameters names that will be varied

GSASIImapvars.**GenerateConstraints**(*groups*, *parmlist*, *varyList*, *constrDict*, *fixedList*)
Takes a list of relationship entries comprising a group of constraints and builds the relationship lists and their inverse and stores them in global variables Also checks for internal conflicts or inconsistencies in parameter/variable definitions.

Parameters

- **groups** (*list*) – a list of grouped contraints where each constraint grouped containts a list of indices for constraint constrDict entries, created in `GroupConstraints()` (returned as 1st value)
- **parmlist** (*list*) – a list containing lists of parameter names contained in each group, created in `GroupConstraints()` (returned as 1st value)
- **varyList** (*list*) – a list of parameters names (strings of form `<ph>:<hst>:<nam>`) that will be varied
- **constrDict** (*dict*) – a list of dicts defining relationships/constraints (as defined in `GroupConstraints()`)
- **fixedList** (*list*) – a list of values specifying a fixed value for each dict in constrDict. Values are either strings that can be converted to floats, float values or None if the constraint defines a new parameter
- **constrDict** – a list of dicts defining relationships/constraints

GSASIImapvars.**GetDependentVars**()
Return a list of dependent variables: e.g. variables that are constrained in terms of other variables

Returns a list of variable names

GSASIImapvars.**GetIndependentVars**()
Return a list of independent variables: e.g. variables that are created by constraints of other variables

Returns a list of variable names

GSASIImapvars.**GramSchmidtOrtho**(*a*, *nkeep=0*)
Use the Gram-Schmidt process (http://en.wikipedia.org/wiki/Gram-Schmidt) to find orthonormal unit vectors relative to first row.

If nkeep is non-zero, the first nkeep rows in the array are not changed

input: arrayin: a 2-D non-singular square array

returns: a orthonormal set of unit vectors as a square array

GSASIImapvars.**GroupConstraints**(*constrDict*)
divide the constraints into groups that share no parameters.

Parameters **constrDict** (*dict*) – a list of dicts defining relationships/constraints

```
constrDict = [{<constr1>}, {<constr2>}, ...]
```

where {<constr1>} is {'param1': mult1, 'param2': mult2,...}

Returns

two lists of lists:

- a list of grouped contraints where each constraint grouped containts a list of indices for constraint constrDict entries
- a list containing lists of parameter names contained in each group

GSASIImapvars.**InitVars**()
Initializes all constraint information

GSASIImapvars.**Map2Dict**(*parmDict*, *varyList*)
Create (or update) the Independent Parameters from the original set of Parameters

Removes dependent variables from the varyList

This should be done once, after the constraints have been defined using `StoreEquivalence()`, `GroupConstraints()` and `GenerateConstraints()` and before any variable refinement is done to complete the parameter dictionary by defining independent parameters and satisfying the constraint equations.

Parameters

- **parmDict** (*dict*) – a dict containing parameter values keyed by the parameter names. This will contain updated values for both dependent and independent parameters after Dict2Map is called. It will also contain some unexpected entries of every constant value {'0':0.0} & {'1.0':1.0}, which do not cause any problems.
- **varyList** (*list*) – a list of parameters names that will be varied

GSASIImapvars.**PrintIndependentVars**(*parmDict*, *varyList*, *sigDict*, *PrintAll=False*, *pFile=None*)
Print the values and uncertainties on the independent variables

GSASIImapvars.**StoreEquivalence**(*independentVar*, *dependentList*)
Takes a list of dependent parameter(s) and stores their relationship to a single independent parameter (independentVar)

Parameters

- **independentVar** (*str*) – name of master parameter that will be used to determine the value to set the dependent variables
- **dependentList** (*list*) – a list of parameters that will set from independentVar. Each item in the list can be a string with the parameter name or a tuple containing a name and multiplier: `['parm1',('parm2',.5),]`

GSASIImapvars.**VarRemapShow**(*varyList*)

List out the saved relationships. This should be done after the constraints have been defined using StoreEquivalence(), GroupConstraints() and GenerateConstraints().

Returns a string containing the details of the contraint relationships

CHAPTER

EIGHT

GSASIIIMAGE: IMAGE CALC MODULE

Ellipse fitting & image integration

needs some minor refactoring to remove wx code

GSASIIimage.**EdgeFinder**(*image*, *data*)
this makes list of all x,y where I>edgeMin suitable for an ellipse search?

GSASIIimage.**Fill2ThetaAzimuthMap**(*masks*, *TA*, *tam*, *image*)
Needs a doc string

GSASIIimage.**FitCircle**(*ring*)
Needs a doc string

GSASIIimage.**FitDetector**(*rings*, *varyList*, *parmDict*)
Needs a doc string

GSASIIimage.**FitEllipse**(*ring*)
Needs a doc string

GSASIIimage.**FitRing**(*ring*, *delta*)
Needs a doc string

GSASIIimage.**FitStrSta**(*Image*, *StrSta*, *Controls*, *Masks*)
Needs a doc string

