PyPhy (python + physics/math)
Home
Description of Displacement Parameters for Creating Custom Presets
Examples for using overlays
Overlays
Overlays for Backward Settings
Backward
Backward Settings
API Reference
pyphy.utilities
pyphy.dip
pyphy.dipset
pyphy.gamma
pyphy.gammaset
pyphy.utilities.cmd
pyphy.utilities.io
pyphy.utilities.math
pyphy.utilities.plot
pyphy.utilities.stats
pyphy.utilities.tex
pyphy.utilities.time
pyphy.utilities.tqdm
pyphy.utilities.tqdm.tqdm
pyphy.utilities.tqdm.tqdm_pandas
pyphy.utilities.tqdm.trange
pyphy.utilities.tqdm.tqdm_notebook
pyphy.utilities.tqdm.tqdm_pandas_notebook
pyphy.utilities.tqdm.trange_notebook
pyphy.utilities.tqdm.tqdm_gui
pyphy.utilities.tqdm.trange_gui
pyphy.utilities.tqdm.tqdm_notebook_gui
pyphy.utilities.tqdm.trange_notebook_gui
pyphy.utils

PyPhy Parameters for Creating Custom Presets

PyPhy (python + physics/math) is a collection of tools for creating custom presets for the Gamma Spectrum Generator (GSG). This documentation contains information on the parameters used in the GSG. This dictionary is typically named `gamma_dict`.

Some parameters are used in the `pyphy.gammaset` and `pyphy.dipset` modules, which helps generate `gamma_dict` and `dip_dict` respectively. Some parameters are used in the GSG. Some complicated or advanced parameters, this guide explains how to construct your own `gamma_dict` and `dip_dict`. Dictionary manually.

Lambda Type Parameters in PyPhy

Without some knowledge of lambda functions, it's important to understand a special type of input PyPhy supports **lambda type parameters**.

Because PyPhy is factoring complicated parameters, it is important to understand the required input. To do this, in the GSG, lambda type parameters are that allow a parameter to accept a **numpy array/numpy scalar**, **lambda function**, **lambda function with numpy array/numpy scalar**, **dictionaries/lists and combinations of the underlying types**.

For example, to enable a parameter starting at epoch 10 and keep it some offset/shift

```
lambda epoch: epoch - 10
```

This pattern is ideal for having an input accepting an offset/shift/scalar/scaling parameter after an initialization period (because the parameter is constant).

Full list of parameters for `pyphy.iterative.run()`

Backward Settings

Parameter	Default Value	Default Value Type	Description
`backend`	`'dip'`	str, `'dip'`, `'dipset'`, `'gamma'`, `'gammaset'`	Currently not used but will be in future as the Fission, Flight, Fluo simulation will be available. It is currently for PyPhy backends. It is mainly a backend for `pyphy.iterative.run()` function. Currently, `'dip'` and `'dipset'` are the only available options. Future options will be `'gamma'` and `'gammaset'` for the Gamma Spectrum Generator.
`dipfunc_dtype`	`'uint32'`	str	Default dtype type for `dipfunction`. Smaller option is `'uint16'`.

Dataset

Parameter	Default Value	Default Value Type	Description
`dataset`	`None`	str	Path to the dataset `.h5` file. On the GSG, a dataset is required. Refer to the documentation for more information.
`dataset_keys`	`None`	str, list of str, `None`	Keys in the dataset to use. Default `None` means all keys.
`dataset_keep_in_memory`	`False`	bool	If set, loads entire dataset into RAM. Can speed up I/O but uses more memory. Recommended for small datasets.
`dataset_cache_size`	`0`	int	Number of chunks to cache in memory. Only relevant for chunked HDF5 datasets. Set to 0 to disable caching.
`dataset_cache_nslots`	`1`	int	Number of hash table slots for chunk cache. Should be prime for best performance. Only relevant for chunked HDF5 datasets.
`dataset_rdcc_w0`	`0.75`	float, `None`	Preemption policy for chunk cache (0.0 = always evict, 1.0 = never evict). Only relevant for chunked HDF5 datasets.
`dataset_dtype`	`None`	str, `None`	Force all data to this dtype (e.g., `'float32'`, `'uint16'`). Saves memory if you don't need full precision.
`dataset_transform`	`None`	callable, `None`	Function applied to each batch after loading. Signature: `(batch_dict) -> batch_dict`. Use for normalization, augmentation, etc.
`dataset_cast_to_float32`	`True`	bool	Automatically cast all arrays to float32. Set to False if you want to keep original dtypes.
`dataset_pin_memory`	`False`	bool	Pin memory for faster GPU transfer. Only useful if using GPU and batches are large enough to benefit from pinned memory.
`dataset_num_workers`	`0`	int	Number of worker processes for data loading. Set to 0 for single-process loading. Multi-worker loading can speed up I/O but uses more memory.
`dataset_prefetch_factor`	`None`	int, `None`	Number of batches to prefetch per worker. Only relevant when num_workers > 0. Higher values use more memory but can improve throughput.
`dataset_persistent_workers`	`False`	bool	Keep worker processes alive between epochs. Avoids overhead of spawning workers each epoch. Only relevant when num_workers > 0.
`dataset_pin_device`	`None`	str, `None`	Device to pin memory to (e.g., `'cuda:0'`). Only relevant if pin_memory is True.
`dataset_drop_last`	`False`	bool	Drop the last incomplete batch if dataset size is not divisible by batch size. Useful for training with batch normalization.
`dataset_shuffle`	`False`	bool	Shuffle dataset before each epoch. Set to False for deterministic ordering.
`dataset_seed`	`None`	int, `None`	Random seed for shuffling. Set for reproducible data ordering.
`dataset_sampler`	`None`	str, `None`	Sampler type: `'random'`, `'sequential'`, `'weighted'`, `'subset_random'`, `'distributed'`, `'custom'`. Overrides shuffle if specified.
`dataset_sampler_weights`	`None`	str, `None`	Dataset key for sample weights (for weighted sampler). Only relevant when sampler = `'weighted'`.
`dataset_sampler_num_samples`	`None`	int, `None`	Number of samples per epoch (for random/weighted sampler). Defaults to dataset size.
`dataset_sampler_replacement`	`False`	bool	Sample with replacement (for random/weighted sampler). Allows duplicate samples in an epoch.
`dataset_custom_sampler`	`None`	str, `None`	Name of custom sampler registered with `register_sampler()`. Only used when sampler = `'custom'`.
`dataset_custom_sampler_kwargs`	`None`	str, `None`	JSON string of keyword arguments for custom sampler. Example: `'{"alpha": 0.5}'`. Only used when sampler = `'custom'`.
`dataset_distributed`	`False`	bool	Enable distributed sampling across multiple GPUs/processes. Only relevant in multi-GPU training.
`dataset_distributed_rank`	`None`	int, `None`	Rank of this process in distributed training. Only relevant when distributed = True.
`dataset_distributed_world_size`	`None`	int, `None`	Total number of processes in distributed training. Only relevant when distributed = True.

Saving and Printing

Parameter	Default Value	Default Value Type	Description
`print_every`	`None`	int, `None`	Print every this many steps. Prints all by default.
`save_every`	`None`	int, `None`	Save every this many steps. Saves only at end by default.
`save_dir`	`'.'`	str	Directory to save checkpoints. Supports relative and absolute paths.
`save_prefix`	`None`	str, `None`	Prefix for checkpoint filenames. Defaults to `'checkpoint'` if not specified.
`save_keep_last`	`None`	int, `None`	Keep only the last N checkpoints. Older checkpoints are deleted. Set to `None` to keep all checkpoints.
`save_keep_every`	`None`	int, `None`	Keep every Nth checkpoint regardless of age. Useful for keeping milestones while cleaning up others.
`save_compressed`	`False`	bool	Use compressed .npz format instead of .pickle. Saves disk space but slower.
`save_dipfunc`	`False`	bool	Save dipfunction arrays. Set to False to save only metadata and parameters.
`save_dipfunc_only_last`	`False`	bool	Save only the last dipfunction. Overrides save_dipfunc.
`save_dipfunc_dtype`	`None`	str, `None`	Force dipfunction to this dtype when saving (e.g., `'float32'`). Saves disk space.
`save_dipfunc_compressed`	`False`	bool	Use compressed .npz for dipfunction. Slower but smaller files.
`save_dipfunc_chunk_size`	`None`	int, `None`	Chunk size for streaming dipfunction to disk. Enables memory-efficient saving for large arrays.
`save_dipfunc_streaming`	`False`	bool	Stream dipfunction to disk during computation. Requires save_dipfunc_chunk_size.
`save_dipfunc_streaming_keep_in_memory`	`False`	bool	Keep dipfunction in memory while also streaming to disk. Uses more memory but faster.
`save_dipfunc_streaming_compressed`	`False`	bool	Use compressed format for streaming dipfunction. Slower but smaller files.
`save_dipfunc_streaming_flush_every`	`None`	int, `None`	Flush streaming data every N chunks. Controls disk write frequency.
`save_dipfunc_streaming_dtype`	`None`	str, `None`	Force streaming dipfunction to this dtype. Saves disk space.
`save_dipfunc_streaming_only_last`	`False`	bool	Streaming only the last dipfunction. Overrides other streaming options.
`save_dipfunc_streaming_only_last_dtype`	`None`	str, `None`	Dtype for streaming only last dipfunction. Overrides save_dipfunc_streaming_dtype.
`save_dipfunc_streaming_only_last_compressed`	`False`	bool	Compressed streaming only last dipfunction. Overrides save_dipfunc_streaming_compressed.
`save_dipfunc_streaming_only_last_chunk_size`	`None`	int, `None`	Chunk size for streaming only last dipfunction. Overrides save_dipfunc_streaming_chunk_size.
`save_dipfunc_streaming_only_last_keep_in_memory`	`False`	bool	Keep only last dipfunction in memory while streaming. Overrides save_dipfunc_streaming_keep_in_memory.
`save_dipfunc_streaming_only_last_flush_every`	`None`	int, `None`	Flush only last dipfunction every N chunks. Overrides save_dipfunc_streaming_flush_every.
`save_dipfunc_streaming_only_last_dtype`	`None`	str, `None`	Dtype for only last dipfunction. Overrides save_dipfunc_streaming_only_last_dtype.
`save_dipfunc_streaming_only_last_compressed`	`False`	bool	Compressed only last dipfunction. Overrides save_dipfunc_streaming_only_last_compressed.
`save_dipfunc_streaming_only_last_chunk_size`	`None`	int, `None`	Chunk size for only last dipfunction. Overrides save_dipfunc_streaming_only_last_chunk_size.
`save_dipfunc_streaming_only_last_keep_in_memory`	`False`	bool	Keep only last dipfunction in memory while streaming. Overrides save_dipfunc_streaming_only_last_keep_in_memory.
`save_dipfunc_streaming_only_last_flush_every`	`None`	int, `None`	Flush only last dipfunction every N chunks. Overrides save_dipfunc_streaming_only_last

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline

API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

Breakdown of Examples for Iterative Ptychography

This section provides a detailed breakdown of an iterative ptychography reconstruction using PyPty. To get started, create a Python script and import the necessary libraries:

```
import numpy as np
import pypty
# NumPy is used for minor preprocessing, while all major operations are handled by
print("PyPty version: ", pypty._version_)
```

We will need numpy to do some small preprocessing outside of pypty, but the major steps will be done with pypty functions.

