MStack Documentation

Release 0.1

Peter C Metz

Apr 03, 2017

Contents

Peter C. Metz*
Inamori School of Engineering, Alfred University, 2 Pine St., Alfred, NY 14802
**Contact: pcm1@alfred.edu | (315) 350 1585*

CHAPTER 1

Abstract:

In order to refine stacking disorder models in real and reciprocal space, MSTACK has been written to extend two established profile generators: DIFFaX, a reciprocal space intensity distribution calculator built on a stochastic stacking disorder model description; and DiffPy-CMI, a suite of tools including pair distribution function calculators. MSTACK includes tools to expand the stochastic stacking model parameters typical of DIFFaX into supercell models suitable for calculation of stacking disordered pair distribution function data, and to drive refinement of layer structure models from real and reciprocal space data.

MSTACK has been designed with advanced refinement tools in mind. MSTACK is built on the code lmfit which permits the user to include arbitrary constraint equations enabling parametric refinement. Further, MSTACK is designed to be compatible with any minimizer method in the SciPy package, enabling the application of global minimization techniques. Currently implemented minimization methods include the L-BFGS-B non-linear optimization algorithm and the Differential Evolution algorithm. Finally, lmfit and MSTACK enable the user to apply Markov-Chain Monte Carlo analysis, via the package emcee, to the resulting fit. This Bayesian statistical analysis tool has been used predominantly in the astronomical community to interpret data with substantial noise where the error in the data is uncertain. Application of this tool to PDF data suggests many model parameters are not normally distributed- an insight that is expected to have substantial impact on the future of scattering analysis of nanostructured material.

CHAPTER 2

Contents:

background

Created on Thu Mar 30 14:26:43 2017

Background functions for reciprocal-space refinements

@author: Peter C Metz

`mstack.background.`**`inv_x_plus_poly3`**(*x*, *a*, *b*, *c*, *d*, *e*)

    define custom fit function 3rd order polynomial + 1/x term

interface

Created on Tue Apr 26 13:54:44 2016

@author: Peter C Metz

class `mstack.interface.`**`Interface`**(*phases=None*, *mno=None*, *debug=False*)

Bases: `object`

Interface DIFFaX-Py object with DiffPy Structure/Calculator objects

_modules return *return self.attribute.update({stuff})* and are accessed from interface.attribute

`__init__`(*phases=None*, *mno=None*, *debug=False*)

A collection of tools for retreiving values from Phase object(s) in order to construct diffpy.Structure objects and PDF calculations therefrom.

- Phases: (dict | Structure.phase)
- mno: (int, int, int): i.e. (1, 2, 3) a sequence of three integers for supercell expansion

`add`(*attribute*, *value*)

generic method

expand_supercell (*lattice*, *atoms*, *trans*, *dim*, *label*, *debug=False*)
Expand first unit cell (*lattice* + *atoms*) with the transition vectors in *trans* and the dimension *dim*

- lattice: diffpy.Structure.Lattice(a, b, c, alpha, beta, gamma)
- atoms: [diffpy.Stucture.Atom(1), ..., Atom(N)]
- trans: Transition.transition(alpij, rx, ry, rz, cijk)
- dim: intintint (i.e. 123) as dim of supercell
- label: label to be applied to supercell

Returns: diffpy.Structure instance of supercell

is_Uiso (*atom*)
requires atom as Structure.atom as input returns boolean discriminating ADP type

to_cif ()
write supercells to *supercell.label.cif* in cwd

update_phases (*phases*)
phase updater

`mstack.interface.`**is_dict** (*d*)
returns boolean T|F if d passes type check

pairdistributionfunction

Created on Tue Apr 19 13:20:59 2016

Designed to integrate lmfit/scipy differential evolution, existing structure tools, and diffpy PDF generator for global minimization of complex stacking disorered PDF data.

@author: Peter C Metz

`mstack.pairdistributionfunction.`**not_implemented** (*parameter*)
raise exception if

`mstack.pairdistributionfunction.`**name_check** (*iteritem*)
check if names in iteritem are unique

`mstack.pairdistributionfunction.`**load** (*filename*, *subdir=None*)
load a pickled .dat file Note: if all modules needed by the refinement object are not imported at time of

unpickling, there will likely be AttributeErrors thrown.

~! actually, this is problematic. cPickle serializes class by reference, nor definition, so changinging the namespace (e.g. adding or removing modules to this program) will break the pickle.

upgrade to dill which serializes by definition

pairdistributionfunction.PdfData

class `mstack.pairdistributionfunction.`**PdfData** (*name=None*, *data=None*, *qmax=None*, *qmin=None*, *qbroad=None*, *qdamp=None*, *scale=None*, *rmin=None*, *rmax=None*, *rstep=None*, *use=True*)
Bases: *mstack.utilities.UpdateMethods*, *mstack.utilities.MergeParams*

A container for G(r) data which aggregates the miscellaneous necessary values.

`__init__`(*name=None*, *data=None*, *qmax=None*, *qmin=None*, *qbroad=None*, *qdamp=None*, *scale=None*, *rmin=None*, *rmax=None*, *rstep=None*, *use=True*)

Parameters

- **`name`** – as string
- **`data`** – as filepath or (filepath, column) or np.array
- **`qmax`** – as Å ** -1
- **`qmin`** – as Å ** -1
- **`qbroad`** – as Å?? look it up
- **`qdamp`** – as Å?? look it up
- **`scale`** – data scale factor
- **`fit_min`** – minimum r value for refinement as Å
- **`fit_max`** – maximum r value for refinement as Å
- **`sampling`** – sampling interval as float or 'Nyquist'
- **`use`** – boolean flag for refinement

`update_data`(*data*, *column=1*)

add/upate data to data object

Parameters

- **`data`** (`str | np.array`) – file path or data array
- **`column`** (`int`) – if read, read from column

Returns None

pairdistributionfunction.PdfModel

class `mstack.pairdistributionfunction.`**`PdfModel`**(*name=None*, *structure=None*, *scale=None*, *delta1=None*, *delta2=None*, *spdiameter=None*, *sthick=None*, *mno=None*, *use=True*, *sratio=None*, *rcut=None*, *stepcut=None*)

Bases: `mstack.utilities.UpdateMethods`, `mstack.utilities.MergeParams`

A pdf_model is a single structure and the associated envelope parameters. (i.e. scale, Qres, spdiameter, correlated motion delta1|delta2).