GSASIIimage.**FitStrain**(*rings*, *p0*, *wave*, *phi*)
Needs a doc string

GSASIIimage.**GetAzm**(*x*, *y*, *data*)
Needs a doc string

GSASIIimage.**GetDetXYfromThAzm**(*Th*, *Azm*, *data*)
Needs a doc string

GSASIIimage.**GetDetectorXY**(*dsp*, *azm*, *data*)
Needs a doc string

GSASIIimage.**GetDsp**(*x*, *y*, *data*)
Needs a doc string

GSASIIimage.**GetEllipse**(*dsp*, *data*)
Needs a doc string

GSASIIimage.**GetTth**(*x*, *y*, *data*)
Needs a doc string

GSASIIimage.**GetTthAzm**(*x*, *y*, *data*)
Needs a doc string

GSASIIimage.**GetTthAzmDsp**(*x*, *y*, *data*)
Needs a doc string

GSASIIimage.**ImageCalibrate**(*self*, *data*)
Needs a doc string

GSASIIimage.**ImageCompress**(*image*, *scale*)
Needs a doc string

GSASIIimage.**ImageIntegrate**(*image*, *data*, *masks*)
Needs a doc string

GSASIIimage.**ImageLocalMax**(*image*, *w*, *Xpix*, *Ypix*)
Needs a doc string

GSASIIimage.**ImageRecalibrate**(*self*, *data*)
Needs a doc string

GSASIIimage.**Make2ThetaAzimuthMap**(*data*, *masks*, *iLim*, *jLim*)
Needs a doc string

GSASIIimage.**calcDist**(*radii*, *tth*)
Needs a doc string

GSASIIimage.**calcFij**(*omg*, *phi*, *azm*, *th*)
Does something...

Uses parameters as defined by Bob He & Kingsley Smith, Adv. in X-Ray Anal. 41, 501 (1997)

Parameters

- **omg** – his omega = sample omega rotation; 0 when incident beam || sample surface, 90 when perp. to sample surface
- **phi** – his phi = sample phi rotation; usually = 0, axis rotates with omg.
- **azm** – his chi = azimuth around incident beam
- **th** – his theta = theta

GSASIIimage.**calcZdisCosB**(*radius*, *tth*, *radii*)
Needs a doc string

GSASIIimage.**checkEllipse**(*Zsum*, *distSum*, *xSum*, *ySum*, *dist*, *x*, *y*)
Needs a doc string

GSASIIimage.**makeIdealRing**(*ellipse*, *azm=None*)
Needs a doc string

GSASIIimage.**makeMat**(*Angle*, *Axis*)
Make rotation matrix from Angle and Axis

Parameters

- **Angle** (*float*) – in degrees
- **Axis** (*int*) – 0 for rotation about x, 1 for about y, etc.

GSASIIimage.**makeRing**(*dsp*, *ellipse*, *pix*, *reject*, *scalex*, *scaley*, *image*)
Needs a doc string

GSASIIimage.**peneCorr**(*tth*, *dep*)
Needs a doc string

GSASIIimage.**pointInPolygon**(*pXY*, *xy*)
Needs a doc string

CHAPTER

NINE

GSASIIMATH: COMPUTATION MODULE

Routines for least-squares minimization and other stuff

GSASIImath.**AV2Q**(*A*, *V*)
convert angle (radians) & vector to quaternion q=r+ai+bj+ck

GSASIImath.**AVdeg2Q**(*A*, *V*)
convert angle (degrees) & vector to quaternion q=r+ai+bj+ck

GSASIImath.**AtomTLS2UIJ**(*atomData*, *atPtrs*, *Amat*, *rbObj*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**AtomUij2TLS**(*atomData*, *atPtrs*, *Amat*, *Bmat*, *rbObj*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**ChargeFlip**(*data*, *reflDict*, *pgbar*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**FillAtomLookUp**(*atomData*)
create a dictionary of atom indexes with atom IDs as keys

Parameters **atomData** (*list*) – Atom table to be used

Returns dict atomLookUp: dictionary of atom indexes with atom IDs as keys

GSASIImath.**FindAtomIndexByIDs**(*atomData*, *IDs*, *Draw=True*)
finds the set of atom array indices for a list of atom IDs. Will search either the Atom table or the drawAtom table.

Parameters

- **atomData** (*list*) – Atom or drawAtom table containting coordinates, etc.
- **IDs** (*list*) – atom IDs to be found
- **Draw** (*bool*) – True if drawAtom table to be searched; False if Atom table is searched

Returns list indx: atom (or drawAtom) indices

GSASIImath.**FourierMap**(*data*, *reflDict*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**GetAngleSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**GetAtomCoordsByID**(*pId*, *parmDict*, *AtLookup*, *indx*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**GetAtomItemsById**(*atomData*, *atomLookUp*, *IdList*, *itemLoc*, *numItems=1*)
gets atom parameters for atoms using atom IDs

Parameters

- **atomData** (*list*) – Atom table to be used
- **atomLookUp** (*dict*) – dictionary of atom indexes with atom IDs as keys
- **IdList** (*list*) – atom IDs to be found
- **itemLoc** (*int*) – pointer to desired 1st item in an atom table entry
- **numItems** (*int*) – number of items to be retrieved

Returns type name: description

GSASIImath.**GetAtomsById**(*atomData*, *atomLookUp*, *IdList*)
gets a list of atoms from Atom table that match a set of atom IDs

Parameters

- **atomData** (*list*) – Atom table to be used
- **atomLookUp** (*dict*) – dictionary of atom indexes with atom IDs as keys
- **IdList** (*list*) – atom IDs to be found

Returns list atoms: list of atoms found

GSASIImath.**GetDATSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**GetDistSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**GetSHCoeff**(*pId*, *parmDict*, *SHkeys*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**GetTorsionSig**(*Oatoms*, *Atoms*, *Amat*, *SGData*, *covData={}*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**GetXYZDist**(*xyz*, *XYZ*, *Amat*)
gets distance from position xyz to all XYZ, xyz & XYZ are np.array and are in crystal coordinates; Amat is crystal to Cart matrix