Normal 4D-STEM iterative ptychography.

In this section I will go through the steps required to do a ptychographic reconstruction from normal 4D-STEM data (far-field).

Creation of Calibrated Data

In a first step we want to create and center a 4D-STEM dataset (in h5 format). PyPty can also accept raw numpy-arrays, but it's better to do this preprocessing step.

```
path = path_to_your_data
name = name of your dataset
path_raw = path+name+".npy"
path_h5  = path+name+".h5"

pypty.initialize.create_pypty_data(path_raw, path_h5, swap_axes=False,
                    flip_ky=0,flip_kx=0, comcalc_len=200*200,
                    comx=0, comy=0, bin=1, crop_left=None,
                    crop_right=None, crop_top=None, crop_bottom=None, normalize=Fa
```

Experimental parameters

PyPty accepts .json metadata from Nion microscopes, but you can also specify all entries yourself. Here I create a dictionary `experimental_params` with relevant parameters. See the experiment description for further details.

```
path_json="" ## I will leave this string like this and specify everything by hand.
output_folder=your_output_folder

experimental_params={
    'output_folder': output_folder,
    'path_json': path_json,
    'data_path': path_h5,

    ### You can specify one of the following three parameters
    'rez_pixel_size_A': None,
    'rez_pixel_size_mrad': None,
    'conv_semiangle_mrad': 33.8,


    'scan_size': [256,256], ## this is number of scan points along slow and fast a

    ###  You can specify one of the following two parameters
    'scan_step_A': 0.18, # scan step in Angstrom
    'fov_nm': None,

    'acc_voltage': 60, ## kV

    'aberrations': [200,0,0],## Angstrom  [C10, C12a, C12b], can be longer

    'PLRotation_deg': 4.65, # deg


    'bright_threshold': 0.2, ## this parameter will be used to estimate the apertu

    'total_thickness': 6, ## Angstrom, thickness of the sample
    'num_slices': 1,  ## number of slices in your reconstruction

    'plot': True, ## pypty will plot a few things
    'print_flag': 3, ## max verbosity
    'data_pad': None, ## None will add 1/4 of pixels to each side. This is done to

    ## now there are 2 nexus tags you can specify. This will help to identify your
    'chemical_formula': YOUR_Formula,
    'sample_name': NAME_of_YOUR_Sample,

    }
```

Loading a reconstruction preset

Parameters for iterative ptychography are stored in a dictionary typically named `pypty_params`.

1) Such presets are typically stored as .pkl data and you can load them via a function in `utils` module

```
path_to_your_pkl_preset=
pypty_params=pypty.utils.load_preset(path_to_your_pkl_preset)
```

2) Alternatively, you can load them from a nexus data of another reconstruction

```
path_to_your_nxs_preset=
pypty_params=pypty.utils.load_nexus_params(path_to_your_nxs_preset)
```

3) A third option is to construct your custom preset as a dictionary from scratch (Please see the guide)

```
pypty_params={
    "your_entry": something,
    ...
    }
```

4) Finally, you can specify it as `None`, then you will have a standard preset

```
pypty_params=None
```

Joining a preset with experimental parameters

Once you have a preset, you can add experimental data to it.

```
pypty_params=pypty.initialize.append_exp_params(experimental_params, pypty_params)
```

Starting Ptychography

This is done with a single function and single argument, `pypty_params`:

```
pypty.iterative.run(pypty_params)
```

Saving your Results as one Nexus file

Once your reconstruction is done, you will find a bunch of files in the `output_folder`. You can stack them into one big nexus file via a single function. You can also select where the .nxs file will be saved via `path_to_your_nexus_file`

```
pypty.utils.convert_to_nxs(output_folder, path_to_your_nexus_file)
```

Reconstruction without an extra h5 file

If you do not want to create an extra .h5 file (**not recommended**), you can specify the 4D-STEM data directly in the experimental parameters (before you join them with `pypty_params`).

```
experimental_params["dataset"]=dataset
```

Note than in this case the entry `data_path` will be ignored and you can leave this field empty. The dataset itself can be either 4D or 3D (with 2 scan axes merged into one).

Iterative Ptychography from Virtual Detectors

If you want to perform a reconstruction from data compressed by virtual detectors, you first have to ensure that the array of detectors has shape `[N_detectors, k_Y, k_X]` and your data has shape `[Scan_Y, Scan_X, N_detectors]`. Then you attach these two arrays to experimental parameters.

```
experimental_params={
    "dataset": compressed_data,
    "masks": virtual_detector_array,


    ###  In this case you must specify one of the following two parameters
    'rez_pixel_size_A': None,
    'rez_pixel_size_mrad': 1,

   # ... Rest of your parameters.
}
```

Also, you may want to update your reconstruction settings. For Compressed data PyPty has following objective function types:

Objective_name	Description
`lsq_compressed`	Least-squared fit
`gauss_compressed`	Gaussian noise model
`poisson_compressed`	Poissonian noise model

Then you must specify `algorithm` as one of these three options in `pypty_params`:

```
pypty_params={
        'algorithm':  gauss_compressed,
        #... Rest of your parameters
        }
```

Then you can launch ptychography with the same pypty.iterative.run function.

Table of contents
Normal 4D-STEM iterative ptychography.
Creation of Calibrated Data
Experimental parameters
Loading a reconstruction preset
Joining a preset with experimental parameters
Starting Ptychography
Saving your Results as one Nexus file
Reconstruction without an extra h5 file
Iterative Ptychography from Virtual Detectors

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline

API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

Examples for direct methods

This page presents example usage of PyPty's direct reconstruction methods: Differential Phase Contrast (DPC), Tilt-Corrected Bright Field (tcBF), and Wigner Distribution Deconvolution (WDD). These methods are used when a direct (non-iterative) reconstruction is preferred or as preprocessing for iterative methods.

Differential Phase Contrast (DPC)

PyPty is supplied with two functions for differential phase contrast. Both of them aim so solve a Poisson equation, i.e. to reconstruct a phase from its laplacian, gradient of the COM vector field. The first method, based on the Fast Fourier Transform (FFT), reconstructs the phase efficiently and can be executed as follows (including visualization using matplotlib)

```
dpc_phase, pypty_params= pypty.dpc.fft_based_dpc(pypty_params, hpass=1e-1, lpass=0

plt.imshow(dpc_phase, cmap="gray")
plt.title("DPC Phase (FFT-based)")
plt.colorbar()
plt.show()
```

Note that fft-based DPC returns both phase and parameter dictionary as it stores the calculated COM vector field (and the angle between real and reciprocal spaces). If these two quantities are not in `pypty_params`, it attempts to automatically calculate them.

Second method is iterative and accounts for non-periodic boundary conditions:

```
dpc_phase=pypty.dpc.iterative_dpc(pypty_params, num_iterations=100, beta=0.5, hpas
plt.imshow(dpc_phase, cmap="gray")
plt.title("DPC Phase (iterative)")
plt.colorbar()
plt.show()
```

Please see the module description for detailed description of the parameters.

Tilt-Corrected Bright Field (tcBF)

In PyPty framework tcBF is done in two steps. First one aims to fit the aberrations of the beam and rotation angle between real and reciprocal spaces by creating an **unupsampled** tcBF image via `run_tcbf_alignment`:

```
pypty_params=pypty.tcbf.run_tcbf_alignment(
        pypty_params,
        binning_for_fit=np.tile([5], 30),
        optimize_angle=True,

        cross_corr_type="abs",
        refine_box_dim=5,
        upsample=10,
        cancel_large_shifts=0.9,
        reference_type="bf",
        tol_ctf=1e-8
        )
```

Note that you have to play with binning values and cross-correlation parameters to get an optimal result. Also, fitted rotation angle typically has a +- pi ambiguity, so if your defocus value comes out with a sign different from the experiment, I suggest to rerun the reconstruction.

A second tcBF function assumes that you already did an alignment and performs an upsampled reconstruction:

```
tcbf_image,tcbf_px_size = pypty.tcbf.upsampled_tcbf(pypty_params, upsample=3,
                                    pad=10, default_float=32,round_shifts=True)

plt.imshow(tcbf_image,  cmap="gray")
plt.title("tcBF image (x3 upsampling)")
plt.colorbar()
plt.show()
```

Wigner Distribution Deconvolution (WDD)

The last direct reconstruction method is Wigner distribution deconvolution (WDD). It is a direct method that reconstructs the object's complex wavefunction by deconvolving the measured intensity distribution with the known probe. WDD requires a calibrated `pypty_params` dictionary containing rotation angle, scan steps and a complex beam (or a set of aberrations). A simple usage example is

```
obj_wdd=pypty.direct.wdd(pypty_params,  eps_wiener=1e-3)

plt.imshow(np.angle(obj_wdd), cmap="gray")
plt.title("WDD Phase")
plt.colorbar()
plt.show()
```

The parameter `eps_wiener` serves as a regularization term (high-pass filter) to stabilize the deconvolution, preventing division by zero.

Using Direct Methods for Initial Guesses

pypty.initialize module is supplied with a function `get_ptycho_obj_from_scan` that can be used to create complex transmission function based on a measurement sampled at the scan points, this can be a DPC or WDD phase. Usage example is:

```
pypty_params =  pypty.initialize.get_ptycho_obj_from_scan(pypty_params, array_phas
```

If you have an array on a grid with finer sampling than the actual scan grid of your reconstruction, you may first create an upsampled grid in the image coordinates and use the array for initial guess like this:

```
image = ## your upsampled image
image_pixel_size = ## pixel size of your image
left_zero_of_scan_grid = # how many pixels of the image are from left of the scan
top_zero_of_scan_grid  = # how many pixels of the image are from top of the scan
scan_array_A=pypty.initialize.get_grid_for_upsampled_image(pypty_params, image, im

pypty_params =  pypty.initialize.get_ptycho_obj_from_scan(pypty_params, array_phas
```

You can also manually specify the initial guess by providing keys like `"obj"`, `"probe"` or `"positions"`, for more see Guide for creating custom presets.

These direct methods are essential tools in PyPty for rapid insight and as a starting point for more advanced reconstructions. You can combine them with iterative pipelines by injecting results into the initialization phase.

Table of contents
Differential Phase Contrast (DPC)
Tilt-Corrected Bright Field (tcBF)
Wigner Distribution Deconvolution (WDD)
Using Direct Methods for Initial Guesses

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline
API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

Getting started with the `initialize` Module

PyPty comes with an `initialize` module (see the API Reference), which is designed specifically for **far-field ptychography**.