In the calculation sequence, model+data attributes spawn calculators by passing config dicts (i.e. PDFcalculator(**cfg)) which act on the contained diffpy.Structure to produce a PDF.

pdf_models are subordinate to pdf_phases which are the corresponding object to reciprocal space objects

`__init__`(*name=None*, *structure=None*, *scale=None*, *delta1=None*, *delta2=None*, *spdiameter=None*, *sthick=None*, *mno=None*, *use=True*, *sratio=None*, *rcut=None*, *stepcut=None*)

Parameters

- **`name`** (`str | None`) – model name
- **`structure`** (`diffpy.Structure | None`) – a supercell generated from interface
- **`scale`** (`float`) – scale factor

- **delta1** (*float*) – [Å] component in the broadening equation below
- **delta2** (*float*) – [Å ** 2] component in the broadening equation below
- **spdiameter** (*float*) – [Å] particle diameter in analytic damping function for spherical nanoparticles.
- **sthick** (*float*) – [Å] sheet thickness in analytic damping function (infinite width)
- **mno** (*int, list*) – [int] (int, int, int) supercell dimensions for expansion of structures
- **use** (*bool*) – include in refinement?
- **sratio** – [-] sigma ratio for bonded atoms- peak sharpening due to correlated motion
- **rcut** – [Å] radius cutoff for application of sratio
- **stepcut** – [Å] distance above which G(r) is truncated

Note:

peak width is given by the Jeong peak width model: σ_ij = σ'_ij * sqrt(1 - δ_1 / r_ij - δ_2 / r^2_ij + Q^2_broad * r^2_ij)

pairdistributionfunction.PdfRefinement

class `mstack.pairdistributionfunction.`**`PdfRefinement`**(*name=None*, *data=None*, *phases=None*)

Bases: *`mstack.utilities.UpdateMethods`*, *`mstack.utilities.MergeParams`*

A pdf refinement is comprised of a structure model(s) and data to be fit against

Contains:

- structure_exchange- translate Structure object into diffpy.Structure.Structure object
- generator- subprocess call to diffpy.srreal.pdfcalculator
- residual_method- ojective function for minimization
- callback- tasks to be evaluated at each fit call
- lsq_minimize- least squares minimization wrapper for lmfit
- diffev_minimize- differential evolution minimization wrapper for lmfit

`__init__`(*name=None*, *data=None*, *phases=None*)

Parameters

- **name** (*str*) – PdfRefinement name
- **data** (`pairdistributionfunction.PdfData`*(s)*) – listlinstance of PdfData
- **phases** (*pairdistributionfunction.PdfPhase(s)*) – listlinstance of PdfPhase

`apply_sheetcf`(*gr*, *sthick*)

Apply sheet cf to a 2xN numpy array containing G(r) in angstroms

Parameters

- **gr** (*np.array*) – G(r) data array
- **sthick** (*float*) – sheet thickness

Returns G(r) data array

Return type np.array

apply_sphericalcf (*gr*, *psize*)

apply spherical cf to a Nx2 shape numpy array containing G(r)

Parameters

- **gr** (*np.array*) – G(r) data array
- **psize** (*float*) – spherical partical diameter

Returns G(r) data array

Return type np.array

calculator (*model*, *data*)

real-space PDF calculation via srreal.pdfcalculator. Calculator built from attributes of model and data.

Parameters

- **model** (PdfModel) – single instance (structure, name=None, scale=None, delta1=None, delta2=None, spdiameter=None, sratio=None, rcut=None, stepcut=None, mno=None)
- **data** (PdfData) – single instance (data, name=None, qmax=None, qmin=None, qbroad=None, qdamp=None, scale=None, rmin=None, rmax=None, rstep=None, use=True)

Returns Calculated (np.array())g(r) scaled by model_scale only

Note: Final difference should be calculated from **sum_i**{(*data_scale* * data_i) - **sum_j**{*phase_scale_j* * **sum_k**{g(r)_k}}}

Note: No shape function applied to calculator result.

callback (*params*, *iter*, *resid*, **args*, ***kwargs*)

Add residual point to dynamic plot, model history

Parameters

- **params** (*lmfit.Parameters*) –
- **iter** (*int*) – iteration number
- **resid** (*array*) – residual array
- **kws** (*dict*) – mostly ignored. use "plot_resid"(bool) to initiate dynamic plot of residual vs. iteration

Returns None

Note: Return type is important in this case. I believe a return type of True causes the minimization to abort.

diffev_minimize (*subdir=None*, *plot_resid=False*, *sqrt_filter=False*, *disp=True*, *popsize=5*, *tol=0.1*, *mutation=(0.4, 0.8)*, *recombination=0.8*, *seed=None*, *polish=False*)

Wrapper for lmfit differential_evolution method (global minimization).

Parameters

- **subdir** (*str*) – directory to stash output files
- **plot_resid** (*bool*) – plot residual vs. iteration
- **sqrt_filter** (*bool*) – plot data scaled by (Yobs) ** 1/2
- **disp** (*bool*) – I forget
- **popsize** (*int*) – see below
- **tol** (*float*) – see below
- **mutation** (*tuple*) – see below
- **recombination** (*float*) – see below
- **seed** (*lmfit.Parameter?*) – see below
- **polish** (*bool*) – follow DIFFEV opt by least squares

Returns

- np.array([sqrt(yo) - sqrt(yc)]) if sqrt_filter is True
- np.array([yo-yc]) if sqrt_filter is False

see scipy.optimize.differential_evolution for compete list of minimizer keys & descriptions

Notes

Differential evolution is a stochastic population based method that is useful for global optimization problems. At each pass through the population the algorithm mutates each candidate solution by mixing with other candidate solutions to create a trial candidate. There are several strategies [R141] for creating trial candidates, which suit some problems more than others. The 'best1bin' strategy is a good starting point for many systems. In this strategy two members of the population are randomly chosen. Their difference is used to mutate the best member (the best in best1bin), b0, so far:

b' = b0 + mutation (population[rand0] population[rand1])

A trial vector is then constructed. Starting with a randomly chosen 'i'th parameter the trial is sequentially filled (in modulo) with parameters from b' or the original candidate. The choice of whether to use b' or the original candidate is made with a binomial distribution (the 'bin' in 'best1bin') - a random number in [0, 1) is generated. If this number is less than the recombination constant then the parameter is loaded from b', otherwise it is loaded from the original candidate. The final parameter is always loaded from b'. Once the trial candidate is built its fitness is assessed. If the trial is better than the original candidate then it takes its place. If it is also better than the best overall candidate it also replaces that. To improve your chances of finding a global minimum use higher popsize values, with higher mutation and (dithering), but lower recombination values. This has the effect of widening the search radius, but slowing convergence.