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**HessianLSQ**(*func*, *x0*, *Hess*, *args=()*, *ftol=1.49012e-08*, *xtol=1.49012e-08*, *maxcyc=0*, *Print=False*)
Minimize the sum of squares of a function (f) evaluated on a series of values (y): $\sum_{y=0}^{N_{obs}} f(y, args)$

```
            Nobs
x = arg min(sum(func(y)**2,axis=0))
            y=0
```

Parameters

- **func** (*function*) – callable method or function should take at least one (possibly length N vector) argument and returns M floating point numbers.
- **x0** (*np.ndarray*) – The starting estimate for the minimization of length N
- **Hess** (*function*) – callable method or function A required function or method to compute the weighted vector and Hessian for func. It must be a symmetric NxN array
- **args** (*tuple*) – Any extra arguments to func are placed in this tuple.
- **ftol** (*float*) – Relative error desired in the sum of squares.
- **xtol** (*float*) – Relative error desired in the approximate solution.
- **maxcyc** (*int*) – The maximum number of cycles of refinement to execute, if -1 refine until other limits are met (ftol, xtol)
- **Print** (*bool*) – True for printing results (residuals & times) by cycle

Returns

(x,cov_x,infodict) where

- x : ndarray The solution (or the result of the last iteration for an unsuccessful call).
- cov_x : ndarray Uses the fjac and ipvt optional outputs to construct an estimate of the jacobian around the solution. `None` if a singular matrix encountered (indicates very flat curvature in some direction). This matrix must be multiplied by the residual standard deviation to get the covariance of the parameter estimates – see curve_fit.
- infodict : dict a dictionary of optional outputs with the keys:
  - 'fvec' : the function evaluated at the output
  - 'num cyc':
  - 'nfev':
  - 'lamMax':

– ‘psing’:

GSASIImath.**OmitMap**(*data*, *reflDict*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**PeaksEquiv**(*data*, *Ind*)
Find the equivalent map peaks for those selected. Works on the contents of data[’Map Peaks’].

Parameters

- **data** – the phase data structure
- **Ind** (*list*) – list of selected peak indices

Returns augmented list of peaks including those related by symmetry to the ones in Ind

GSASIImath.**PeaksUnique**(*data*, *Ind*)
Finds the symmetry unique set of peaks from those selected. Works on the contents of data[’Map Peaks’].

Parameters

- **data** – the phase data structure
- **Ind** (*list*) – list of selected peak indices

Returns the list of symmetry unique peaks from among those given in Ind

GSASIImath.**Q2AV**(*Q*)
convert quaternion to angle (radians 0-2pi) & normalized vector q=r+ai+bj+ck

GSASIImath.**Q2AVdeg**(*Q*)
convert quaternion to angle (degrees 0-360) & normalized vector q=r+ai+bj+ck

GSASIImath.**Q2Mat**(*Q*)
make rotation matrix from quaternion q=r+ai+bj+ck

GSASIImath.**RotateRBXYZ**(*Bmat*, *Cart*, *oriQ*)
rotate & transform cartesian coordinates to crystallographic ones no translation applied. To be used for numerical derivatives

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**SearchMap**(*data*)
Does a search of a density map for peaks meeting the criterion of peak height is greater than mapData[’cutOff’]/100 of mapData[’rhoMax’] where mapData is data[’General’][’mapData’]; the map is also in mapData.

Parameters **data** – the phase data structure

Returns

(peaks,mags,dzeros) where

- peaks : ndarray x,y,z positions of the peaks found in the map
- mags : ndarray the magnitudes of the peaks
- dzeros : ndarray the distance of the peaks from the unit cell origin

GSASIImath.**SetMolCent**(*model*, *RBData*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**TLS2Uij**(*xyz*, *g*, *Amat*, *rbObj*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**UpdateMCSAxyz**(*Bmat*, *MCSA*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**UpdateRBUIJ**(*Bmat*, *Cart*, *RBObj*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**UpdateRBXYZ**(*Bmat*, *RBObj*, *RBData*, *RBType*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**ValEsd**(*value*, *esd=0*, *nTZ=False*)
Format a floating point number with a given level of precision or with in crystallographic format with a "esd", as value(esd). If esd is negative the number is formatted with the level of significant figures appropriate if abs(esd) were the esd, but the esd is not included. if the esd is zero, approximately 6 significant figures are printed. nTZ=True causes "extra" zeros to be removed after the decimal place. for example:

- "1.235(3)" for value=1.2346 & esd=0.003
- "1.235(3)e4" for value=12346. & esd=30
- "1.235(3)e6" for value=0.12346e7 & esd=3000
- "1.235" for value=1.2346 & esd=-0.003
- "1.240" for value=1.2395 & esd=-0.003
- "1.24" for value=1.2395 & esd=-0.003 with nTZ=True
- "1.23460" for value=1.2346 & esd=0.0

Parameters

- **value** (*float*) – number to be formatted
- **esd** (*float*) – uncertainty or if esd < 0, specifies level of precision to be shown e.g. esd=-0.01 gives 2 places beyond decimal
- **nTZ** (*bool*) – True to remove trailing zeros (default is False)

Returns value(esd) or value as a string

GSASIImath.**adjHKLmax**(*SGData*, *Hmax*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**anneal**(*func*, *x0*, *args=()*, *schedule='fast'*, *full_output=0*, *T0=None*, *Tf=1e-12*, *maxeval=None*, *maxaccept=None*, *maxiter=400*, *boltzmann=1.0*, *learn_rate=0.5*, *feps=1e-06*, *quench=1.0*, *m=1.0*, *n=1.0*, *lower=-100*, *upper=100*, *dwell=50*, *slope=0.9*, *ranStart=False*, *ranRange=0.1*, *autoRan=False*, *dlg=None*)

Minimize a function using simulated annealing.