This module makes it easy to:

- Create a **calibrated dataset**
- Generate a **PyPty preset** from scratch

If your experimental setup involves additional complexity or non-standard conditions, please refer to the Custom Preset Guide for more flexible configuration.

However, if your setup fits the typical **far-field ptychography** case, you can summarize all key experimental parameters in a single Python dictionary. For clarity, we usually refer to this as `experimental_params`.

The dictionary should contain the following keys:

Essential Data Paths

Key	Type	Description
`data_path`	`str`	Path to a **3D PyPty `.h5` dataset** (`[N_measurements, ky, kx]`) or a **4D-STEM `.npy` dataset**.
`masks`	`ndarray` or `None`	If the data is compressed, provide the virtual detectors (`[N_masks, ky, kx]`).
`output_folder`	`str`	Directory where results will be stored.
`path_json`	`str`	Path to a Nion-style `.json` file with metadata (optional).

Electron Beam Properties

Key	Type	Description
`acc_voltage`	`float`	Accelerating voltage (in **kV**).
`rez_pixel_size_A`	`float`	Reciprocal pixel size (in **Å⁻¹**).
`rez_pixel_size_mrad`	`float`	Reciprocal pixel size (in **mrad**).
`conv_semiangle_mrad`	`float`	Beam convergence semi-angle (in **mrad**).
`aperture`	`ndarray` (optional)	Binary **2D mask** representing the aperture.
`bright_threshold`	`float`	Threshold for estimating an aperture. Everything above `threshold * max(PACBED)` is considered bright field.

Scan and Positioning

Key	Type	Description
`scan_size`	`tuple(int, int)`	Number of scan points along **slow (y)** and **fast (x)** axes.
`scan_step_A`	`float`	Scan step (STEM pixel size) in **Å**.
`fov_nm`	`float`	Field of view (**FOV**) along the **fast axis** in **nm**.
`special_positions_A`	`ndarray` (optional)	If data was acquired on a non-rectangular grid, specify positions as `[y_0, x_0], ..., [y_n, x_n]` (in **Å**).
`transform_axis_matrix`	`ndarray (2×2)`	Transformation matrix for position correction.
`PLRotation_deg`	`float` or `"auto"`	Rotation angle between scan and detector axes. If `"auto"`, an iDPC measurement estimates this angle.

Reconstruction Settings

Key	Type	Description
`num_slices`	`int`	Number of slices used for multislice propagation (default: `1`).
`total_thickness`	`float`	Total thickness of the sample (in **Å**).
`data_pad`	`int` or `None`	Reciprocal space padding. Default: `1/4` of pattern width.
`upsample_pattern`	`int`	Upsampling factor for diffraction patterns.
`flip_ky`	`bool`	Flip the y-axis of diffraction patterns.
`defocus`	`float`	Extra probe defocus (besides aberrations).
`aberrations`	`list` or `ndarray`	Beam aberrations (stored in Krivanek notation).

Output & Debugging

Key	Type	Description
`plot`	`bool`	If `True`, generates plots of key experimental parameters.
`print_flag`	`int`	Controls verbosity (`0` = silent, `1` = summary, `2+` = detailed logs).
`save_preprocessing_files`	`bool`	If `True`, saves intermediate preprocessing files.

Nexus Tags

Key	Type	Description
`chemical_fomula`	`string`	You can optionally provide a formula for your sample. PyPty will store this information.
`sample_name`	`string`	You can optionally provide a name of your sample. PyPty will store this information.

Table of contents
Essential Data Paths
Electron Beam Properties
Scan and Positioning
Reconstruction Settings
Output & Debugging
Nexus Tags

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline
API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

Overview

PyPty is a **phase retrieval** library for **near-field** and **far-field imaging** in **TEM/STEM**. It supports:

- **Iterative ptychography**
- **Direct ptychography** (Wigner Distribution Deconvolution)
- **Differential phase contrast (DPC)**
- **Tilt-corrected bright field (tcBF)**
- **Focal series reconstructions**
- **LARBED reconstructions**

Developed by Anton Gladyshev (AG SEM, Physics Department, Humboldt-Universität zu Berlin).

Installation

To create a suitable Python environment for PyPty, you can use **conda**, **mamba**, or **micromamba**. Below are installation instructions using `conda`.

GPU Installation

```
git clone git@github.com:Anton-Gladyshev/pypty.git
cd pypty
conda env create -f pypty_gpu.yml
conda activate pypty
pip install .[gpu]
```

CPU Installation

```
git clone git@github.com:Anton-Gladyshev/pypty.git
cd pypty
conda env create -f pypty_cpu.yml
conda activate pypty
pip install .
```

About this Code

Main Purpose

The primary function of PyPty is **gradient-based phase reconstruction** from far-field intensities. This is handled via `pypty.iterative.run()`. Most other functions are designed to simplify usage of this core function.

A complete list of required arguments for `pypty.iterative.run()` is available in the custom presets guide.

Logic

Most PyPty functions accept a single input dictionary called `pypty_params`, which typically includes both experimental and reconstruction settings. These experimental parameters are often defined separately in a dictionary called `experimental_params` (see experiment description).

Once you've created a valid `pypty_params` for one experiment, you can reuse it for others by simply "attaching" new experimental parameters.

PyPty also supports various direct methods and tools for initialization.

Input Format

PyPty primarily operates on `.h5` files and provides tools to convert from NumPy. The `.h5` file should contain a dataset named `"data"` with a 3D array: one scan axis and two detector axes.

PyPty also accepts 4D Nion-style NumPy arrays (with accompanying JSON metadata).

Output Format

By default, PyPty saves all output as `.npy` files inside an `output_folder`. The folder structure may look like this:

```
output_folder
├── cg.npy                                  **overwritable checkpoint for scan grid**
├── cp.npy                                  **overwritable checkpoint for probe**
├── co.npy                                  **overwritable checkpoint for object**
├── checkpoint_obj_epoch_N.npy              **checkpoint for object at epoch N**
├── checkpoint_probe_epoch_N.npy            **checkpoint for probe at epoch N**
├── checkpoint_positions_epoch_N.npy        **checkpoint for scan grid at epoch N**
├── params.pkl                              **parameter file of your reconstruction**
├── loss.csv                                **.csv log-file**
├── tcbf                                    **folder  with tcBF results**
│   ├── tcbf_image_upsampling_5.npy         **upsampled tcBF image.**
│   ├── tcbf_image_N.png                    **intermediate tcBF image at iteration N i
│   ├── estimated_shifts_N.npy              **Fitted shifts at iteration N in .npy for
│   ├── aberrations_N.npy                   **Fitted aberrations (in Angstrom) at iter
│   ├── PL_angle_deg.npy                    **array containing fitted rotation angles
│   ├── aberrations_A.npy                   **2d array containing fitted aberrations
└── dpc                                     **folder  with DPC results**
    ├── idpc.npy                            **fft-based dpc phase**
    ├── iterative_dpc.npy                   **DPC phase (iterative reconstruction)**
    └── wdd                                 **folder  with WDD results**
        ├── object.npy                      **complex object (WDD ptychography)**
```

You can also convert an output folder into single Nexus `.nxs` file via pypty.utils.convert_to_nxs() function.

Examples

Example workflows are available in the `examples` folder on GitHub.

For a quick start, check out the following:

- Full Jupyter Notebook Example — A complete pipeline example
- Step-by-Step Breakdown — Explains key functions and logic
- Direct Methods — For Wigner distribution and other direct approaches
- Custom Presets Guide — For advanced setups

Relevant Literature

If you have any questions after reading this guide, the following papers, books and links might explain the working principle of the code:

Multi-slice formalism and the NN-style approach

1) Earl J. Kirkland. Advanced Computing in Electron Microscopy

2) W. Van den Broek and C. Koch. General framework for quantitative three-dimensional reconstruction from arbitrary detection geometries in TEM

3) W. Van den Broek and C. Koch. Method for Retrieval of the Three-Dimensional Object Potential by Inversion of Dynamical Electron Scattering

Error metrics:

LSQ:

4) M. Schloz et al. Overcoming information reduced data and experimentally uncertain parameters in ptychography with regularized optimization

5) M. Du et al. Adorym: a multi-platform generic X-ray image reconstruction framework based on automatic differentiation

Maximum Likelihood (ML):

6) P. Thibault and M. Guizar-Sicairos. Maximum-likelihood refinement for coherent diffractive imaging

Compressed LSQ:

7) A. Gladyshev et al. Lossy Compression of Electron Diffraction Patterns for Ptychography via Change of Basis

lsq_sqrt and lsq_sqrt2:

8) P. Godard et al. (2012). Noise models for low counting rate coherent diffraction imaging

Mixed state formalism:

9) P Thibault & A Menzel Reconstructing state mixtures from diffraction measurements— Flux-preserving formalism (for near-field imaging)

Near-field imaging

10) C. Koch A flux-preserving non-linear inline holography reconstruction algorithm for partially coherent electrons

Tilted propagator:

11) Earl J. Kirkland. Advanced Computing in Electron Microscopy

12) H. She, J. Cui and R. Yu. Deep sub-angstrom resolution imaging by electron ptychography with misorientation correction

Regularization constaints:

13) M. Schloz et al. Overcoming information reduced data and experimentally uncertain parameters in ptychography with regularized optimization

14) A. Gladyshev et al. Lossy Compression of Electron Diffraction Patterns for Ptychography via Change of Basis

Linesearch

15) L. Armijo (1966). Minimization of functions having Lipschitz continuous first partial derivatives 16) P. Wolfe (1969). Convergence Conditions for Ascent Methods

BFGS algotithm

17) C. G. Broyden (1970). The convergence of a class of double-rank minimization algorithms

18) R. Fletcher (1970). A New Approach to Variable Metric Algorithms

19) D. Goldfarb (1970). A Family of Variable Metric Updates Derived by Variational Means

20) D. F. Shanno (1970). Conditioning of quasi-Newton methods for function minimization

Complex derivatives

21) W. Wirtinger (1927). Zur formalen theorie der funktionen von mehr komplexen veränderlichen.

Table of contents
Installation
GPU Installation
CPU Installation
About this Code
Main Purpose
Logic
Input Format
Output Format
Examples
Relevant Literature
Multi-slice formalism and the NN-style approach
Error metrics:
LSQ:
Maximum Likelihood (ML):
Compressed LSQ:
lsq_sqrt and lsq_sqrt2:
Mixed state formalism:
Near-field imaging
Tilted propagator:
Regularization constaints:
Linesearch
BFGS algotithm
Complex derivatives

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline

API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

pypty.direct

wdd

Perform Wigner Distribution Deconvolution (WDD).