[R140]: Storn, R and Price, K, "Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization* 11, 341 - 359 (1997).

dim_check (*calc_data*, *exp_data*)

Returns len(calc_data) == len(exp_data)

Return type bool

filter_report (*variable=True*, *constrained=False*)
print a limited portion of the lmfit minimizer fit report ~! moved to utilities

generic_update (*params*)
generic update method passes parameters to subordinate objects

Parameters **`params`** (*`lmfit.Parameters`*) –

Returns True

Return type bool

`lsq_minimize`(*subdir=None*, *plot_resid=False*, *epsfcn=None*, *xtol=None*, *sqrt_filter=False*, *method='leastsq'*, *minkws=None*)

Wrapper for lmfit least_squares method (Levenberg-Marquardt)

Parameters

- **`subdir`** (*`str`*) – directory to put the DIFFaX input/output into.
- **`plot_resid`** (*`bool`*) – toggle dynamic plotting of R vs. iter.
- **`epsfcn`** (*`float`*) – (default = 1e-02) if step-length is too small the mininimizer may not progress as no Jacobian is calculated.
- **`xtol`** (*`float`*) – (default = 1e-04) convergence criterion for the approximate solution.
- **`method`** (*`str`*) – (default = leastsq) optimizer method (i.e. leastsq, nelder, lbfgsb)

Returns np.array([(yo-yc)])

Note: See the SciPy minimizer documentation for full list of minimizer methods.

`map_exp_calc`(*exp_data*, *calc_data*, *rmin*, *rmax*)

map exp data, calc data onto same array dim and stride

Parameters

- **`exp_data`** (*`list array`*) – experimental data
- **`calc_data`** (*`list array`*) – calculated data
- **`rmin`** (*`float`*) – min real space radius
- **`rmax`** (*`float`*) – max real space radius

Returns exp_data, (np.array)calc_data

Return type np.array

`merge_add`(*A1*, *A2*)

Add A1 and A2, merging length to longest vector if unequal

Parameters **`A2`** (*`A1,`*) –

Returns np.array

`model_composite`(*phase*, *data*)

Populate dict of model components for (PdfPhase)phase and (PdfData)data: Model components: self.gr: {data_1: {phase_1: {model_1: g(r)_1, ...}, ...}, ...}

Parameters

- **`phase`** (*`pairdistributionfunction.PdfPhase`*) –
- **`data`** (*`pairdistributionfunction.PdfPhase`*) –

Returns phase_scale * sum_i{gr_i} | dtype=float `objective_function` (*params*, ***kwargs*)

> **Note:** individual residuals aren't returned for each data set. I'm not sure how to introduce a weighting scheme yet (all data equally weighted).

Parameters

- **params** (*lmfit.Parameters*) –
- **kwargs** – see below

kwargs: -subdir: subdirectory -plot_resid: real-time residual plotting (pass thru to callback)

Returns: **sum_i**{residual_i} for i in phases

`phase_composite` (*phases*, *data*, *recalc=True*)

Get sum of scaled model gr for data. We need to wade through three levels to get to comprable patterns:

- Model components: self.gr: {data_1: {phase_1: {model_1: g(r)_1, ...}, ...}, ...}
- Phase components: self.GR: {data_1: {phase_1: G(r)_1, ...}, ...}
- Phase composite: self.composite: {data_1: G(r)_1, ...}

Parameters

- **phases** (*dict*) – PdfPhases
- **data** (pairdistributionfunction.PdfData) – pdf data
- **recalc** (*bool*) – not implemented

> **Note:**
>
> - phases –> dict({phase.name: phase}) (plural)
> - data –> (PdfData) (singular)
> - shape function applied to each model_composite G(r)
> - recalc(default=True) passed in | determined at objective function

Returns: None

`plot_min_result` (*data*, *fontsize=12*)

Plot the calculated, observed, background and difference curves of the last computation. Executed at end of every minimization.

Parameters

- **sqrt_filter** (*bool*) – plot data scaled by (Yobs) ** 1/2
- **fontsize** (*float*) – font size

Returns matplotlib.Figure

`report_refined` (*tabulate=True*)

report parameters with attribute vary == True ~! moved to utilities

reset()
use self.original to reset refined parameters to previous values

residual_method(*data*)
For each phase in refinement, get DIFFaX pattern and calculate residual

Parameters

- **params** (*lmfit.Parameters*) –
- **kws** – see below

kws: subdir: subdirectory plot_resid: real-time residual plotting (pass thru to callback) sqrt_filter: sounds silly, actually just compare sqrt intensities

Returns residual with length of data

Return type np.array

revert()
use self.backup to revert Parameters instance to last minimizer call

rwp()
calculate rwp for the refinement (utilities method) .. note:: ~! not suitable for multiple data

save(*filename=None*, *subdir=None*)
Create a pickled save state of the refinement.

Parameters

- **filename** (*str*) – filename.pkl or some such
- **subdir** (*str*) – directory

Returns None

update_data(*data*)
update data dictionary

Parameters **data** (*list?*) –

update_phases(*phases*)
update structure dict

Parameters **phases** (*pairdistributionfunction.PdfPhase*) – layer phases

Returns None

validate_diffev()
Differential evolution requires min/max values to be supplied for all variables, not just those that are refined.

This function coerces min/max values from supplied information if none are given by the user.

Returns True

refinement

Created on Thu Dec 03 09:24:07 2015

Designed to integrate lmfit/scipy differential evolution, existing structure tools, and DIFFaX I(Q) generator for global minimization of complex stacking disorered powder diffraction data

@author: Peter C Metz

`mstack.refinement.load`(*filename*, *subdir=None*)
load a pickled .dat file

Parameters

- **filename** (*str*) – file to load
- **subdir** (*str | None*) – directory

Note: !!!!!!! EXTREMELY IMPORTANT !!!!!!!!! cPickle saves dependencies by reference. If the source code changes between execution, save state, and loading, the save state WILL NOT LOAD. THIS WILL MAKE YOU VERY SAD.

The next step is to switch from pickle to dill, which saves dependencies by definition. This should make save files compatible across development.