Schedule is a schedule class implementing the annealing schedule. Available ones are 'fast', 'cauchy', 'boltzmann'

Parameters

- **func** (*callable*) – f(x, *args) Function to be optimized.
- **x0** (*ndarray*) – Initial guess.
- **args** (*tuple*) – Extra parameters to *func*.
- **schedule** (*base_schedule*) – Annealing schedule to use (a class).
- **full_output** (*bool*) – Whether to return optional outputs.
- **T0** (*float*) – Initial Temperature (estimated as 1.2 times the largest cost-function deviation over random points in the range).
- **Tf** (*float*) – Final goal temperature.
- **maxeval** (*int*) – Maximum function evaluations.
- **maxaccept** (*int*) – Maximum changes to accept.
- **maxiter** (*int*) – Maximum cooling iterations.
- **learn_rate** (*float*) – Scale constant for adjusting guesses.
- **boltzmann** (*float*) – Boltzmann constant in acceptance test (increase for less stringent test at each temperature).
- **feps** (*float*) – Stopping relative error tolerance for the function value in last four coolings.
- **quench,m,n** (*float*) – Parameters to alter fast_sa schedule.
- **lower,upper** (*float/ndarray*) – Lower and upper bounds on *x*.
- **dwell** (*int*) – The number of times to search the space at each temperature.
- **slope** (*float*) – Parameter for log schedule
- **ranStart=False** (*bool*) – True for set 10% of ranges about x

Returns

(xmin, Jmin, T, feval, iters, accept, retval) where

- xmin (ndarray): Point giving smallest value found.
- Jmin (float): Minimum value of function found.
- T (float): Final temperature.
- feval (int): Number of function evaluations.
- iters (int): Number of cooling iterations.
- accept (int): Number of tests accepted.
- retval (int): Flag indicating stopping condition:
  - 0: Points no longer changing

- 1: Cooled to final temperature
- 2: Maximum function evaluations
- 3: Maximum cooling iterations reached
- 4: Maximum accepted query locations reached
- 5: Final point not the minimum amongst encountered points

Notes: Simulated annealing is a random algorithm which uses no derivative information from the function being optimized. In practice it has been more useful in discrete optimization than continuous optimization, as there are usually better algorithms for continuous optimization problems.

Some experimentation by trying the difference temperature schedules and altering their parameters is likely required to obtain good performance.

The randomness in the algorithm comes from random sampling in numpy. To obtain the same results you can call numpy.random.seed with the same seed immediately before calling scipy.optimize.anneal.

We give a brief description of how the three temperature schedules generate new points and vary their temperature. Temperatures are only updated with iterations in the outer loop. The inner loop is over xrange(dwell), and new points are generated for every iteration in the inner loop. (Though whether the proposed new points are accepted is probabilistic.)

For readability, let d denote the dimension of the inputs to func. Also, let x_old denote the previous state, and k denote the iteration number of the outer loop. All other variables not defined below are input variables to scipy.optimize.anneal itself.

In the 'fast' schedule the updates are

```
u ~ Uniform(0, 1, size=d)
y = sgn(u - 0.5) * T * ((1+ 1/T)**abs(2u-1) -1.0)
xc = y * (upper - lower)
x_new = x_old + xc

c = n * exp(-n * quench)
T_new = T0 * exp(-c * k**quench)
```

In the 'cauchy' schedule the updates are

```
u ~ Uniform(-pi/2, pi/2, size=d)
xc = learn_rate * T * tan(u)
x_new = x_old + xc

T_new = T0 / (1+k)
```

In the 'boltzmann' schedule the updates are

```
std = minimum( sqrt(T) * ones(d), (upper-lower) / (3*learn_rate) )
y ~ Normal(0, std, size=d)
x_new = x_old + learn_rate * y

T_new = T0 / log(1+k)
```

GSASIImath.**calcRamaEnergy**(*phi*, *psi*, *Coeff*=[ ])

Computes pseudo potential energy from a pair of torsion angles and a numerical description of the potential energy surface. Used to create penalty function in LS refinement: $Eval(\phi,\psi) = C[0] * exp(-V/1000)$

where $V = -C[3] * (\phi - C[1])^2 - C[4] * (\psi - C[2])^2 - 2 * (\phi - C[1]) * (\psi - C[2])$

Parameters

- **phi** (*float*) – first torsion angle (ϕ)
- **psi** (*float*) – second torsion angle (ψ)
- **Coeff** (*list*) – pseudo potential coefficients

Returns list (sum,Eval): pseudo-potential difference from minimum & value; sum is used for penalty function.

GSASIImath.**calcTorsionEnergy**(*TOR*, *Coeff*=[ ])
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**findOffset**(*SGData*, *A*, *Fhkl*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getAngSig**(*VA*, *VB*, *Amat*, *SGData*, *covData={}*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getAtomXYZ**(*atoms*, *cx*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getCWgam**(*ins*, *pos*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getCWgamDeriv**(*pos*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getCWsig**(*ins*, *pos*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getCWsigDeriv**(*pos*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getDensity**(*generalData*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getDistDerv**(*Oxyz*, *Txyz*, *Amat*, *Tunit*, *Top*, *SGData*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getMass**(*generalData*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRamaDeriv**(*XYZ*, *Amat*, *Coeff*)
Computes numerical derivatives of torsion angle pair pseudo potential with respect of crystallographic atom coordinates of the 5 atom sequence

Parameters

- **XYZ** (*nparray*) – crystallographic coordinates of 5 atoms
- **Amat** (*nparray*) – crystal to cartesian transformation matrix
- **Coeff** (*list*) – pseudo potential coefficients

Returns list (deriv) derivatives of pseudopotential with respect to 5 atom crystallographic xyz coordinates.