This function applies Wigner Distribution Deconvolution to the provided data, allowing for enhanced reconstruction of complex objects and probes. It handles both GPU (via CuPy) and CPU (via NumPy) computations based on the availability of CuPy. The `pypty_params` dictionary must be prepared via the `initialize` module.

PARAMETER	DESCRIPTION
`pypty_params`	Dictionary containing calibrated parameters, including paths and settings for data processing. **TYPE:** `dict`
`eps_wiener`	Epsilon parameter for the Wiener filter. Default is 1e-3. **TYPE:** `float` **DEFAULT:** `0.001`
`thresh`	If provided, `eps_wiener` is ignored. Denominator values below this threshold are set to 1, and the corresponding numerator values are set to 0. **TYPE:** `float` **DEFAULT:** `None`
`save`	Whether to save output files (False by default). Ignored if `save_preprocessing_files` is specified in `pypty_params`. **TYPE:** `bool` **DEFAULT:** `0`

RETURNS	DESCRIPTION
`deconvolved_object`	2D deconvolved complex object. **TYPE:** `ndarray`

Notes

Make sure `pypty_params` includes all necessary paths and data arrays prior to calling this function.

Table of contents
direct
wdd

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline
API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

pypty.dpc

get_curl

Compute the standard deviation of the curl of a rotated DPC vector field. This is the objective function for minimization. This particular function was copied from a DPC plugin written by Jordan Hachtel.

PARAMETER	DESCRIPTION
angle	Rotation angle in radians. **TYPE:** float
dpcx	X-component of the DPC signal. **TYPE:** ndarray
dpcy	Y-component of the DPC signal. **TYPE:** ndarray

RETURNS	DESCRIPTION
float	Standard deviation of the curl after rotation.

get_curl_derivative

Compute the derivative of the curl-based objective function with respect to rotation angle.

PARAMETER	DESCRIPTION
angle	Rotation angle in radians. **TYPE:** float
dpcx	X-component of the DPC signal. **TYPE:** ndarray
dpcy	Y-component of the DPC signal. **TYPE:** ndarray

RETURNS	DESCRIPTION
float	Derivative of the curl-based objective function.

GetPLRotation

Estimate rotation angle that minimizes the curl of the DPC signal.

PARAMETER	DESCRIPTION
dpcx	X-component of the DPC signal (2D array). **TYPE:** ndarray
dpcy	Y-component of the DPC signal (2D array). **TYPE:** ndarray

RETURNS	DESCRIPTION
float	Optimal rotation angle in radians.

fft_based_dpc

FFT-based DPC phase reconstruction. If you setted up the pypty_params properly, you would only need to specify the hpass and lpass values, both are non-negative floats.

PARAMETER	DESCRIPTION
pypty_params	PyPty parameter dictionary with dataset and calibration settings. **TYPE:** dict
hpass	High-pass filtering coefficient (default is 0). **TYPE:** float **DEFAULT:** 0
lpass	Low-pass filtering coefficient (default is 0). **TYPE:** float **DEFAULT:** 0
save	Whether to save the reconstructed phase (default is False). **TYPE:** bool **DEFAULT:** False
comx	Precomputed center-of-mass x-component. **TYPE:** ndarray or None **DEFAULT:** None
comy	Precomputed center-of-mass y-component. **TYPE:** ndarray or None **DEFAULT:** None
plot	If True, display the phase reconstruction. **TYPE:** bool **DEFAULT:** False

RETURNS	DESCRIPTION
pot	Reconstructed 2D phase image. **TYPE:** ndarray
pypty_params	Updated parameter dictionary with computed COM and rotation angle. **TYPE:** dict

iterative_dpc

Iterative DPC phase reconstruction. If you setted up the pypty_params properly, you would only need to specify the hpass and lpass values, both are non-negative floats.

PARAMETER	DESCRIPTION
pypty_params	PyPty parameter dictionary. **TYPE:** dict
num_iterations	Number of gradient descent iterations (default is 100). **TYPE:** int **DEFAULT:** 100
beta	Step reduction factor for backtracking (default is 0.5). **TYPE:** float **DEFAULT:** 0.5
hpass	High-pass filtering coefficient (default is 0). **TYPE:** float **DEFAULT:** 0
lpass	Low-pass filtering coefficient (default is 0). **TYPE:** float **DEFAULT:** 0
step_size	Initial gradient descent step size (default is 0.1). **TYPE:** float **DEFAULT:** 0.1
COMx	X-component of COM map. **TYPE:** ndarray or None **DEFAULT:** None
COMy	Y-component of COM map. **TYPE:** ndarray or None **DEFAULT:** None
px_size	Scan step size in Ångströms. **TYPE:** float or None **DEFAULT:** None
print_flag	Whether to print progress information (default is False). **TYPE:** bool **DEFAULT:** False
save	Whether to save the result to disk (default is False). **TYPE:** bool **DEFAULT:** False
select	Optional binary mask to constrain reconstruction. **TYPE:** ndarray or None **DEFAULT:** None
plot	If True, plot the reconstruction result . **TYPE:** bool **DEFAULT:** True
use_backtracking	Whether to use backtracking line search (default is True). **TYPE:** bool **DEFAULT:** True
pad_width	Padding width to suppress FFT boundary artifacts (default is 5). **TYPE:** int **DEFAULT:** 5

RETURNS	DESCRIPTION
padded_phase	Reconstructed 2D phase image. **TYPE:** ndarray

iterative_poisson_solver

Iterative solver for Poisson equation given a Laplacian map.

PARAMETER	DESCRIPTION
laplace	Input 2D array representing the Laplacian of the desired phase. **TYPE:** ndarray
num_iterations	Number of iterations (default is 100). **TYPE:** int **DEFAULT:** 100
beta	Step size reduction factor (default is 0.5). **TYPE:** float **DEFAULT:** 0.5
hpass	High-pass filtering parameter (default is 0). **TYPE:** float **DEFAULT:** 0
lpass	Low-pass filtering parameter (default is 0). **TYPE:** float **DEFAULT:** 0
select	Optional binary mask to restrict updates. **TYPE:** ndarray or None **DEFAULT:** None
px_size	Pixel size in Ångströms (default is 1). **TYPE:** float **DEFAULT:** 1
print_flag	If True, print convergence status (default is False). **TYPE:** bool **DEFAULT:** False
step_size	Initial gradient descent step size (default is 0.1). **TYPE:** float **DEFAULT:** 0.1
use_backtracking	Whether to use backtracking line search (default is True). **TYPE:** bool **DEFAULT:** True
pad_width	Number of pixels to pad around the solution (default is 1). **TYPE:** int **DEFAULT:** 1
xp	Backend array library (NumPy or CuPy, default is NumPy). **TYPE:** module **DEFAULT:** numpy

RETURNS	DESCRIPTION
ndarray	Reconstructed 2D phase from the input Laplacian.

Table of contents
dpc
get_curl
get_curl_derivative
GetPLRotation
fft_based_dpc
iterative_dpc
iterative_poisson_solver

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline
API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

pypty.fft

fftfreq

Wrapper for fftfreq function.

PARAMETER	DESCRIPTION
length	Length of the output array. **TYPE:** int
sampling	Sample spacing (default is 1). **TYPE:** float **DEFAULT:** 1

RETURNS	DESCRIPTION
ndarray	Discrete Fourier Transform sample frequencies.

ifftfreq

Wrapper for ifftfreq function.

PARAMETER	DESCRIPTION
length	Length of the output array. **TYPE:** int
sampling	Sample spacing (default is 1). **TYPE:** float **DEFAULT:** 1

RETURNS	DESCRIPTION
ndarray	Discrete inverse Fourier Transform sample frequencies.

fftshift

Wrapper for fftshift function.

PARAMETER	DESCRIPTION
array	Input array to be shifted. **TYPE:** ndarray

RETURNS	DESCRIPTION
ndarray	Shifted array.

ifftshift

Wrapper for ifftshift function.

PARAMETER	DESCRIPTION
array	Input array to be inverse shifted. **TYPE:** ndarray

RETURNS	DESCRIPTION
ndarray	Inverse shifted array.

shift_fft2

Wrapper for xp.fft.fftshift(xp.fft.fft2(...)) functions.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the FFT (default is (0, 1)). **TYPE:** tuple of int **DEFAULT:** (0, 1)
overwrite_x	If True, allow overwriting the input array (default is False). **TYPE:** bool **DEFAULT:** False

RETURNS	DESCRIPTION
ndarray	2D Fourier Transform of the input array, shifted.

fft2

Wrapper for fft2 function.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the FFT (default is (0, 1)). **TYPE:** tuple of int **DEFAULT:** (0, 1)
overwrite_x	If True, allow overwriting the input array (default is False). **TYPE:** bool **DEFAULT:** False

RETURNS	DESCRIPTION
ndarray	2D Fourier Transform of the input array.

ifft2

Wrapper for ifft2 function.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the inverse FFT (default is (0, 1)). **TYPE:** tuple of int **DEFAULT:** (0, 1)
overwrite_x	If True, allow overwriting the input array (default is False). **TYPE:** bool **DEFAULT:** False

RETURNS	DESCRIPTION
ndarray	2D inverse Fourier Transform of the input array.

ifft2_ishift

Wrapper for xp.fft.ifft2(xp.fft.ifftshift(...)) function.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the inverse FFT (default is (0, 1)). **TYPE:** tuple of int **DEFAULT:** (0, 1)
overwrite_x	If True, allow overwriting the input array (default is False). **TYPE:** bool **DEFAULT:** False

RETURNS	DESCRIPTION
ndarray	2D inverse Fourier Transform of the input array, after shifting.

shift_fftn

Wrapper for xp.fft.fftshift(xp.fft.fftn(...)) function.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the FFT (default is (0, 1, 2)). **TYPE:** tuple of int **DEFAULT:** (0, 1, 2)

RETURNS	DESCRIPTION
ndarray	N-dimensional Fourier Transform of the input array, shifted.

ifftn_ishift

Wrapper for xp.fft.ifftn(xp.fft.ifftshift(...)) function.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the inverse FFT (default is (0, 1, 2)). **TYPE:** tuple of int **DEFAULT:** (0, 1, 2)

RETURNS	DESCRIPTION
ndarray	N-dimensional inverse Fourier Transform of the input array, after shifting.

ifftn

Wrapper for ifftn function.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the inverse FFT (default is (0, 1, 2)). **TYPE:** tuple of int **DEFAULT:** (0, 1, 2)

RETURNS	DESCRIPTION
ndarray	N-dimensional inverse Fourier Transform of the input array.

fftn

Wrapper for fftn function.

PARAMETER	DESCRIPTION
arr	Input array to be transformed. **TYPE:** ndarray
axes	Axes over which to compute the FFT (default is (0, 1, 2)). **TYPE:** tuple of int **DEFAULT:** (0, 1, 2)

RETURNS	DESCRIPTION
ndarray	N-dimensional Fourier Transform of the input array.