If all modules needed by the refinement object are note imported at time of unpickling, there will likely be AttributeErrors thrown.

refinement.Refinement

class `mstack.refinement.Refinement`(*wavelength=None*, *exp_data=None*, *t_range=None*, *broadening=None*, *background=None*, *phases=None*, *weights=None*, *global_scale=None*, *lateral_broadening=None*, *phase_params=None*, *name=None*)
Bases: *mstack.utilities.MergeParams*, *mstack.utilities.UpdateMethods*

hierarchy of refinement objects:

- refinement: contains experiment (data, parameters) + phase(s) + weights (normalized to 100%)
- phase: described by a structure + transitions
- transitions: holds stacking disorder parameters
- structure: holds asymmetric unit and cell parameters
- atoms: holds coordinates and isotropic thermal displacement parameters

Note: Specific ref_to_phase and phase_to_ref methods are depricated by the UpdateMethods in the utilities module.

I haven't tested the code since replacing the depricated method here.

If this is problematic replace refinement_to_phase and phase_to_refinement methods before __init__ and uncomment the appropriate lines (indicated with in line comments).

__init__(*wavelength=None*, *exp_data=None*, *t_range=None*, *broadening=None*, *background=None*, *phases=None*, *weights=None*, *global_scale=None*, *lateral_broadening=None*, *phase_params=None*, *name=None*)

Parameters

- **wavelength** (*) – experimental radiation in angstrom
- **exp_data** (*) – array like [(x1, y1), ..., (xn, yn)]
- **t_range** (*) – 2-theta range like [2T_min, 2T_max, 2T_step]

- **broadening** (*) – [gau] gaussian FWHM or [u, v, w, sigma] pseudo-voight parameters
- **background** (*) – list of coefficients to yb = A/x + B + C*x + D*x**2 + E*x**2
- **phases** (*) – list of phase instance(s) like [<phase_1>, ... <phase_N>]
- **weights** (*) – dictionary of weight percents like {phase_1.name: weight_1, ..., phase_N.name, weight_N}
- **global_scale** (*) – global scale factor (float)
- **lateral_broadening** (*) – lateral dimension in Angstroms, per DIFFaX Manual (float)
- **phase_params** (*) – dict of {'phase_name': <lmfit.Parameters>}
- **name** (*) – a string to identify the refinement instance

callback (*params*, *iter*, *resid*, ***kws*)

Add residual point to dynamic plot, model history

Parameters

- **params** (*lmfit.Parameters*) –
- **iter** (*int*) – iteration number
- **resid** (*array*) – residual array
- **kws** (*dict*) – mostly ignored. use "plot_resid"(bool) to initiate dynamic plot of residual vs. iteration

Returns None

Note: Return type is important in this case. I believe a return type of True causes the minimization to abort.

diffev_minimize (*subdir=None*, *plot_resid=False*, *sqrt_filter=False*, *disp=True*, *popsize=5*, *tol=0.1*, *mutation=(0.4, 0.8)*, *recombination=0.8*, *seed=None*, *polish=False*)

Wrapper for lmfit differential_evolution method (global minimization).

Parameters

- **subdir** (*str*) – directory to stash output files
- **plot_resid** (*bool*) – plot residual vs. iteration
- **sqrt_filter** (*bool*) – plot data scaled by (Yobs) ** 1/2
- **disp** (*bool*) – I forget
- **popsize** (*int*) – see below
- **tol** (*float*) – see below
- **mutation** (*tuple*) – see below
- **recombination** (*float*) – see below
- **seed** (*lmfit.Parameter?*) – see below
- **polish** (*bool*) – follow DIFFEV opt by least squares

Returns

- np.array([sqrt(yo) - sqrt(yc)]) if sqrt_filter is True

- np.array([yo-yc]) if sqrt_filter is False

see scipy.optimize.differential_evolution for compete list of minimizer keys & descriptions

Notes

Differential evolution is a stochastic population based method that is useful for global optimization problems. At each pass through the population the algorithm mutates each candidate solution by mixing with other candidate solutions to create a trial candidate. There are several strategies [R141] for creating trial candidates, which suit some problems more than others. The 'best1bin' strategy is a good starting point for many systems. In this strategy two members of the population are randomly chosen. Their difference is used to mutate the best member (the best in best1bin), b0, so far:

b' = b0 + mutation (population[rand0] population[rand1])

A trial vector is then constructed. Starting with a randomly chosen 'i'th parameter the trial is sequentially filled (in modulo) with parameters from b' or the original candidate. The choice of whether to use b' or the original candidate is made with a binomial distribution (the 'bin' in 'best1bin') - a random number in [0, 1) is generated. If this number is less than the recombination constant then the parameter is loaded from b', otherwise it is loaded from the original candidate. The final parameter is always loaded from b'. Once the trial candidate is built its fitness is assessed. If the trial is better than the original candidate then it takes its place. If it is also better than the best overall candidate it also replaces that. To improve your chances of finding a global minimum use higher popsize values, with higher mutation and (dithering), but lower recombination values. This has the effect of widening the search radius, but slowing convergence.

[R140]: Storn, R and Price, K, "Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization* 11, 341 - 359 (1997).

filter_report(*variable=True*, *constrained=False*)
print a limited portion of the lmfit minimizer fit report ~! moved to utilities

flag(*true=None*, *false=None*)
Toggle elements of each list True|False respectively

Parameters

- **true** (`list`) – parameter name strings
- **false** (`list`) – parameter name strings

Returns None

generic_update(*params*)
generic update method passes parameters to subordinate objects

Parameters **params** (`lmfit.Parameters`) –

Returns True

Return type bool

lsq_minimize(*subdir=None*, *plot_resid=False*, *epsfcn=None*, *xtol=None*, *sqrt_filter=False*, *method='leastsq'*, *minkws=None*)
Wrapper for lmfit least_squares method (Levenberg-Marquardt)

Parameters

- **subdir** (`str`) – directory to put the DIFFaX input/output into.
- **plot_resid** (`bool`) – toggle dynamic plotting of R vs. iter.
- **epsfcn** (`float`) – (default = 1e-02) if step-length is too small the mininimizer may not progress as no Jacobian is calculated.

- **xtol** (*float*) – (default = 1e-04) convergence criterion for the approximate solution.
- **method** (*str*) – (default = leastsq) optimizer method (i.e. leastsq, nelder, lbfgsb)

Returns np.array([(yo-yc)])

Note: See the SciPy minimizer documentation for full list of minimizer methods.

map_calc_exp_background(*calc_data*)
Map calc, exp, background data onto same array dim and stride

Parameters **calc_data** (*list*) – [(x1, y1), ...,]

Returns [(x1, y1), ...]

Return type list

plot_min_result(*sqrt_filter=False*, *fontsize=12*)
Plot the calculated, observed, background and difference curves of the last computation. Executed at end of every minimization.