GSASIImath.**getRestAngle**(*XYZ*, *Amat*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRestChiral**(*XYZ*, *Amat*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRestDeriv**(*Func*, *XYZ*, *Amat*, *ops*, *SGData*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRestDist**(*XYZ*, *Amat*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRestPlane**(*XYZ*, *Amat*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRestPolefig**(*ODFln*, *SamSym*, *Grid*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRestPolefigDerv**(*HKL*, *Grid*, *SHCoeff*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getRestRama**(*XYZ*, *Amat*)
Computes a pair of torsion angles in a 5 atom string

Parameters

- **XYZ** (*nparray*) – crystallographic coordinates of 5 atoms
- **Amat** (*nparray*) – crystal to cartesian transformation matrix

Returns list (phi,psi) two torsion angles in degrees

GSASIImath.**getRestTorsion**(*XYZ*, *Amat*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getSyXYZ**(*XYZ*, *ops*, *SGData*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFalpha**(*ins*, *dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFalphaDeriv**(*dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFbeta**(*ins*, *dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFbetaDeriv**(*dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFgamma**(*ins*, *dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFgammaDeriv**(*dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFsig**(*ins*, *dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTOFsigDeriv**(*dsp*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getTorsionDeriv**(*XYZ*, *Amat*, *Coeff*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**getVCov**(*varyNames*, *varyList*, *covMatrix*)
obtain variance-covariance terms for a set of variables. NB: the varyList and covMatrix were saved by the last least squares refinement so they must match.

Parameters

- **varyNames** (*list*) – variable names to find v-cov matric for
- **varyList** (*list*) – full list of all variables in v-cov matrix
- **covMatrix** (*nparray*) – full variance-covariance matrix from the last least squares refinement

Returns nparray vcov: variance-covariance matrix for the variables given in varyNames

GSASIImath.**getWave**(*Parms*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**invQ**(*Q*)
get inverse of quaternion q=r+ai+bj+ck; q* = r-ai-bj-ck

GSASIImath.**makeQuat**(*A*, *B*, *C*)
Make quaternion from rotation of A vector to B vector about C axis

Parameters **A,B,C** (*np.array*) – Cartesian 3-vectors

Returns quaternion & rotation angle in radians q=r+ai+bj+ck

GSASIImath.**mcsaSearch**(*data*, *RBdata*, *reflType*, *reflData*, *covData*, *pgbar*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**normQ**(*QA*)
get length of quaternion & normalize it q=r+ai+bj+ck

GSASIImath.**printRho**(*SGLaue*, *rho*, *rhoMax*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**prodQQ**(*QA*, *QB*)
Grassman quaternion product QA,QB quaternions; q=r+ai+bj+ck

GSASIImath.**prodQVQ**(*Q*, *V*)
compute the quaternion vector rotation qvq-1 = v' q=r+ai+bj+ck

GSASIImath.**randomAVdeg**(*r0*, *r1*, *r2*, *r3*)
create random angle (deg),vector from 4 random number in range (-1,1)

GSASIImath.**randomQ**(*r0*, *r1*, *r2*, *r3*)
create random quaternion from 4 random numbers in range (-1,1)

GSASIImath.**setPeakparms**(*Parms*, *Parms2*, *pos*, *mag*, *ifQ=False*, *useFit=False*)
default doc string

Parameters **name** (*type*) – description

Returns type name: description

GSASIImath.**sortArray**(*data*, *pos*, *reverse=False*)
data is a list of items sort by pos in list; reverse if True

CHAPTER

TEN

GSASIIINDEX: CELL INDEXING MODULE

Cell indexing program: variation on that of A. Coehlo includes cell refinement from peak positions (not zero as yet)

This needs a bit of refactoring to remove the little bit of GUI code referencing wx

GSASIIindex.**A2values**(*ibrav*, *A*)
needs a doc string

GSASIIindex.**DoIndexPeaks**(*peaks*, *wave*, *controls*, *bravais*)
needs a doc string

GSASIIindex.**FitHKL**(*ibrav*, *peaks*, *A*, *Pwr*)
needs a doc string

GSASIIindex.**FitHKLZ**(*wave*, *ibrav*, *peaks*, *A*, *Z*, *Zref*, *Pwr*)
needs a doc string

GSASIIindex.**IndexPeaks**(*peaks*, *HKL*)
needs a doc string

GSASIIindex.**TestData**()
needs a doc string

GSASIIindex.**Values2A**(*ibrav*, *values*)
needs a doc string

GSASIIindex.**calc_M20**(*peaks*, *HKL*)
needs a doc string

GSASIIindex.**findBestCell**(*dlg*, *ncMax*, *A*, *Ntries*, *ibrav*, *peaks*, *V1*)
needs a doc string

GSASIIindex.**getDmax**(*peaks*)
needs a doc string

GSASIIindex.**getDmin**(*peaks*)
needs a doc string

GSASIIindex.**halfCell**(*ibrav*, *A*, *peaks*)
needs a doc string

GSASIIindex.**monoCellReduce**(*ibrav*, *A*)
needs a doc string

GSASIIindex.**oddPeak**(*indx*, *peaks*)
needs a doc string

GSASIIindex.**ran2axis**(*k*, *N*)
needs a doc string

GSASIIindex.**ranAbyR**(*Bravais*, *A*, *k*, *N*, *ranFunc*)
needs a doc string

GSASIIindex.**ranAbyV**(*Bravais*, *dmin*, *dmax*, *V*)
needs a doc string

GSASIIindex.**ranaxis**(*dmin*, *dmax*)
needs a doc string

GSASIIindex.**rancell**(*Bravais*, *dmin*, *dmax*)
needs a doc string

GSASIIindex.**refinePeaks**(*peaks*, *ibrav*, *A*)
needs a doc string

GSASIIindex.**refinePeaksZ**(*peaks*, *wave*, *ibrav*, *A*, *Zero*, *ZeroRef*)
needs a doc string