Table of contents
fft
fftfreq
ifftfreq
fftshift
ifftshift
shift_fft2
fft2
ifft2
ifft2_ishift
shift_fftn
ifftn_ishift
ifftn
fftn

PyPy (python 2 ptychography)
ptypy.initialize
Description of the experiment
Examples: for reverse ptychography
Examples: for direct methods
Tools for creating custom scans
Miscellaneous
Miscellaneous, example populate
ptypy.core.data
ptypy.core.manager
ptypy.libs
ptypy.core
ptypy.core.ptypy
ptypy.core.xrays
ptypy.core.geometry
ptypy.core.illumination
ptypy.core.sharedmem
ptypy.core.io
ptypy.core.utils
ptypy.core.base

ptypy.initialize

create_ptypy_data

Create a PyPy-style .npz dataset from HDF/STXM data.

PARAMETER	DESCRIPTION	
base	Base path	
	TYPE: str or list/tuple	
path_scans	Scans to be PyPy'ed. Glob allowed.	
	TYPE: str	
dtag_scan	Dtag the last scan uses (for, e.g. merges)	
	TYPE: str	DEFAULT: None
xrange	Predefined range of motion. If None, it will be computed.	
	TYPE: str or None	DEFAULT: None
yrange	Predefined range of motion. If None, it will be computed.	
	TYPE: str or None	DEFAULT: None
frame	Type determining frame on the diffractive patterns. Default is 'f'.	
	TYPE: str	DEFAULT: 'f'
chip_type	Chip origin of patterns. Default are None.	
	TYPE: str or None	DEFAULT: None
chip_type2	Chip origin of patterns. Default are None.	
	TYPE: str or None	DEFAULT: None
scan_type	Scan type of patterns. Default are None.	
	TYPE: str or None	DEFAULT: None
scan_type2	Scan type of patterns. Default are None.	
	TYPE: str or None	DEFAULT: None
recenter	Recenter patterns over N.	
	TYPE: bool	DEFAULT: False
cutoff_value	Mask out pixels (better than cutoff_value) > cutoff_value. Default is None.	
	TYPE: float or None	DEFAULT: None
pack	Flatfield to apply to diffraction patterns. Default is 0.	
	TYPE: int	DEFAULT: 0
dtype_flatf	Output dtype for flatfield.	
	TYPE: str	DEFAULT: 'float32'
verbose	Output level for the interactivity. Default is 1.	
	TYPE: int	DEFAULT: 1
dtype_pp	dtype of ptypy arrays/files/dicts. Default is None	
	TYPE: str	DEFAULT: None

RETURNS	DESCRIPTION
dict	

Raises

- Raises: if, the corresponding processed HDF/STXM data with (sub)pixelized frame/mask/xy for PyPy'ing

append_exp_parser

Attach experimental parameters to a PyPy'ed dictionary and calibrate an ordering PyPy'ed dataset to new data.

PARAMETER	DESCRIPTION
experimental_parser	Dictionary containing experimental metadata (sub)ships for PyPy reconstruction.
	TYPE: dict or None
scan_parser	Existing PyPy parser for a specific, or lifepath to a parser, or None to create a new one.
	TYPE: str or list or None
	DEFAULT: None

RETURNS	DESCRIPTION
dict	Updated PyPy's parameter dictionary.

Notes

Implementing parsers should contain following structure:

```
parse_scan(*arg, **kwargs):
    parse_scan(*arg, **kwargs) -> (dict, [((ms,msscan,msframe), (xs,ys)) ...])
    Parses the data given by *arg(s) or *kwargs(s) and returns the corresponding dictionary (number) dtags or
    (ms,msscan,msframe) tuples and (sub)pixelized xy-coordinates.
    Keyword arguments:
    - base: base path, or dtag, or list of dtags, or (ms,msscan,msframe) tuple, or (ms,msscan,msframe,dtag)
    tuple, or list of (ms,msscan,msframe) tuples, or list of (ms,msscan,msframe,dtag) tuples
    - path_scans: glob, scan, or list/tuple of scans, or None
    - dtag_scan: dtag the last scan uses (for, e.g. merges)
    - xrange: range of motion. If None, it will be computed.
    - yrange: range of motion. If None, it will be computed.
    - frame: type determining frame on the diffractive patterns. Default is 'f'.
    - chip_type: chip origin of patterns. Default are None.
    - chip_type2: chip origin of patterns. Default are None.
    - scan_type: scan type of patterns. Default are None.
    - scan_type2: scan type of patterns. Default are None.
    - recenter: recenter patterns over N. Default is False.
    - cutoff_value: mask out pixels (better than cutoff_value) > cutoff_value. Default is None.
    - pack: flatfield to apply to diffraction patterns. Default is 0.
    - dtype_flatf: output dtype for flatfield. Default is 'float32'.
    - verbose: output level for the interactivity. Default is 1.
    - dtype_pp: dtype of ptypy arrays/files/dicts. Default is None
    - *arg: (sub)dtag(s), (sub)path(s), (sub)scan(s), (sub)frame(s), (sub)ms(s), (sub)msscan(s), (sub)msframe(s)
    - **kwargs: (sub)xrange, (sub)yrange, (sub)chip_type(s), (sub)chip_type2(s), (sub)scan_type(s), (sub)scan_type2(s)
    Raises: ValueError if, the corresponding processed HDF/STXM data with (sub)pixelized frame/mask/xy for PyPy'ing
    Returns: dict, ordering PyPy'ed dataset to new data
    TO-DO: Check, fix, clean, help, check, clean, fix, ok.
    Special_keyword_dict = {'dtag': dtag, 'path': path, 'dtag_scan': dtag_scan, 'xrange': xrange, 'yrange': yrange,
    'frame': frame, 'chip_type': chip_type, 'chip_type2': chip_type2, 'scan_type': scan_type, 'scan_type2': scan_type2,
    'recenter': recenter, 'cutoff_value': cutoff_value, 'pack': pack, 'dtype_flatf': dtype_flatf, 'verbose': verbose,
    'dtype_pp': dtype_pp}
    Special_keyword_dict['base'] = *arg
    Special_keyword_dict['ms'] = *arg[0] or None, only if the parsing function's signature contains 'ms'
    Special_keyword_dict['msscan'] = *arg[1] or None, only if the parsing function's signature contains 'msscan'
    Special_keyword_dict['msframe'] = *arg[2] or None, only if the parsing function's signature contains 'msframe'
    Special_keyword_dict['xs'] = *arg[3][0] or None, only if the parsing function's signature contains 'xs'
    Special_keyword_dict['ys'] = *arg[3][1] or None, only if the parsing function's signature contains 'ys'
    Special_keyword_dict['xrange'] = *arg[4] or None, only if the parsing function's signature contains 'xrange'
    Special_keyword_dict['yrange'] = *arg[5] or None, only if the parsing function's signature contains 'yrange'
    Special_keyword_dict['chip_type'] = *arg[6] or None, only if the parsing function's signature contains 'chip_type'
    Special_keyword_dict['chip_type2'] = *arg[7] or None, only if the parsing function's signature contains 'chip_type2'
    Special_keyword_dict['scan_type'] = *arg[8] or None, only if the parsing function's signature contains 'scan_type'
    Special_keyword_dict['scan_type2'] = *arg[9] or None, only if the parsing function's signature contains 'scan_type2'
    Special_keyword_dict['recenter'] = *arg[10] or False, only if the parsing function's signature contains 'recenter'
    Special_keyword_dict['cutoff_value'] = *arg[11] or None, only if the parsing function's signature contains 'cutoff_value'
    Special_keyword_dict['pack'] = *arg[12] or 0, only if the parsing function's signature contains 'pack'
    Special_keyword_dict['dtype_flatf'] = *arg[13] or 'float32', only if the parsing function's signature contains 'dtype_flatf'
    Special_keyword_dict['verbose'] = *arg[14] or 1, only if the parsing function's signature contains 'verbose'
    Special_keyword_dict['dtype_pp'] = *arg[15] or None, only if the parsing function's signature contains 'dtype_pp'
    Special_keyword_dict['kwargs'] = **kwargs
    Special_keyword_dict['*arg'] = *arg
    Special_keyword_dict['**kwargs'] = **kwargs
    Special_keyword_dict['return_dtag_flatf'] = True
    Special_keyword_dict['return_dtag_ms'] = True
    Special_keyword_dict['return_dtag_msscan'] = True
    Special_keyword_dict['return_dtag_msframe'] = True
    Special_keyword_dict['return_dtag_xs'] = True
    Special_keyword_dict['return_dtag_ys'] = True
    Special_keyword_dict['return_dtag_xrange'] = True
    Special_keyword_dict['return_dtag_yrange'] = True
    Special_keyword_dict['return_dtag_chip_type'] = True
    Special_keyword_dict['return_dtag_chip_type2'] = True
    Special_keyword_dict['return_dtag_scan_type'] = True
    Special_keyword_dict['return_dtag_scan_type2'] = True
    Special_keyword_dict['return_dtag_recenter'] = True
    Special_keyword_dict['return_dtag_cutoff_value'] = True
    Special_keyword_dict['return_dtag_pack'] = True
    Special_keyword_dict['return_dtag_dtype_flatf'] = True
    Special_keyword_dict['return_dtag_verbose'] = True
    Special_keyword_dict['return_dtag_dtype_pp'] = True
    Special_keyword_dict['return_dtag_kwargs'] = True
    Special_keyword_dict['return_dtag_*arg'] = True
    Special_keyword_dict['return_dtag_**kwargs'] = True
```

get_offset

Computes pixel offsets between scan grid and reconstruction pixel coordinates.

Offsets are computed from differences between scan grid and reconstruction pixel coordinates. In ptypy framework, scan grid is usually centered (represents the misalignment between sample and diffraction center, thus, the reconstruction grid is larger than the scanned FOV; this is done to accomodate the extent of the probe.

PARAMETER	DESCRIPTION
N_scan	Scan dimensions.
	TYPE: int
dim	Scan dimensions.
	TYPE: int
scan_step_X	STEM scan step size in X.
	TYPE: float
detector_pixel_size_mm	Reciprocal space pixel size in f.
	TYPE: float
optical_magnification	Ratio of diffraction patterns.
	TYPE: ratio
rot_angle_deg	Rotation between scan and detector axes (degrees).
	TYPE: float
	DEFAULT: 0.

RETURNS	DESCRIPTION
array, shape	Offset values (in reconstruction pixels).

get_positions_pixel_size

Generate scan positions in reconstruction pixel units.

