
dddm Documentation

Release 4.0.0

J. R. Angevaare

Apr 18, 2024

SETUP AND BASICS

1	Content	3
1.1	Setting up dddm	3
1.2	ddd package	3
2	Indices and tables	21
	Python Module Index	23
	Index	25

Github page: <https://github.com/JoranAngevaare/dddm>

This code is for Angevaare, J. R., et al. “Complementarity of direct detection experiments in search of light Dark Matter.” *Journal of Cosmology and Astroparticle Physics* 2022.10 (2022): 004. (<https://iopscience.iop.org/article/10.1088/1475-7516/2022/10/004>).

CONTENT

1.1 Setting up dddm

For most practical purposes, one can simply run *pip install dddm* to install it via <https://pypi.org/project/dddm/>.

For some implementations (MultiNest), one is advised to follow instructions as on this script: https://github.com/JoranAngevaere/dddm/blob/master/.github/scripts/install_on_linux.sh

That said, as ultranest is an implemented and working alternative method to MultiNest, a pip setup may prove much simpler.

1.2 dddm package

1.2.1 Subpackages

dddm.detectors package

Submodules

dddm.detectors.examples module

```
class dddm.detectors.examples.ArgonSimple(n_energy_bins=10, e_min_kev=0, e_max_kev=100)
```

Bases: *Experiment*

background_function(energies_in_kev)

Assume background free detector

cut_efficiency: `Union[int, float]` = 0.8

detection_efficiency: `Union[int, float]` = 0.8

detector_name: `str` = 'Ar_simple'

energy_threshold_kev: `Union[int, float]` = 30

exposure_tonne_year: `Union[int, float]` = 10

interaction_type: `str` = 'SI'

location: `str` = 'XENON'

resolution(*energies_in_kev*)

Simple square root dependency of the energy resolution

target_material: **str** = 'Ar'

class dddm.detectors.examples.**GermaniumSimple**(*n_energy_bins=10, e_min_kev=0, e_max_kev=100*)

Bases: *Experiment*

background_function(*energies_in_kev*)

Assume background free detector

cut_efficiency: **Union[int, float]** = 0.8

detection_efficiency: **Union[int, float]** = 0.9

detector_name: **str** = 'Ge_simple'

energy_threshold_kev: **Union[int, float]** = 10

exposure_tonne_year: **Union[int, float]** = 3

interaction_type: **str** = 'SI'

location: **str** = 'SUF'

resolution(*energies_in_kev*)

Simple resolution model

target_material: **str** = 'Ge'

class dddm.detectors.examples.**XenonSimple**(*n_energy_bins=10, e_min_kev=0, e_max_kev=100*)

Bases: *Experiment*

background_function(*energies_in_kev*)

Assume background free detector

cut_efficiency: **Union[int, float]** = 0.8

detection_efficiency: **Union[int, float]** = 0.5

detector_name: **str** = 'Xe_simple'

energy_threshold_kev: **Union[int, float]** = 10

exposure_tonne_year: **Union[int, float]** = 5

interaction_type: **str** = 'SI'

location: **str** = 'XENON'

resolution(*energies_in_kev*)

Simple square root dependency of the energy resolution

target_material: **str** = 'Xe'

dddm.detectors.experiment module

```
class dddm.detectors.experiment.Experiment(n_energy_bins=50, e_min_kev=0, e_max_kev=5)
```

Bases: `object`

Base class of experiments. To use, subclass and set the required attributes

background_function(*energies_in_kev: ndarray*) → ndarray

Return background at <energies [keV]>

property `config`

cut_efficiency: `Union[int, float]` = None

detection_efficiency: `Union[int, float]` = None

property `detector_hash`

detector_name: `str` = None

e_max_kev: `Union[int, float]` = None

e_min_kev: `Union[int, float]` = None

property `effective_exposure`

energy_threshold_kev: `Union[int, float]` = None

exposure_tonne_year: `Union[int, float]` = None

interaction_type: `str` = 'SI'

location: `str` = None

n_energy_bins: `int` = 50

resolution(*energies_in_kev: ndarray*) → ndarray

Return resolution at <energies [keV]>

target_material: `str` = None

dddm.detectors.super_cdms module

```
class dddm.detectors.super_cdms.SuperCdmsHvGeMigdal(n_energy_bins=50, e_min_kev=0,  
                                                    e_max_kev=5)
```

Bases: `_BaseSuperCdms`

background_function(*energies_in_kev*)

Flat bg rate

cut_efficiency: `Union[int, float]` = 0.85

detection_efficiency: `Union[int, float]` = 0.5

detector_name: `str` = 'SuperCDMS_HV_Ge_Migdal'