GSASIIindex.**rotOrthoA**(*A*)
needs a doc string

GSASIIindex.**scaleAbyV**(*A*, *V*)
needs a doc string

GSASIIindex.**sortM20**(*cells*)
needs a doc string

GSASIIindex.**swapMonoA**(*A*)
needs a doc string

CHAPTER

ELEVEN

GSASIIPLOT: PLOTTING ROUTINES

class GSASIIplot.**G2Plot3D**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)
needs a doc string

class GSASIIplot.**G2PlotMpl**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)
needs a doc string

class GSASIIplot.**G2PlotNoteBook**(*parent*, *id=-1*)
create a tabbed window for plotting

Delete(*name*)
delete a tabbed page

OnNotebookKey(*event*)
Called when a keystroke event gets picked up by the notebook window rather the child. This is not expected, but somehow it does sometimes on the Mac and perhaps Linux.

Assume that the page associated with the currently displayed tab has a child, .canvas; give that child the focus and pass it the event.

OnPageChanged(*event*)
respond to someone pressing a tab on the plot window

Rename(*oldName*, *newName*)
rename a tab

add3D(*name=''*)
Add a tabbed page with a 3D plot

addMpl(*name=''*)
Add a tabbed page with a matplotlib plot

addOgl(*name=''*)
Add a tabbed page with an openGL plot

clear()
clear all pages from plot window

class GSASIIplot.**G2PlotOgl**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)
needs a doc string

class GSASIIplot.**GSASIItoolbar**(*plotCanvas*)
needs a doc string

OnHelp(*event*)
needs a doc string

OnKey(*event*)
needs a doc string

GSASIIplot.**PlotCovariance**(*G2frame*, *Data*)
needs a doc string

GSASIIplot.**PlotDeltSig**(*G2frame*, *kind*)
needs a doc string

GSASIIplot.**PlotExposedImage**(*G2frame*, *newPlot=False*, *event=None*)
General access module for 2D image plotting

GSASIIplot.**PlotISFG**(*G2frame*, *newPlot=False*, *type=''*)
PLotting package for PDF analysis; displays I(q), S(q), F(q) and G(r) as single or multiple plots with waterfall and contour plots as options

GSASIIplot.**PlotImage**(*G2frame*, *newPlot=False*, *event=None*, *newImage=True*)
Plot of 2D detector images as contoured plot. Also plot calibration ellipses, masks, etc.

GSASIIplot.**PlotIntegration**(*G2frame*, *newPlot=False*, *event=None*)
Plot of 2D image after image integration with 2-theta and azimuth as coordinates

GSASIIplot.**PlotPatterns**(*G2frame*, *newPlot=False*)
Powder pattern plotting package - displays single or multiple powder patterns as intensity vs 2-theta, q or TOF. Can display multiple patterns as "waterfall plots" or contour plots. Log I plotting available.

GSASIIplot.**PlotPeakWidths**(*G2frame*)
Plotting of instrument broadening terms as function of 2-theta Seen when "Instrument Parameters" chosen from powder pattern data tree

GSASIIplot.**PlotPowderLines**(*G2frame*)
plotting of powder lines (i.e. no powder pattern) as sticks

GSASIIplot.**PlotRama**(*G2frame*, *phaseName*, *Rama*, *RamaName*, *Names=[]*, *PhiPsi=[]*, *Coeff=[]*)
needs a doc string

GSASIIplot.**PlotRigidBody**(*G2frame*, *rbType*, *AtInfo*, *rbData*, *defaults*)
RB plotting package. Can show rigid body structures as balls & sticks

GSASIIplot.**PlotSeq**(*G2frame*, *SeqData*, *SeqSig*, *SeqNames*, *sampleParm*)
needs a doc string

GSASIIplot.**PlotSizeStrainPO**(*G2frame*, *data*, *Start=False*)
Plot 3D mustrain/size/preferred orientation figure. In this instance data is for a phase

GSASIIplot.**PlotSngl**(*self*, *newPlot=False*)
Single crystal structure factor plotting package - displays zone of reflections as rings proportional to F, F**2, etc. as requested

GSASIIplot.**PlotStructure**(*G2frame*, *data*)
Crystal structure plotting package. Can show structures as balls, sticks, lines, thermal motion ellipsoids and polyhedra

GSASIIplot.**PlotTRImage**(*G2frame*, *tax*, *tay*, *taz*, *newPlot=False*)
a test plot routine - not normally used

GSASIIplot.**PlotTexture**(*G2frame*, *data*, *Start=False*)
Pole figure, inverse pole figure, 3D pole distribution and 3D inverse pole distribution plotting. dict generalData contains all phase info needed which is in data

GSASIIplot.**PlotTorsion**(*G2frame*, *phaseName*, *Torsion*, *TorName*, *Names=[]*, *Angles=[]*, *Coeff=[]*)
needs a doc string

GSASIIplot.**PlotXY**(*G2frame*, *XY*, *newPlot=False*, *type=''*)
simple plot of xy data, used for diagnostic purposes