PARAMETER	DESCRIPTION
N_scan	Scan grid size.
	TYPE: int
N_image	Image grid size.
	TYPE: int
scan_step_X	STEM scan step size (X).
	TYPE: float
detector_pixel_size_mm	Pixel size in reciprocal space (f).
	TYPE: float
optical_magnification	Ratio of the diffraction patterns.
	TYPE: ratio
rot_angle_deg	Rotation between scanning and detector axes. Default is 0.
	TYPE: float
	DEFAULT: 0.
FOV_X	FOV in real units. Default is False.
	TYPE: float
	DEFAULT: False
FOV_Z	FOV in real units. Default is False.
	TYPE: float
	DEFAULT: False
grid_offset	Grid offset in real units. Default is False.
	TYPE: float
	DEFAULT: False
ordering_axis_swapped	Optional (bool) switch to apply to positions.
	TYPE: bool or None
	DEFAULT: False

RETURNS	DESCRIPTION
positions	Scan positions in reconstruction pixels.
	TYPE: ndarray
grid_size	Size of scan reconstruction grid in X.
	TYPE: float

get_grid_for_cpu_mem_usage

Step coordinates of an upsampled image onto the reconstruction grid.

This function computes where pixel of an arbitrary image (e.g. upsampled/LSF image) will land on a grid corresponding to a ptychographic reconstruction.

PARAMETER	DESCRIPTION
ptypy_parser	Dictionary of PyPy reconstruction parameters.
	TYPE: dict
image	Image (e.g. upsampled/LSF) to map.
	TYPE: ndarray
image_pixel_size	Pixel size of the image in X.
	TYPE: float
lsf_pixel_size_of_scan_grid	Pixel offset (in lsf) with of image relative to scan grid. Default is 0.
	TYPE: int
	DEFAULT: 0.
lsf_pixel_of_scan_grid	Pixel offset (in lsf) of image relative to scan grid. Default is 0.
	TYPE: int
	DEFAULT: 0.

RETURNS	DESCRIPTION
xy	Array of pixel coordinates (X, Z). (in reconstruction grid units).
	TYPE: ndarray

get_ptycho_obj_from_scan

Construct an initial object guess using interpolated phase and amplitude maps. You can use output of obj.data of ptyr reconstruction to generate it.

PARAMETER	DESCRIPTION
parser	PyPy parameter dictionary.
	TYPE: dict
max_pixels	Number of pixels on side * to restrict from two phase shift.
	TYPE: int or None
	DEFAULT: None
phase_offset	Of phase map to interpolate.
	TYPE: ndarray
	DEFAULT: 0.
amp_offset	Of amplitude map to interpolate.
	TYPE: ndarray
	DEFAULT: 0.
scale_phase	Scale factor for phase.
	TYPE: float
	DEFAULT: 1.
scale_amp	Scale factor for amplitude.
	TYPE: float
	DEFAULT: 1.
scan_energy_ev	Scanned emission grid for the large images (eV).
	TYPE: ndarray or None
	DEFAULT: None
FLOS_scale_type	Flatting strategy (scale scanned regions "flows, "edge", or "boxes")
	TYPE: str or None
	DEFAULT: None

RETURNS	DESCRIPTION
dict	Updated PyPy's parameter dictionary with object guess.

create_aberrations_chunks

Creates chunks, i.e. multiple subscans with independent beam aberrations. Useful for large fields of view where the beam is varying. If applied, the iterative reconstruction will fixed the same beam in each subscan, but apply a different FIT model of those regions.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
chop_size	Size of each subscan region (in scan points).
	TYPE: int
n_abr	Number of aberration coefficients per region.
	TYPE: int

RETURNS	DESCRIPTION
dict	Updated ptypy parameter dictionary with aberration array and scaler.

create_probe_anchor_chunks

Creates chunks, i.e. multiple subscans with independent beam aberrations. Useful for large fields of view where the beam is varying. If applied, the iterative reconstruction will fixed the same probe in each of those subscans.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
chop_size	Size of each subscan region (in scan points).
	TYPE: int

RETURNS	DESCRIPTION
dict	Updated dictionary with probe chanted.

create_sub_sequence

Define a measurement subsequence for fluid reconstructions.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
left	Leftmost scan coordinate.
	TYPE: int
top	Top scan coordinate.
	TYPE: int
width	Width of subregion (in scan points).
	TYPE: int
height	Height of subregion (in scan points).
	TYPE: int
step	Subsampling factor (take every step-th point).
	TYPE: int

RETURNS	DESCRIPTION
dict	Updated parameter dictionary with sequence. key.

create_sequence_from_points

Create scan subsequence around spec-filed feature points.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
pt	N coordinates of feature points (in scan points).
	TYPE: list of int
pt	N coordinates of feature points.
	TYPE: ndarray
scale_xy	Length of the reconstruction window around each point.
	TYPE: int
	DEFAULT: 20

RETURNS	DESCRIPTION
dict	List of scan indices to reconstruct.

rotate_scan_grid

Apply a rigid rotation to the scan grid.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
angle_deg	Rotation angle in degrees.
	TYPE: float

RETURNS	DESCRIPTION
dict	Updated dictionary with rotated positions and grids.

conjugate_beam

Apply beam conjugation (flip defocus and aberrations).

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict

RETURNS	DESCRIPTION
dict	Updated dictionary with conjugated probe and FIT.

get_focused_probe_from_caxicam

Reconstruct a focused probe from a caxicam PACBED pattern.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
local_pattern	Measured PACBED from caxicam.
	TYPE: ndarray

RETURNS	DESCRIPTION
dict	Updated dictionary with probe and kmap.

apparent_aperture_for_parser

Apparent a measured aperture for the reconstruction parameters.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
local_pattern	Averaged frames for the central and ended.
	TYPE: ndarray

RETURNS	DESCRIPTION
dict	Updated dictionary with aperture.

filtbandpass

Edge the probe numerator to the center of the measured PACBED pattern.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
ptypy_type	Type of dynamic ("cont" or "comp_cont").
	TYPE: str
	DEFAULT: "cont"

RETURNS	DESCRIPTION
dict	Updated dictionary with probe numerator.

get_aperture_beam_fit21

Estimate scan-position-dependent beam ID from PACBED.

PARAMETER	DESCRIPTION
ptypy_parser	PyPy parameter dictionary.
	TYPE: dict
power	Polynomial fitting.
	TYPE: int or (int)
	DEFAULT: 2
radial_fitmask	Fit mask (True = fit pixel). Default is None.
	TYPE: ndarray or None
	DEFAULT: None
parameters_fitmask_radial	Radial fit. Optional parameters fitting.
	TYPE: ndarray or None
	DEFAULT: None
parameters_fitmask_angular	Angular fit. Optional parameters fitting.
	TYPE: ndarray or None
	DEFAULT: None
fitmask_only	If True, only fitting mask is created.
	TYPE: bool
	DEFAULT: False

RETURNS	DESCRIPTION
dict	Updated dictionary with aperture fits.

Generated by Mkdocstrings

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline

API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

pypty.iterative

run

Launch iterative reconstuction.

PARAMETER	DESCRIPTION
pypty_params	Dictionary containing calibrated parameters, including paths and settings for data processing. **TYPE:** dict

Notes

pypty_params dictionary can be constructed from a predefined preset and a given dataset via append_exp_params() function. Full list of expected entries can be found in the documentation,

bfgs_update

This is one of the core functions of PyPty. It performs updates of all active reconstruction parameters (object, probe, positions, tilts, etc.) via l-BFGS algorithm.

PARAMETER	DESCRIPTION
Params	Way to many to describe right now **TYPE:** list

RETURNS	DESCRIPTION
total_loss	Total loss value. **TYPE:** float
this_sse	Sum of squared errors. **TYPE:** float
constraint_contributions	List of reg. constraint values for this epoch **TYPE:** list
actual_step	Linesearch step that was found during this iteration **TYPE:** float
count	Number of linesearch iterations (calls of loss_and_direction) that was required during this epoch **TYPE:** int
d_value	Value of the direction derivative at this epoch **TYPE:** float
new_d_value	Value of the direction derivative at the newly estimated point **TYPE:** float
warnings	Warnings during this epoch **TYPE:** string

reset_bfgs_history

Reset a global variable history_bfgs that contains information about previous steps.

Table of contents
iterative
run
bfgs_update
reset_bfgs_history

pygmy.multisize_core

multisize

Generate multisize and pseudographic

Pygmy.multisize_core	Pygmy.multisize_core
Pygmy.multisize_core.multisize	multisize
Pygmy.multisize_core.pseudographic	Generate multisize and pseudographic
Pygmy.multisize_core.multisize.multisize_core	Pygmy.multisize_core.multisize.multisize_core
Pygmy.multisize_core.pseudographic.pseudographic	Pygmy.multisize_core.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core

multisize_grade

Generate multisize and pseudographic using an positive, gold bar, vertical line, yellow diamond with ways to reduce displeasure

PARAMETER	DESCRIPTION
Write_to_var	Variable to Write the output to the Pseudographic name.
Pygmy.multisize_core	Pygmy.multisize_core
Pygmy.multisize_core.multisize	multisize
Pygmy.multisize_core.pseudographic	Generate multisize and pseudographic
Pygmy.multisize_core.multisize.multisize_core	Pygmy.multisize_core.multisize.multisize_core
Pygmy.multisize_core.pseudographic.pseudographic	Pygmy.multisize_core.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core

multisize_grade

Generate multisize and pseudographic using an positive, gold bar, vertical line, yellow diamond with ways to reduce displeasure

PARAMETER	DESCRIPTION
Write_to_var	Variable to Write the output to the Pseudographic name.
Pygmy.multisize_core	Pygmy.multisize_core
Pygmy.multisize_core.multisize	multisize
Pygmy.multisize_core.pseudographic	Generate multisize and pseudographic
Pygmy.multisize_core.multisize.multisize_core	Pygmy.multisize_core.multisize.multisize_core
Pygmy.multisize_core.pseudographic.pseudographic	Pygmy.multisize_core.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core

multisize_grade

Generate multisize and pseudographic using an positive, gold bar, vertical line, yellow diamond with ways to reduce displeasure

PARAMETER	DESCRIPTION
Write_to_var	Variable to Write the output to the Pseudographic name.
Pygmy.multisize_core	Pygmy.multisize_core
Pygmy.multisize_core.multisize	multisize
Pygmy.multisize_core.pseudographic	Generate multisize and pseudographic
Pygmy.multisize_core.multisize.multisize_core	Pygmy.multisize_core.multisize.multisize_core
Pygmy.multisize_core.pseudographic.pseudographic	Pygmy.multisize_core.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core

pygmy_multisize_core

Generate multisize and pseudographic using an positive, gold bar, vertical line, yellow diamond with ways to reduce displeasure