Parameters

- **sqrt_filter** (*bool*) – plot data scaled by (Yobs) ** 1/2
- **fontsize** (*float*) – font size

Returns matplotlib.Figure

preview(*subdir=None*, *sqrt_filter=False*)
get peak at first calculated state

pub_control(*subdir=None*, *path='/media/sf_Dropbox/Thesis/MStack/mstack_0.1/doc'*)
Publish control file for all structures in self.phases Control.dif written in working directory Path as os.path.join(*[k for k in [subdir, phase] if k is not None])

Parameters

- **subdir** (*str*) – directory in which to write
- **path** (*str*) – directory in which to write

Returns None

pub_input(*subdir=None*, *path='/media/sf_Dropbox/Thesis/MStack/mstack_0.1/doc'*)
Raises method of phase to refinement level Passes dictionary of ancillary information (info) to phase.pub_input to maintain backwards compatibility with DIFFEV/old input methods Default behavior is to publish control file for all structures in self.phase Path as os.path.join(*[k for k in [subdir, phase] if k is not None])

Parameters

- **subdir** (*str*) – directory in which to write
- **path** (*str*) – directory in which to write

Returns None

report_constrained(*tabulate=False*)
report parameters with attribute expr != None

report_refined(*tabulate=False*)
report parameters with attribute vary == True ~! moved to utilities

reset()
use self.original to reset refined parameters to previous values

residual_method(*params*, ***kws*)
For each phase in refinement, get DIFFaX pattern and calculate residual

Parameters

- **params** (*lmfit.Parameters*) –
- **kws** – see below

kws: subdir: subdirectory plot_resid: real-time residual plotting (pass thru to callback) sqrt_filter: sounds silly, actually just compare sqrt intensities

Returns residual with length of data

Return type np.array

revert()
use self.backup to revert Parameters instance to last minimizer call

rwp(*weight=None*)

calculate rwp for the model: Rwp = {sum_m(w_m * (Yo,m - Yc,m) ** 2) / sum_m(wm * Yo,m) ** 2} ** 1/2 wm = 1 / sigma ** 2

weight (length == data) defalut weight: (Yo,m ** 1/2) ** -2

save(*filename=None*, *subdir=None*)
Create a pickled save state of the refinement.

Parameters

- **filename** (*str*) – filename.pkl or some such
- **subdir** (*str*) – directory

Returns None

update_background(*background_coefficients=None*, *params=None*)
update background from list of coefficients or parameters instances assumes a functional form ybg = A/x + B + C * x + D * x **2 + E * x ** 3

Parameters

- **background_coefficients** (*list | None*) –
- **params** (*lmfit.Parameters | None*) –

Returns None

update_broadening(*broadening*)
update empirical instrumental broadening parameters from list gaussian broadening: [FWHM] length 1 argument pseudo-voight: [u, v, w, sigma] length 4 argument

Parameters **broadening** (*list*) –

Returns None

update_phase(*phases*)
add phases to refinement.Phase(s) dict

update_phase_params(*phase_params*)
updatelinitialize phase_params

Parameters **`phase_params`** (*`dict`*) – {'phase_name': <lmfit.Parameters>}

Returns None

`update_theta_range` (*theta_range*)
Update the refined data range

Parameters **`theta_range`** (*`list`*) – [min, max, stride] in units of 2 theta

Returns None

`update_weights` (*weights*)
Update weights.

Parameters **`weights`** (*`dict`*) – {phase name: weight}

Note: weights are automatically normalized to 1

Returns None

`validate_diffev`()
Differential evolution requires min/max values to be supplied for all variables, not just those that are refined.

This function coerces min/max values from supplied information if none are given by the user.

Returns True

`weighted_composite` (*subdir=None*, *individual=False*, *column=2*, *path='/media/sf_Dropbox/Thesis/MStack/mstack_0.1/doc'*)
Return composite of patterns generated by phases & associated weighting factors. looks for phase_name.spc in pathsubdir

Args:

Returns all weighted components individual is False (list | default): [(x1, y1), ...]

Return type individual is True (dict)

structure

A structure contains atoms (x ,y, z, occ, type, ADP), and a structure contains a unit cell (a, b, c, α, β, γ) and asymmetric unit (list of atoms).

A phase contains layer structures as well as transitions to propagate the layer motif in along the perpendicular vector. The stacking vector is always taken as parallel to **c**. Created on Fri Oct 16 16:37:48 2015

A structure is a essentially an enhanced dictionary that contains lattice parameters and atom instances.

@author: Peter C Metz

`mstack.structure.`**`build_cif`** (*filename*, *structure_name=None*, *layer_number=1*, *path=None*)

Use PyCifRW to parse a .cif file and output a Structure instance.

Args:

- filename (str): pathfilname.extension
- structure_name (str): name for Structure intsance

- layer_number (int): layer number
- path (str|None): directory

•currently required site labels as "TypeNumber"; i.e. Mn1, O5000, Uuo195

Note: There appears to be an error with parsing a filename containing a complete path. i.e. C:Path1Path2...

ilename.cif is interpreted as a URL for some dumb reason. The current use of ReadCif simply strips the mount point of your disk drive, which seems to work so long as the current working directory is on the same disk as your .cif file.

structure.Atom

class `mstack.structure.`**`Atom`**(*atom_name*, *number*, *x*, *y*, *z*, *Uiso_or_equiv*, *occ*, *disp_type='Uiso'*)

Bases: `object`

an atom instance contains its type (name) and number (of its kind) which define a unique label (i.e. O1, Nb5). x, y, & z are float fractional coordinates and Uiso is the thermal displacement parameter (Biso = 8*pi**2*<u**2> = 8*pi**2*Uiso)

Parameters

- **`atom_name`** (*) – atom name
- **`number`** (*) – site number
- **`x, y, z`** (*) – fractional coordinates
- **`Uiso_or_equiv`** (*) – isotropic thermal displacement parameter
- **`occ`** (*) – occupancy fractional
- **`disp_type`** (*) – 'Biso' or 'Uiso'

`__init__`(*atom_name*, *number*, *x*, *y*, *z*, *Uiso_or_equiv*, *occ*, *disp_type='Uiso'*)

Parameters

- **`atom_name`** (*) – atom name
- **`number`** (*) – site number
- **`x, y, z`** (*) – fractional coordinates
- **`Uiso_or_equiv`** (*) – isotropic thermal displacement parameter
- **`occ`** (*) – occupancy fractional
- **`disp_type`** (*) – 'Biso' or 'Uiso'

structure.Structure

class `mstack.structure.`**`Structure`**(*name*, *a*, *b*, *c*, *alp=90*, *bet=90*, *gam=90*, *atoms=None*, *number=1*)

Bases: *`mstack.utilities.UpdateMethods`*, *`mstack.utilities.MergeParams`*

A structure instance contains the lattice and asymmetric unit, presently limited only to P1 symmetry, of a layer structure.