CHAPTER

TWELVE

GSASII POWDER CALCULATION MODULE

GSASIIpwd.**Absorb**(*Geometry*, *MuR*, *Tth*, *Phi=0*, *Psi=0*)
Calculate sample absorption :param str Geometry: one of 'Cylinder','Bragg-Brentano','Tilting Flat Plate in transmission','Fixed flat plate' :param float MuR: absorption coeff * sample thickness/2 or radius :param Tth: 2-theta scattering angle - can be numpy array :param float Phi: flat plate tilt angle - future :param float Psi: flat plate tilt axis - future

GSASIIpwd.**AbsorbDerv**(*Geometry*, *MuR*, *Tth*, *Phi=0*, *Psi=0*)
needs a doc string

GSASIIpwd.**CalcPDF**(*data*, *inst*, *xydata*)
needs a doc string

GSASIIpwd.**Dict2Values**(*parmdict*, *varylist*)
Use before call to leastsq to setup list of values for the parameters in parmdict, as selected by key in varylist

GSASIIpwd.**DoPeakFit**(*FitPgm*, *Peaks*, *Background*, *Limits*, *Inst*, *Inst2*, *data*, *oneCycle=False*, *controls=None*, *dlg=None*)
needs a doc string

GSASIIpwd.**GetAsfMean**(*ElList*, *Sthl2*)
Calculate various scattering factor terms for PDF calcs

Parameters

- **ElList** (*dict*) – element dictionary contains scattering factor coefficients, etc.
- **Sthl2** (*np.array*) – numpy array of sin theta/lambda squared values

Returns mean(f^2), mean(f)^2, mean(compton)

GSASIIpwd.**GetNumDensity**(*ElList*, *Vol*)
needs a doc string

GSASIIpwd.**LorchWeight**(*Q*)
needs a doc string

GSASIIpwd.**Oblique**(*ObCoeff*, *Tth*)
needs a doc string

GSASIIpwd.**Polarization**(*Pola*, *Tth*, *Azm=0.0*)
Calculate angle dependent x-ray polarization correction (not scaled correctly!)

Parameters

- **Pola** – polarization coefficient e.g 1.0 fully polarized, 0.5 unpolarized

- **Azm** – azimuthal angle e.g. 0.0 in plane of polarization
- **Tth** – 2-theta scattering angle - can be numpy array which (if either) of these is "right"?

Returns (pola, dpdPola) * pola = ((1-Pola)*npcosd(Azm)**2+Pola*npsind(Azm)**2)*npcosd(Tth)**2+ (1-Pola)*npsind(Azm)**2+Pola*npcosd(Azm)**2 * dpdPola: derivative needed for least squares

GSASIIpwd.**Ruland**(*RulCoff*, *wave*, *Q*, *Compton*)
needs a doc string

GSASIIpwd.**SetBackgroundParms**(*Background*)
needs a doc string

GSASIIpwd.**TestData**()
needs a doc string

GSASIIpwd.**Transmission**(*Geometry*, *Abs*, *Diam*)
Calculate sample transmission

Parameters

- **Geometry** (*str*) – one of 'Cylinder','Bragg-Brentano','Tilting flat plate in transmission','Fixed flat plate'
- **Abs** (*float*) – absorption coeff in cm-1
- **Diam** (*float*) – sample thickness/diameter in mm

GSASIIpwd.**Values2Dict**(*parmdict*, *varylist*, *values*)
Use after call to leastsq to update the parameter dictionary with values corresponding to keys in varylist

GSASIIpwd.**calcIncident**(*Iparm*, *xdata*)
needs a doc string

class GSASIIpwd.**cauchy_gen**(*momtype=1*, *a=None*, *b=None*, *xa=-10.0*, *xb=10.0*, *xtol=1e-14*, *badvalue=None*, *name=None*, *longname=None*, *shapes=None*, *extradoc=None*)
needs a doc string

GSASIIpwd.**ellipseSize**(*H*, *Sij*, *GB*)
needs a doc string

GSASIIpwd.**ellipseSizeDerv**(*H*, *Sij*, *GB*)
needs a doc string

GSASIIpwd.**factorize**(*num*)
Provide prime number factors for integer num Returns dictionary of prime factors (keys) & power for each (data)

class GSASIIpwd.**fcjde_gen**(*momtype=1*, *a=None*, *b=None*, *xa=-10.0*, *xb=10.0*, *xtol=1e-14*, *badvalue=None*, *name=None*, *longname=None*, *shapes=None*, *extradoc=None*)
Finger-Cox-Jephcoat D(2phi,2th) function for S/L = H/L Ref: J. Appl. Cryst. (1994) 27, 892-900.

Parameters

- **x** – array -1 to 1
- **t** – 2-theta position of peak
- **s** – sum(S/L,H/L); S: sample height, H: detector opening, L: sample to detector opening distance
- **dx** – 2-theta step size in deg

Returns

for fcj.pdf

- T = x*dx+t
- s = S/L+H/L
- if x < 0:

```
fcj.pdf = [1/sqrt({cos(T)**2/cos(t)**2}-1) - 1/s]/|cos(T)|
```

- if x >= 0: fcj.pdf = 0

GSASIIpwd.**getBackground**(*pfx*, *parmDict*, *bakType*, *xdata*)
needs a doc string

GSASIIpwd.**getBackgroundDerv**(*pfx*, *parmDict*, *bakType*, *xdata*)
needs a doc string

GSASIIpwd.**getEpsVoigt**(*pos*, *alp*, *bet*, *sig*, *gam*, *xdata*)
needs a doc string