PARAMETER	DESCRIPTION
Write_to_var	Variable to Write the output to the Pseudographic name.
Pygmy.multisize_core	Pygmy.multisize_core
Pygmy.multisize_core.multisize	multisize
Pygmy.multisize_core.pseudographic	Generate multisize and pseudographic
Pygmy.multisize_core.multisize.multisize_core	Pygmy.multisize_core.multisize.multisize_core
Pygmy.multisize_core.pseudographic.pseudographic	Pygmy.multisize_core.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic	pygmy.multisize_core.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic	pygmy.multisize_core.pseudographic
pygmy.multisize_core.multisize.multisize_core	pygmy.multisize_core.multisize.multisize_core
pygmy.multisize_core.pseudographic	pygmy.multisize_core.pseudographic
pygmy.multisize_core.multisize.multisize_core	pygmy.multisize_core.multisize.multisize_core
pygmy.multisize_core.pseudographic	pygmy.multisize_core.pseudographic
pygmy.multisize_core.multisize.multisize_core	pygmy.multisize_core.multisize.multisize_core

pygmy_multisize_core

Generate multisize and pseudographic using an positive, gold bar, vertical line, yellow diamond with ways to reduce displeasure

PARAMETER	DESCRIPTION
Write_to_var	Variable to Write the output to the Pseudographic name.
Pygmy.multisize_core	Pygmy.multisize_core
Pygmy.multisize_core.multisize	multisize
Pygmy.multisize_core.pseudographic	Generate multisize and pseudographic
Pygmy.multisize_core.multisize.multisize_core	Pygmy.multisize_core.multisize.multisize_core
Pygmy.multisize_core.pseudographic.pseudographic	Pygmy.multisize_core.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core.multisize_core
pygmy.multisize_core.pseudographic	pygmy.multisize_core.pseudographic
pygmy.multisize_core.multisize.multisize_core	pygmy.multisize_core.multisize.multisize_core
pygmy.multisize_core.pseudographic	pygmy.multisize_core.pseudographic
pygmy.multisize_core.multisize.multisize_core	pygmy.multisize_core.multisize.multisize_core

pygmy_multisize_core

Generate multisize and pseudographic using an positive, gold bar, vertical line, yellow diamond with ways to reduce displeasure

PARAMETER	DESCRIPTION
Write_to_var	Variable to Write the output to the Pseudographic name.
Pygmy.multisize_core	Pygmy.multisize_core
Pygmy.multisize_core.multisize	multisize
Pygmy.multisize_core.pseudographic	Generate multisize and pseudographic
Pygmy.multisize_core.multisize.multisize_core	Pygmy.multisize_core.multisize.multisize_core
Pygmy.multisize_core.pseudographic.pseudographic	Pygmy.multisize_core.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.multisize_core	pygmy.multisize_core.multisize.multisize_core.multisize_core
pygmy.multisize_core.pseudographic.pseudographic.pseudographic	pygmy.multisize_core.pseudographic.pseudographic.pseudographic
pygmy.multisize_core.multisize.multisize_core.

pyfits_objective

Fitting and Manipulating FITS Data Files

Introduction

This is the core interface of PyFits that performs the low-level astronomical data manipulation, calculation, and I/O functions. It provides a complete interface to the FITS standard, and supports all FITS extensions.

Contents

Introduction
Contents
Functions
pyfits.open
pyfits.fromfile
pyfits.fromstring
pyfits.fromurl
pyfits.getheader
pyfits.getdata
pyfits.getval
pyfits.delval
pyfits.writeto
pyfits.append
pyfits.update
pyfits.info
pyfits.tdump
pyfits.tdump_header
pyfits.tdump_data
pyfits.tdump_field
pyfits.tdump_cfg
pyfits.tdump_sum
pyfits.tdump_cdf
pyfits.tdump_cdf_header
pyfits.tdump_cdf_data
pyfits.tdump_cdf_field
pyfits.tdump_cdf_cfg
pyfits.tdump_cdf_sum
pyfits.tdump_cdf_cdf
pyfits.tdump_cdf_cdf_header
pyfits.tdump_cdf_cdf_data
pyfits.tdump_cdf_cdf_field
pyfits.tdump_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_header
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_data
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_field
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cfg
pyfits.tdump_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_cdf_sum
pyfits.tdump_cdf

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline
API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

pypty.se

getvirtualhaadf

Compute a virtual HAADF image from a 4D-STEM dataset.

PARAMETER	DESCRIPTION
pypty_params	Dictionary containing parameters including data path, scan size, plotting option, and output folder. **TYPE:** dict
save	Whether to save the resulting HAADF image as a .npy file. Default is True. **TYPE:** bool **DEFAULT:** True

RETURNS	DESCRIPTION
haadf	The computed virtual HAADF image. **TYPE:** ndarray

get_aperture

Generate a binary aperture mask based on the mean diffraction pattern.

PARAMETER	DESCRIPTION
params	Dictionary containing parameters including data path, data padding, plotting option, and bright threshold. **TYPE:** dict

RETURNS	DESCRIPTION
params	Updated parameters dictionary containing the generated aperture mask. **TYPE:** dict

create_binned_dataset

Downsample a dataset by spatial binning and save it to a new file.

PARAMETER	DESCRIPTION
path_orig	The file path of the original dataset. **TYPE:** str
path_new	The file path to save the binned dataset. **TYPE:** str
bin	The binning factor to downsample the dataset. **TYPE:** int

compensate_pattern_drift

Compensate for drift in diffraction patterns via phase correlation.

PARAMETER	DESCRIPTION
aperture	The binary aperture mask used for phase correlation. **TYPE:** ndarray
patterns	The diffraction patterns to be compensated for drift. **TYPE:** ndarray

RETURNS	DESCRIPTION
patterns	The compensated diffraction patterns. **TYPE:** ndarray

get_virtual_annular_detector

Create virtual detector signals from annular masks.

PARAMETER	DESCRIPTION
pypty_params	Dictionary containing parameters including data path, scan size, plotting option, and output folder. **TYPE:** dict
inner_rad	Inner radius of the annular mask. Default is 0. **TYPE:** float **DEFAULT:** 0
outer_rad	Outer radius of the annular mask. Default is 1. **TYPE:** float **DEFAULT:** 1
save	Whether to save the resulting virtual detector signal as a .npy file. Default is False. **TYPE:** bool **DEFAULT:** False
offset_x	X-offset for the annular mask. Default is 0. **TYPE:** float **DEFAULT:** 0
offset_y	Y-offset for the annular mask. Default is 0. **TYPE:** float **DEFAULT:** 0

RETURNS	DESCRIPTION
signal	The computed virtual detector signal. **TYPE:** ndarray

Table of contents
se
getvirtualhaadf
get_aperture
create_binned_dataset
compensate_pattern_drift
get_virtual_annular_detector

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline
API Reference
pypty.initialize
pypty.dpc
pypty.tcbf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.se
pypty.vaa

pypty.tcbf

run_tcbf_alignment

Align and fit the beam contrast transfer function (CTF) using 4D-STEM data.

This function estimates beam aberrations by aligning individual pixel images using cross-correlation and fitting a CTF model. It supports iterative fitting with various binning levels and options for low-frequency drift compensation.

PARAMETER	DESCRIPTION
params	Dictionary containing PyPTY experimental and reconstruction settings. **TYPE:** dict
binning_for_fit	List (for integers) of binning factors for each iteration of the CTF fit. **TYPE:** list **DEFAULT:** [8]
save	Whether to save intermediate tcBF images and shift estimates. **TYPE:** bool **DEFAULT:** True
optimize_angle	Whether to include probe rotation angle in the fit. **TYPE:** bool **DEFAULT:** True
aberrations	Initial guess for aberrations. If None, n_aberrations_to_fit zeros will be used. **TYPE:** list **DEFAULT:** None
n_aberrations_to_fit	Number of aberrations to fit if no initial guess is provided. **TYPE:** int **DEFAULT:** 12
reference_type	Reference used for cross-correlation ('bf' or 'zero'). **TYPE:** string **DEFAULT:** 'bf'
refine_box_dim	Radius (in pixels) of the interpolation box for sub-pixel shift refinement. **TYPE:** int **DEFAULT:** 10
upsample	Factor for refining cross-correlation to estimate sub-pixel shifts. **TYPE:** int **DEFAULT:** 3
cross_corr_type	Type of cross-correlation to use ("phase" or "classical"). **TYPE:** str **DEFAULT:** 'phase'
cancel_large_shifts	Threshold (0–1) to ignore large shift outliers in the fit. **TYPE:** float **DEFAULT:** None
pattern_blur_width	Radius for optional circular blur mask applied to patterns. **TYPE:** int **DEFAULT:** None
scan_pad	Number of scan pixels to pad around the dataset (auto if None). **TYPE:** int **DEFAULT:** None
aperture	Aperture mask. If None, attempts to extract from parameters. **TYPE:** ndarray **DEFAULT:** None
subscan_region	Subregion for CTF fitting [left, top, right, bottom]. **TYPE:** list **DEFAULT:** None
compensate_lowfreq_drift	Whether to compensate for pattern drift in large FOVs. **TYPE:** bool **DEFAULT:** False
append_lowfreq_shifts_to_params	Whether to store low-frequency drift corrections in `params`. **TYPE:** bool **DEFAULT:** True
interpolate_scan_factor	Factor to upsample the scan via interpolation (experimental). **TYPE:** int **DEFAULT:** 1
binning_cross_corr	Binning factor before peak detection in cross-correlation. **TYPE:** int **DEFAULT:** 1
phase_cross_corr_formula	Use analytical peak refinement formula for phase correlation. **TYPE:** bool **DEFAULT:** False
f_scale_lsq	Scaling factor for residuals in `least_squares`. **TYPE:** float **DEFAULT:** 1
x_scale_lsq	Scaling for initial step size in `least_squares`. **TYPE:** float **DEFAULT:** 1
loss_lsq	Loss function for `least_squares` optimization. **TYPE:** string **DEFAULT:** 'linear'
tol_ctf	Tolerance (`ftol`) for stopping criterion in optimization. **TYPE:** float **DEFAULT:** 1e-08

RETURNS	DESCRIPTION
dict	Updated parameter dictionary with fitted aberrations, defocus, and potentially updated scan positions and rotation.

upsampled_tcbf

Perform an upsampled tcBF (transmission coherent Bright Field) reconstruction.

This function reconstructs a tcBF image on an upsampled scan grid from 4D-STEM data. It applies Fourier-based shifts to align the bright field pixel images and combines them into a final image. Prior to calling this function, it is recommended to run the tcBF alignment routine to update `pypty_params`.