`__init__`(*name*, *a*, *b*, *c*, *alp=90*, *bet=90*, *gam=90*, *atoms=None*, *number=1*)

Parameters

- **a** (*) – lattice parameter a [Å]
- **b** (*) – lattice parameter b [Å]
- **c** (*) – lattice parameter c [Å]
- **alp** (*) – lattice parameter alpha [°] *default = 90*
- **bet** (*) – lattice parameter beta [°] *default = 90*
- **gam** (*) – lattice parameter gamma [°] *default = 90*
- **atoms** (*) – list of atom instances *default = None*
- **number** (*) – layer number [integer] used to index layer when building transition matrix

Note: In DIFFaX alpha and beta are constrained to 90°- only gamma may vary. These are presently included as a formality.

get_all_par()

returns dictionary of all paramters stored in structure, e.g.:

- lattice parameters (vector magnitudes and angles)
- atom parameters (x, y, z, occ, Uiso)

Returns {[atom.label]_[par]: value, ...}

Return type dict

get_atom_par(*par*)
return atom par(str) with keys label_par (dict)

lattice_angles()
return lattice angles as variables (dict)

lattice_params()
return lattice parameters as variables (dict)

merge_adp(*atoms*)
get list of atom ADPS in common ADP type (Biso)

Parameters **atoms** (*) – list of atom instances

parameters(**valid_keys*)
Creates a lmfit Parameters instance from the contents of structure.

All pars come fixed by default- user will flag enteries for refinment using the Parameters method instance['parname'].vary=Boolean.

If a list of keywords is passed, parameters() will attempt to assemble a Parameters instance using the valid keys and reporting the invalid keys (i.e. ['a', 'b', 'c', 'Ox']... returned params.keys()= 'a', 'b', 'c'): invalid key 'ox'.)

Parameters **valid_keys** (`str, list`) – parameter keys

Returns instance containing *valid_keys

Return type lmfit.Parameters

structure.Phase

class `mstack.structure.`**`Phase`**(*name*, *transitions=None*, *structures=None*, *parameters=None*, *redchi=None*, *broadening=None*, *mcl=None*, *path='/media/sf_Dropbox/Thesis/MStack/mstack_0.1/doc'*)

Bases: `mstack.utilities.MergeParams`, `mstack.utilities.UpdateMethods`

A Phase contains the layer structure and transition(s) to be expanded into a refinable supercell model.

Multiple Phase objects can be fed to the Refinement object to accout for polytypism or multiphase data, etc.

`__init__`(*name*, *transitions=None*, *structures=None*, *parameters=None*, *redchi=None*, *broadening=None*, *mcl=None*, *path='/media/sf_Dropbox/Thesis/MStack/mstack_0.1/doc'*)

Parameters

- **`transitions- transitions instance`** (*) –
- **`structures- list of structure instances`** (*) –
- **`parameters- lmfit parameters instance`** (*) –
- **`redchi - reduced chi value of last iteration`** (*) –
- **`broadening -[FWHM] |[u, v, w, sigma]`** (*) –
- **`mcl - mean column length`** (*) –
- **`path`** (*) – directory path

`initialize_structure_params`(*stru*)

initialize structure parameters as lmfit parameters for stru(s)

Parameters **`stru`** (*) –

`pub_control`(*info*, *inputname='diffax_in.dat'*, *path='/media/sf_Dropbox/Thesis/MStack/mstack_0.1/doc'*, *subdir=None*)

write control file for single file inputname.dat supply T_min, **T_max_** T_step as info ~! this will change at some point

Parameters

- **`info`** (*) – theta information (min, max, step)
- **`inputname`** (*) – .dat file for DIFFaX to munch crunch upon
- **`path`** (*) – directory
- **`subdir`** (*) – more directory info... probably scrap this

Returns

- None

`pub_input`(*info*, *inputname='diffax_in'*, *path='/media/sf_Dropbox/Thesis/MStack/mstack_0.1/doc'*, *subdir=None*)

Distill phase into diffax input file inputname.dat.

At the moment, ancillary information is not stored in model- i.e. radiation type, 2-theta limits, etc... needs to be passed in needed (key):

•wavelength (wvl)

•broadening parameters (gau) ~! only gaussian implimented right now!

•lateral braodening (lat) – this is optional

•Mean Column Length (MCL)

Parameters

- **info** (*) – indicated above
- **inputname** (*) – fname

Returns

- None

report_refined()
returns dict of items with self.params[item].vary == True

toggle(*flags=None*)
set refinement_flags to refine

flags (str): parameter names to togggle vary - True

update_mcl(*mcl*)
Create mean column length and MCL/1022 variabels. DIFFaX interprets mcl >= 1022 as infinite crystallites, hence the normalization.

Parameters **mcl** (*) – 1 <= mean column length <= 1022

update_structures(*stru*)
add/update structure to/in model

Parameters **stru** (*) –

update_transitions(*T*)
add/update transition to/in model

Parameters **T** (`transition.Transitions`) –

supercell

Created on Wed Mar 02 11:34:35 2016

quick script to translate Structure instances into supercells -copy/shift asymmetric unit according to single vector -accept user dimension (N-units along c-vector) or default to MCL -dump .xyz file and/or .cif (P1) file for visualization and supercell PDF

*** pub_cif**

*** pub_xyz**

*** supercell**

@author: Peter C. Metz

`mstack.supercell.`**pub_cif**(*a, b, c, gam, asym, path, filename*)
publish a structure in .cif format.

Parameters

- **b, c, gam** (*a,*) – lattice parameters
- **asym** (*list*) – list of structure.Atoms
- **path** (*str*) – directory of file.cif
- **filename** (*str*) – filname for dump (omit .cif)

`mstack.supercell.`**`pub_xyz`**(*a*, *b*, *c*, *gam*, *asym*, *path*, *filename*)

write .xyz as <N atoms> <comment line> < atom> <x> <y> <z>

Parameters

- **`b, c, gam`** (*`a,`*) – lattice parameters
- **`asym`** (*`list`*) – list of structure.Atoms
- **`path`** (*`str`*) – directory of file.xyz
- **`filename`** (*`str`*) – filname for dump (omit .xyz)

@!!!!!! Orthogonal vector space conversion is broken

`mstack.supercell.`**`supercell`**(*struct*, *vector*, *N=None*, *cif=True*, *xyz=False*, *path=None*, *filename=None*, *debug=False*)

Dump a supercell model for input structure and vector with dim 1x1xN.