GSASIIpwd.**getFCJVoigt**(*pos*, *intens*, *sig*, *gam*, *shl*, *xdata*)
needs a doc string

GSASIIpwd.**getFCJVoigt3**(*pos*, *sig*, *gam*, *shl*, *xdata*)
needs a doc string

GSASIIpwd.**getFWHM**(*pos*, *Inst*)
needs a doc string

GSASIIpwd.**getHKLpeak**(*dmin*, *SGData*, *A*)
needs a doc string

GSASIIpwd.**getPeakProfile**(*dataType*, *parmDict*, *xdata*, *varyList*, *bakType*)
needs a doc string

GSASIIpwd.**getPeakProfileDerv**(*dataType*, *parmDict*, *xdata*, *varyList*, *bakType*)
needs a doc string

GSASIIpwd.**getWidthsCW**(*pos*, *sig*, *gam*, *shl*)
needs a doc string

GSASIIpwd.**getWidthsTOF**(*pos*, *alp*, *bet*, *sig*, *gam*)
needs a doc string

GSASIIpwd.**getdEpsVoigt**(*pos*, *alp*, *bet*, *sig*, *gam*, *xdata*)
needs a doc string

GSASIIpwd.**getdFCJVoigt3**(*pos*, *sig*, *gam*, *shl*, *xdata*)
needs a doc string

GSASIIpwd.**getgamFW**(*g*, *s*)
needs a doc string

GSASIIpwd.**makeFFTsizeList**(*nmin=1*, *nmax=1023*, *thresh=15*)
Provide list of optimal data sizes for FFT calculations

Parameters

- **nmin** (*int*) – minimum data size >= 1
- **nmax** (*int*) – maximum data size > nmin

- **thresh** (*int*) – maximum prime factor allowed

Returns list of data sizes where the maximum prime factor is < thresh

class GSASIIpwd.**norm_gen**(*momtype=1*, *a=None*, *b=None*, *xa=-10.0*, *xb=10.0*, *xtol=1e-14*, *badvalue=None*, *name=None*, *longname=None*, *shapes=None*, *extradoc=None*)

needs a doc string

CHAPTER

THIRTEEN

GSAS-II SCRIPTS

13.1 testDeriv: Check derivative computation

Use this to check derivatives used in structure least squares refinement against numerical values computed in this script.

To use set `DEBUG=True` in GSASIIstrMain.py (line 22, as of version 1110); run the least squares - one cycle is sufficient. Do the "Save Results"; this will write the file testDeriv.dat in the local directory.

Then run this program to see plots of derivatives for all parameters refined in the last least squares. Shown will be numerical derivatives generated over all observations (including penalty terms) and the corresponding analytical ones produced in the least squares. They should match.

`testDeriv.`**`main`**`()`
Starts main application to compute and plot derivatives

13.2 GSASIItestplot: Plotting for testDeriv

Plotting module used for script testDeriv.

class `GSASIItestplot.`**`Plot`**(*parent*, *id=-1*, *dpi=None*, ***kwargs*)
Creates a plotting window

class `GSASIItestplot.`**`PlotNotebook`**(*id=-1*)
creates a Wx application and a plotting notebook

13.3 scanCCD: reduce data from scanning CCD

Quickly prototyped routine for reduction of data from detector described in B.H. Toby, T.J. Madden, M.R. Suchomel, J.D. Baldwin, and R.B. Von Dreele, "A Scanning CCD Detector for Powder Diffraction Measurements". Journal of Applied Crystallography. 46(4): p. 1058-63 (2013).

`scanCCD.`**`main`**`()`
starts main application to merge data from scanning CCD

13.4 *makeMacApp: Create Mac Applet*

This script creates an AppleScript app to launch GSAS-II. The app is created in the directory where the GSAS-II script is located. A softlink to Python is created, but is named GSAS-II, so that GSAS-II shows up as the name of the app rather than Python in the menu bar, etc. Note that this requires finding an app version of Python (expected name .../Resources/Python.app/Contents/MacOS/Python in directory tree of the calling python interpreter).

Run this script with one optional argument, the path to the GSASII.py The script path may be specified relative to the current path or given an absolute path, but will be accessed via an absolute path. If no arguments are supplied, the GSASII.py script is assumed to be in the same directory as this file.

13.5 *unit_tests: Self-test Module*

A script that can be run to test a series of self-tests in GSAS-II. At present, only modules `GSASIIspc` and `GSASIIlattice` have self-tests.

`unit_tests.`**`test_GSASIIlattice`**`()`
: Test registered self-tests in `GSASIIlattice`. Takes no input and returns nothing. Throws an Exception if a test fails.

`unit_tests.`**`test_GSASIIspc`**`()`
: Test registered self-tests in `GSASIIspc`. Takes no input and returns nothing. Throws an Exception if a test fails.

CHAPTER

FOURTEEN

REQUIRED PACKAGES

Note that GSAS-II requires the Python extension packages

- wxPython (http://wxpython.org/docs/api/),
- NumPy (http://docs.scipy.org/doc/numpy/reference/),
- SciPy (http://docs.scipy.org/doc/scipy/reference/),
- matplotlib (http://matplotlib.org/contents.html) and
- PyOpenGL (http://pyopengl.sourceforge.net/documentation)

These are not distributed as part of the Python standard library and must be obtained separately (or in a bundled Python package such as the Enthought Python Distribution/Canopy). The PyOpenGL package is installed into Python by GSAS-II if not found, so it does not need to be included in the Python bundle.

PYTHON MODULE INDEX

INDEX

D

E

F

G

H

I

P

Q

R

S

T

U

V

W