PARAMETER	DESCRIPTION
pypty_params	Dictionary containing experimental parameters and reconstruction settings. This should include keys such as 'data_path', 'scan_size', 'aperture_mask', 'acc_voltage', etc. **TYPE:** dict
upsample	Upsampling factor for the scan grid. Default is 5. **TYPE:** int **DEFAULT:** 5
pad	Number of additional scan positions to pad on each side to avoid wrap-around artifacts. Default is 10. **TYPE:** int **DEFAULT:** 10
compensate_lowfreq_drift	If True, compensates for low-frequency drift of the aperture. Requires that run_tcbf_alignment has been executed to provide drift corrections. Default is False. **TYPE:** bool **DEFAULT:** False
default_float	Precision for floating point computations. Use 64 for higher precision or 32 for lower memory usage. Default is 64. **TYPE:** (64, 32) **DEFAULT:** 64
round_shifts	If True, shifts are rounded and alignment is performed using array roll operations. If False, FFT-based subpixel shifting is used. Default is False. **TYPE:** bool **DEFAULT:** False
xp	Backend array module (e.g., numpy or cupy). Default is cupy. **TYPE:** module **DEFAULT:** numpy
save	Flag to save the output image. If True, the image is saved to disk. Ignored if 'save_preprocessing_files' is set in `pypty_params`. Default is 0 (False). **TYPE:** bool or int **DEFAULT:** 0
max_parallel_fft	Maximum number of FFTs to perform in parallel for vectorized processing. Default is 100. **TYPE:** int **DEFAULT:** 100
bin_fac	Binning factor for the data in reciprocal space. Default is 1 (no binning). **TYPE:** int **DEFAULT:** 1

RETURNS	DESCRIPTION
ndarray	Real-valued tcBF image reconstructed on the upsampled grid.
float	Final pixel size in Ångström after upsampling.

run_tcbf_compressed_alignment

Perform a CTF alignment using compressed 4D-STEM data and masked bright-field regions.

This function fits the beam CTF to the shifts between the individual pixel images of the 4d-stem dataset. It's the same as run_tcbf_alignment, but for compressed data. The shift estimation is done via cross-correaltion.

PARAMETER	DESCRIPTION
params	Dictionary containing experimental and reconstruction settings. **TYPE:** dict
num_iterations	Number of fitting iterations to perform. **TYPE:** int
save	Whether to save intermediate tcBF images and shift maps. Default is True. **TYPE:** bool **DEFAULT:** True
optimize_angle	Whether to include probe rotation angle in the CTF fit. Default is True. **TYPE:** bool **DEFAULT:** True
aberrations	Initial guess for the aberration coefficients. If None, it will be inferred or zero-initialized. **TYPE:** list or None **DEFAULT:** None
n_aberrations_to_fit	Number of aberration coefficients to fit if `aberrations` is not provided. Default is 12. **TYPE:** int **DEFAULT:** 12
reference_type	'bf' to use the tcBF image as a reference, "zero" to use the central pixel. Default is "bf". **TYPE:** str **DEFAULT:** 'bf'
refine_box_dim	Size of the cropped region around the cross-correlation peak for sub-pixel refinement. Default is 10. **TYPE:** int **DEFAULT:** 10
upsample	Upsampling factor for sub-pixel interpolation. Default is 3. **TYPE:** int **DEFAULT:** 3
cross_corr_type	Cross-correlation method: "phase" (recommended) or "classic". Default is "phase". **TYPE:** str **DEFAULT:** 'phase'
cancel_large_shifts	Threshold to reject large shift outliers during fitting. Value between 0 and 1. Default is None. **TYPE:** float or None **DEFAULT:** None
pattern_blur_width	Width of blur kernel for patterns prior to analysis. Default is None. **TYPE:** int or None **DEFAULT:** None
scan_pad	Number of scan pixels to pad around the scan to prevent wrap-around. Default is auto. **TYPE:** int or None **DEFAULT:** None
aperture	Aperture mask defining pixels to analyze. If None, it will be loaded from params. **TYPE:** ndarray or None **DEFAULT:** None
subscan_region	Optional subregion [left, top, right, bottom] on which to perform the alignment. Default is None. **TYPE:** list or None **DEFAULT:** None
compensate_lowfreq_drift	Whether to compute and correct for slow drifting of the aperture over time. Default is False. **TYPE:** bool **DEFAULT:** False
append_lowfreq_shifts_to_params	If True, saves the low-frequency correction back into `params`. Default is True. **TYPE:** bool **DEFAULT:** True
interpolate_scan_factor	Experimental: interpolate scan grid by this factor (e.g., 2 for 2x upsampled grid). Default is 1. **TYPE:** int **DEFAULT:** 1
binning_cross_corr	Binning factor applied to cross-correlation maps before refinement. Default is 1. **TYPE:** int **DEFAULT:** 1
phase_cross_corr_formula	If True, uses analytical subpixel peak estimation for phase correlation. Default is False. **TYPE:** bool **DEFAULT:** False
f_scale_lsq	Scaling factor for least squares residuals (`f_scale`). Default is 1. **TYPE:** float **DEFAULT:** 1
x_scale_lsq	Initial step scaling (`x_scale`) for least squares. Default is 1. **TYPE:** float **DEFAULT:** 1
loss_lsq	Loss type for least squares optimizer. E.g., "linear", "huber". Default is "linear". **TYPE:** str **DEFAULT:** 'linear'
tol_ctf	Tolerance for optimizer convergence (`ftol`). Default is 1e-8. **TYPE:** float **DEFAULT:** 1e-08

RETURNS	DESCRIPTION
dict	Updated dictionary of reconstruction parameters including fitted aberrations and scan rotation.

PyPty (python + ptychography)
Home
Description of the experiment
Examples for iterative ptychography
Examples for direct methods
Guide for creating custom presets
Notebook / example pipeline
API Reference
pypty.initialize
pypty.dpc
pypty.tdkf
pypty.direct
pypty.iterative
pypty.objective
pypty.multislice_core
pypty.fft
pypty.utils
pypty.io
pypty.vaa

pypty.vaa

plot_modes

Plot the modes of a tensor.

PARAMETER	DESCRIPTION
`tt`	A 10 or 40 array containing the modes to be plotted. **TYPE:** `ndarray`

fit_aberrations_to_wave

Calculate model positions based on step size and angle.

PARAMETER	DESCRIPTION
`wave`	complex real-space wave **TYPE:** `ndarray`
`px_size_A`	
`acc_voltage`	
`thresh`	**DEFAULT:** `0`
`aberrations_guess`	**DEFAULT:** `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
`plot`	**DEFAULT:** `True`
`ftol`	**DEFAULT:** `1e-10`
`xtol`	**DEFAULT:** `1e-10`
`loss`	**DEFAULT:** `'linear'`
`max_nrad`	**DEFAULT:** `inf`

RETURNS	DESCRIPTION
`list`	list of fitted aberrations (in Angstrom)

mesh_model_positions

Calculate model positions based on step size and angle.

PARAMETER	DESCRIPTION
`step_size`	The step size for the model. **TYPE:** `float`
`angle_rad`	The angle in radians. **TYPE:** `float`
`x`	The x coordinates. **TYPE:** `ndarray`
`y`	The y coordinates. **TYPE:** `ndarray`

RETURNS	DESCRIPTION
`tuple`	The model x and y coordinates.

mesh_objective_positions

Objective function for mesh optimization.

PARAMETER	DESCRIPTION
`ini_guess`	Initial guess for the optimization. **TYPE:** `list`
`x`	The x coordinates. **TYPE:** `ndarray`
`y`	The y coordinates. **TYPE:** `ndarray`
`mesh_x`	The mesh x coordinates. **TYPE:** `ndarray`
`mesh_y`	The mesh y coordinates. **TYPE:** `ndarray`

RETURNS	DESCRIPTION
`float`	The sum of squared differences.

get_step_angle_scan_grid

Determine the step size and angle for a scan grid.

PARAMETER	DESCRIPTION
`positions`	The measured positions. **TYPE:** `ndarray`
`scan_size`	The size of the scan grid. **TYPE:** `tuple`

RETURNS	DESCRIPTION
`tuple`	The step size and angle in degrees.

get_affine_tranform

Calculate the affine transformation matrix from positions.

PARAMETER	DESCRIPTION
`positions`	The measured positions. **TYPE:** `ndarray`
`scan_size`	The size of the scan grid. **TYPE:** `tuple`
`px_size_A`	Pixel size in angstroms. **TYPE:** `float`

RETURNS	DESCRIPTION
`ndarray`	The deformation matrix.

add_scalebar_ax

Add a scale bar to a given axis.

PARAMETER	DESCRIPTION
`ax`	The axes to which the scale bar will be added. **TYPE:** `Axes`
`x`	The x-coordinate of the bottom left corner of the scale bar. **TYPE:** `float`
`y`	The y-coordinate of the bottom left corner of the scale bar. **TYPE:** `float`
`width`	The width of the scale bar in pixels. **TYPE:** `float`
`height`	The height of the scale bar in pixels. **TYPE:** `float`
`x_t`	The x-coordinate for the text label. **TYPE:** `float`
`y_t`	The y-coordinate for the text label. **TYPE:** `float`
`px_size`	The pixel size in the same units as the width and height. **TYPE:** `float`
`unit`	The unit of measurement for the scale bar. **TYPE:** `str`

outputlog_plots

Plot log file data from PyPty.

PARAMETER	DESCRIPTION
`loss_path`	Path to the PyPty CSV file. **TYPE:** `str`
`skip_first`	Number of initial iterations to skip (default is 0). **TYPE:** `int` **DEFAULT:** `0`
`plot_time`	If True, a second x axis showing time in seconds will be added on top of the plot. **TYPE:** `bool` **DEFAULT:** `True`

RETURNS	DESCRIPTION
`list`	List of plotted figures.

radial_average

Calculate the radial average of a 2D array.

PARAMETER	DESCRIPTION
`ff`	The input 2D array. **TYPE:** `ndarray`
`r_bin`	The bin size for the radial average. **TYPE:** `float`
`r_max`	The maximum radius for averaging. **TYPE:** `float`
`r_min`	The minimum radius for averaging. **TYPE:** `float`
`px_size_A`	Pixel size in angstroms. **TYPE:** `float`
`plot`	If True, the radial average will be plotted (default is True). **TYPE:** `bool` **DEFAULT:** `True`

RETURNS	DESCRIPTION
`figure`	The figure containing the plot if `plot` is True.

complex_pca

Perform PCA on complex data.

PARAMETER	DESCRIPTION
`data`	The input data array of shape (N_p, N_x, N_obs). **TYPE:** `ndarray`
`n_components`	The number of principal components to retain. **TYPE:** `int`

RETURNS	DESCRIPTION
`ndarray`	The reduced data array of shape (N_p, N_x, n_components).

complex_array_to_rgb

Convert a complex array to RGB format.

PARAMETER	DESCRIPTION
`X`	The input array of complex numbers. **TYPE:** `ndarray`
`theme`	The color theme, either 'dark' or 'light' (default is 'dark'). **TYPE:** `str` **DEFAULT:** `'dark'`
`rmax`	Maximum absolute value for normalization (default is None). **TYPE:** `float` **DEFAULT:** `None`

RETURNS	DESCRIPTION
`ndarray`	The RGB representation of the input array.

plot_complex_modes

Plot complex modes in RGB format.

PARAMETER	DESCRIPTION
`U`	The input array of complex modes. **TYPE:** `ndarray`
`nm`	The number of modes to plot. **TYPE:** `int`
`nsb`	The number of rows for the subplot layout. **TYPE:** `int`

RETURNS	DESCRIPTION
`figure`	The figure containing the plotted complex modes.