Parameters

- **`struct`** (`structure.Structure`) – layer structure
- **`vector`** (*`list, dict`*) – layer vector in fractional values
- **`N`** (*`bool | None`*) – supercell dimension
- **`cif`** (*`bool| True`*) – output cif?
- **`xyz`** (*`bool | False`*) – output xyz?
- **`path`** (*`str|None`*) – directory
- **`filename`** (*`str|None`*) – filename
- **`debug`** (*`bool|False`*) – return expanded asymmetric unit

Returns None

transition

transition.Transition

class `mstack.transition.`**`Transition`**(*i*, *j*, *alpij=None*, *vector=None*, *cij=None*)

Bases: `object`

instantiate a transition to create empty variables that satisfy the program requirements of DIFFaX. Parameters include transition index n, transition probability alphaij, vector R, and uncertainty cij

`* generic`

`* scale_cij`

`* update_cij`

`* update_alpij`

`* update_vector`

`__init__`(*i*, *j*, *alpij=None*, *vector=None*, *cij=None*)

Announce components of the nth transition. Defaults = 0.0.

Parameters

- **alpij** (*dict*) – {'alpn1': float, ..., 'alpnN': float} dim(n_layers)
- **vector** (*dict*) – {'rx': float, 'ry': float 'rz': float}
- **cij** (*dict*) – {'c11': float, 'c22': float, 'c33': float, 'c12': float, 'c13': float, 'c23': float}

Note: Pass additional information as tuple, e.g. (value [, True|False [, min, max]])

generic (*generic*, *name*, *min0=0.0*, *max0=1.0*)
Method for updating params.

Parameters

- **generic** (*dict*) – {key: (value [, True|False [, min, max]]), ...} optional vary boolean, min, max as positional args
- **name** (*str*) – alpij|cij|vector

Returns sets parameter and updates appropriate attribute

scale_cij (*force=False*)
scale cij by smaller corresponding cii to satisfy requirement that cij <= cii

Parameters **force** (*bool*) – executes scaling whether or not self.scaled is True

Returns True

Return type bool

update_alpij (*alpij*)
initialize/update transition probabilities (dim(n))

update_cij (*cij*)
initialize/update cij (fractions) and scaled_cij (appropriately scaled copy)

update_vector (*vector*)
initialize/update vector

transition.Transitions

class mstack.transition.**Transitions** (*nlayers=None*, *transitions=None*)
Bases: object

Container for all the transitions specifying a stacking disorder problem.

- **pub_trans**
  method to publish transitions block in DIFFaX format. I.e.: {alpij Rxj Ryj Rzj (clm)} 1.0000000000 0.3550 0.3550 1.0000 (10.0000 10.0000 2.0000 0.0000 0.0000 0.0000) {1-1} ...

  {N-N}
- **row_normal**
- **todict**
- **update_transitions**
- **validate_transitions**

__init__ (*nlayers=None*, *transitions=None*)

Parameters

- **transitions** (*list*) – list containing transition instances
- **nlayers** (*int*) – number of layer types (N)

pub_trans()
publish the transition information in DIFFaX suitable format, e.g. alpij Rx Ry Rz (Cijk) {i-j}

Returns list of transitions in DIFFaX format (string lists) (0th element always empty)

row_normal(*row=0*)
row normalize entries in alpij

todict()
map numpy array as dict

update_transitions(*transitions*)
initialize/update dictionary of transitions

validate_transitions(*force=False*)

Note: Because empty fields are initialized with appropriate values except for probabilities, we really just need to confirm the user supplied probabilities and that they:

1.are row normalized.

2.of uniform dimension (N x N)

These conditions are enforced (transitions are operated on) if not correct

Parameters **force** (*bool | False*) – recompute scaled cij whether or not trans.scaled is True

Returns True|False

Return type Boolean

utilities

Created on Thu Dec 03 13:28:46 2015

Common utility classes and functions for MStack.

@author: Peter C Metz

class `mstack.utilities.`**DynamicPlot**(*fontsize=14*)
Bases: `object`

Plotting utility used to create output for itterative function. Called in minimizer callback function to create Rwp vs. iter Reserves plot number 100 for this purpose.

call signiture: DynamicPlot(xdata, ydata) –> appended point to plot

__init__(*fontsize=14*)

on_launch()
set up plot

on_running(*new_x, new_y*)
update plot

class `mstack.utilities.`**`MergeParams`**

Bases: `object`

Tools to merge Parmeters instances between objects containing them while maintaining unique parameter names.

The result is an lmfit.Parameters object on the top class with the name 'params' (so call your lmfit.Parameters instance params if you want this to work smoothly) Although the specification of lmfit.Parameters attributes as other names works with specifier

`add_set_params`(*name=None*, *value=None*, *vary=None*, *min=None*, *max=None*, *expr=None*)

add/set parameter in refinement parameters

for list of supported mathematics, see: http://lmfit.github.io/lmfit-py/constraints.html#supported-operators-functions-and-constants

Parameters

- **name** (*str*) – parameter name
- **value** (*float*) – parameter value
- **vary** (*bool*) – vary in refinement?
- **min** (*float*) – minimum bound
- **max** (*float*) – maximum bound
- **expr** (*str*) – constrain expression.

Returns None

`exists`(*attribute*, *value=None*)

check if attribute exists in object | create with value(None) else

`lower_to_upper`(*top_attribute*, *specifier=None*)

When merging lmfit.Parameters instances belonging to different constituent refinement objects, we run into an issue of unique variable naming (x occurs for each atom coodinate, i.e.)

The transmogrifier appends the top_attribute.name to the bottom_attribute.Parameter.name attribute to construct a unique variable label. This change is propagated to variables in the instance's constraint expression to maintain validity.

i.e top_attribute = (attribute as str) indicating dictionary of subordinate objects bottom_attribute = params instance subordinate object

`param_finder`(*bottom_attribute*, *specifier*)

get parameter instance from subordinate object

`upper_to_lower`(*top_attribute*, *specifier=None*, *debug=False*)

•**top_attribute: attribute name for dict of subordinate objects** i.e. 'phases' –> refinement.phases = {'phase_1': <PairDistributionFunction.PdfPhase>}

•specifier: name of parameters instance in subordinate object

class `mstack.utilities.`**`UpdateMethods`**

Bases: `object`

Generic update methods for the data types dealt with in pdf refinement objects

•initialize: set attribute for class if it doesn't exist, add Parameter instance

•update: update an initialized parameter with appropriate method

•update_with_limits: update when value received as (value, min, max)

•update_with_lmfit: update when value received as lmfit.Parameter instance

initialize(*attribute*, *value=None*)
default variable initialization

update(*attribute*, *value=None*)
default update mode

update_with_limits(*attribute*, *tup*)
allow args passed as (value, min, max)

update_with_lmfit(*attribute*, *parameter*)
update self.params with Parameter instance

mstack.utilities.**attributegetter**(*items*)
Return a callable object that retrieves named *attributes* from its operand using the objects __getattribute__() method. This is analagous to the built-in operator *operator.itemgetter*.

Parameters **items** (*str, list*) – attribute names

Returns callable object that retreives attribute values from the operand

mstack.utilities.**checkequal**(*iterator*)
Check if subsequent iterables are equivilant used only in Structure.pub_input

Parameters **iterator** (*iterable*) –

Returns bool

mstack.utilities.**filter_report**(*refinement*, *variable=True*, *constrained=False*, *_print=True*, *_text=False*)
print a limited portion of the lmfit minimizer fit report.

Parameters

- **refinement** (*Refinement instance*) – Pdf or I(Q) Refinement instance
- **variable** (*bool|True*) – report refined variables
- **constrained** (*bool|False*) – report constrained variables
- **_print** (*bool|True*) – print output
- **_text** (*bool|False*) – list output

Returns list of lines of output text

Return type list

mstack.utilities.**flatten**(*iterable*)
flatten list of lists with N-recursion depth

mstack.utilities.**interpolate_data**(*Array1*, *Array2*, **mesh*)
Map Array1 onto Array2 if Array 2 specified Else, map Array1 onto user defined mesh(float)

Parameters

- **Array1** (*list, np.array*) – (x, y) data
- **Array2** (*list, np.array*) – (x, y) data
- **mesh** (*float*) – stride for interpolation if Array2 absent

Returns [(x1, y1), ..., (xn, yn)]

Return type list

`mstack.utilities.`**`isfinite`**`(`*value*`)`
test if value is infinite

`mstack.utilities.`**`not_header`**`(`*line*, *override=False*`)`
Check line in input file for # or text Override used for debugging only

Returns True if data, False if header

Return type bool

`mstack.utilities.`**`plot`**`(`**Array*, ***kwargs*`)`
Takes a list of tuples [(x1,y1),...,(xn,yn)] and plots with line format

~! bug: axis determined on last loaded plot (could lead to truncation)

Parameters

- **`Array`** (*`list, np.array`*) – array(s) with shape (N,2)
- **`kwargs`** – accepts xmin, xmax, ymin, ymax as key word args

Returns matplotlib plot object

`mstack.utilities.`**`print_table`**`(`*dictionary=None*, *table=None*, *key=None*, *headers=None*`)`
pretty print wrapper of tabulate.

Parameters

- **`dictionary`** (*`dict | None`*) – dictionary of table content
- **`table`** (*`list | None`*) – list format of table
- **`key`** (*`sort key | None`*) – last operation is list.sort(key=key)
- **`headers`** (*`list | None`*) – list of headers for columns ['string', ...'Nstring']

Returns prints table, returns True if no exception raised

Return type bool

`mstack.utilities.`**`read_data`**`(`*filename*, *column=1*, *lam=None*, *q=False*, *override=True*`)`
Reads data from space delimited format. Default assumption is that (x, y) are in the first and second column, respectively. Use column argument to change elsewise. use argument 'q' if data is a function of scattering vector rather than 2theta.

Parameters

- **`filename`** – [str] path/filename.extension
- **`column`** – [int] location of f(x), x being the 0th column
- **`lam`** – [float] wavelength of experimental radiation
- **`q`** – [bool] whether data is a function of scattering vector q
- **`override`** – [bool] skip stripping header operation if output is unacceptable

Returns (x,y) array of data like [(x1, y1), ..., (xn, yn)]

Return type list

`mstack.utilities.`**`report_refined`**`(`*minimizer_results_object_params*, *tabulate=False*`)`
report values of parameters object with attribute vary=True

Parameters

- **`result`** (*`lmfit.result`*) – fit result object
- **`tabulate`** (*`bool | False`*) – print or return table

Returns if tabulate is False print: if tabulate is True

Return type dict

`mstack.utilities.`**`rwp`**`(`*PDF_refinement*, *weight=None*`)`

returns the pattern weighted residual for a single data set refinement e.g.

(sum(weight * diff ** 2) / sum(weight * ref.yo ** 2)) ** 0.5

Parameters

- **`PDF_refinement`** – [PdfRefinement instance]
- **`weight`** – [np.array] with same shape as observed data vector Yo

Returns Rwp value

Return type float

utilities.DynamicPlot

class `mstack.utilities.`**`DynamicPlot`**`(`*fontsize=14*`)`

Bases: `object`

Plotting utility used to create output for itterative function. Called in minimizer callback function to create Rwp vs. iter Reserves plot number 100 for this purpose.

call signiture: DynamicPlot(xdata, ydata) –> appended point to plot

`__init__``(`*fontsize=14*`)`

`on_launch``()`

set up plot

`on_running``(`*new_x*, *new_y*`)`

update plot

utilities.MergeParams

class `mstack.utilities.`**`DynamicPlot`**`(`*fontsize=14*`)`

Bases: `object`

Plotting utility used to create output for itterative function. Called in minimizer callback function to create Rwp vs. iter Reserves plot number 100 for this purpose.

call signiture: DynamicPlot(xdata, ydata) –> appended point to plot

`__init__``(`*fontsize=14*`)`

`on_launch``()`

set up plot

`on_running``(`*new_x*, *new_y*`)`

update plot

utilities.UpdateMethods

class `mstack.utilities.`**`DynamicPlot`**`(`*fontsize=14*`)`

Bases: `object`

Plotting utility used to create output for itterative function. Called in minimizer callback function to create Rwp vs. iter Reserves plot number 100 for this purpose.

call signiture: DynamicPlot(xdata, ydata) –> appended point to plot

`__init__`(*fontsize=14*)

`on_launch`()
set up plot

`on_running`(*new_x*, *new_y*)
update plot

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index