property `energy_threshold_kev`

```
    exposure_tonne_year: Union[int, float] = 0.044

    interaction_type: str = 'migdal_SI'

    resolution(energies_in_kev)
        Flat resolution

    target_material: str = 'Ge'

class dddm.detectors.super_cdms.SuperCdmsHvGeNr(n_energy_bins=50, e_min_kev=0, e_max_kev=5)
    Bases: _BaseSuperCdms

    background_function(energies_in_kev)
        Flat bg rate

    cut_efficiency: Union[int, float] = 0.85

    detection_efficiency: Union[int, float] = 0.85

    detector_name: str = 'SuperCDMS_HV_Ge_NR'

    energy_threshold_kev: Union[int, float] = 0.04

    exposure_tonne_year: Union[int, float] = 0.044

    interaction_type: str = 'SI'

    resolution(energies_in_kev)
        Flat resolution

    target_material: str = 'Ge'

class dddm.detectors.super_cdms.SuperCdmsHvSiMigdal(n_energy_bins=50, e_min_kev=0,
                                                    e_max_kev=5)

    Bases: _BaseSuperCdms

    background_function(energies_in_kev)
        Flat bg rate

    cut_efficiency: Union[int, float] = 0.85

    detection_efficiency: Union[int, float] = 0.675

    detector_name: str = 'SuperCDMS_HV_Si_Migdal'

    property energy_threshold_kev

    exposure_tonne_year: Union[int, float] = 0.0096

    interaction_type: str = 'migdal_SI'

    resolution(energies_in_kev)
        Flat resolution

    target_material: str = 'Si'

class dddm.detectors.super_cdms.SuperCdmsHvSiNr(n_energy_bins=50, e_min_kev=0, e_max_kev=5)
    Bases: _BaseSuperCdms
```

```

background_function(energies_in_kev)
    Flat bg rate
cut_efficiency: Union[int, float] = 0.85
detection_efficiency: Union[int, float] = 0.85
detector_name: str = 'SuperCDMS_HV_Si_NR'
energy_threshold_kev: Union[int, float] = 0.078
exposure_tonne_year: Union[int, float] = 0.0096
interaction_type: str = 'SI'
resolution(energies_in_kev)
    Flat resolution
target_material: str = 'Si'

class dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal(n_energy_bins=50, e_min_kev=0,
                                                    e_max_kev=5)

    Bases: _BaseSuperCdms
    background_function(energies_in_kev)
        Flat bg rate
    cut_efficiency: Union[int, float] = 0.75
    detection_efficiency: Union[int, float] = 0.5
    detector_name: str = 'SuperCDMS_iZIP_Ge_Migdal'
    property energy_threshold_kev
    exposure_tonne_year: Union[int, float] = 0.056
    interaction_type: str = 'migdal_SI'
    resolution(energies_in_kev)
        Flat resolution
    target_material: str = 'Ge'

class dddm.detectors.super_cdms.SuperCdmsIzipGeNr(n_energy_bins=50, e_min_kev=0, e_max_kev=5)
    Bases: _BaseSuperCdms
    background_function(energies_in_kev)
        Flat bg rate
    cut_efficiency: Union[int, float] = 0.75
    detection_efficiency: Union[int, float] = 0.85
    detector_name: str = 'SuperCDMS_iZIP_Ge_NR'
    energy_threshold_kev: Union[int, float] = 0.272
    exposure_tonne_year: Union[int, float] = 0.056

```

```
interaction_type: str = 'SI'

resolution(energies_in_kev)
    Flat resolution

target_material: str = 'Ge'

class dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal(n_energy_bins=50, e_min_kev=0,
                                                    e_max_kev=5)

    Bases: _BaseSuperCdms

    background_function(energies_in_kev)
        Flat bg rate

    cut_efficiency: Union[int, float] = 0.75

    detection_efficiency: Union[int, float] = 0.675

    detector_name: str = 'SuperCDMS_iZIP_Si_Migdal'

    property energy_threshold_kev

    exposure_tonne_year: Union[int, float] = 0.0048

    interaction_type: str = 'migdal_SI'

    resolution(energies_in_kev)
        Flat resolution

    target_material: str = 'Si'

class dddm.detectors.super_cdms.SuperCdmsIzipSiNr(n_energy_bins=50, e_min_kev=0, e_max_kev=5)

    Bases: _BaseSuperCdms

    background_function(energies_in_kev)
        Flat bg rate

    cut_efficiency: Union[int, float] = 0.75

    detection_efficiency: Union[int, float] = 0.85

    detector_name: str = 'SuperCDMS_iZIP_Si_NR'

    energy_threshold_kev: Union[int, float] = 0.166

    exposure_tonne_year: Union[int, float] = 0.0048

    interaction_type: str = 'SI'

    resolution(energies_in_kev)
        Flat resolution

    target_material: str = 'Si'
```

dddm.detectors.xenon_nt module

```
class dddm.detectors.xenon_nt.XenonNtMigdal(n_energy_bins=50, e_min_kev=0, e_max_kev=5)
```

Bases: `_BaseXenonNt`

background_function(*energies_in_kev*)

Returns

ER background for Xe detector in events/keV/t/yr

cut_efficiency: `Union[int, float]` = 0.82

detection_efficiency: `Union[int, float]` = 1

detector_name: `str` = 'XENONnT_Migdal'

energy_threshold_kev: `Union[int, float]` = 1

interaction_type: `str` = 'migdal_SI'

resolution(*energies_in_kev*)

Assume the same as the 1T resolution

```
class dddm.detectors.xenon_nt.XenonNtNr(n_energy_bins=50, e_min_kev=0, e_max_kev=5)
```

Bases: `_BaseXenonNt`

background_function(*energies_in_kev*)

Returns

NR background for Xe detector in events/keV/t/yr

cut_efficiency: `Union[int, float]` = 0.83

detection_efficiency: `Union[int, float]` = 1

detector_name: `str` = 'XENONnT_NR'

energy_threshold_kev: `Union[int, float]` = 1.6

interaction_type: `str` = 'SI'

resolution(*energies_in_kev*)

Use `_get_nr_resolution` to calculate the energy resolution.

Parameters

energies_in_kev – NR energies to evaluate the resolution function at

Returns

Module contents

dddm.plotting package

Submodules

dddm.plotting.confidence_figures module

Utility for opening and displaying results from multinest optimization

```

class dddm.plotting.confidence_figures.DDDMResult(path, sampler='multinest')
    Bases: object
    Parse results from fitting from nested sampling
    config_summary(get_props=('detector', 'mass', 'sigma', 'nlive', 'halo_model', 'notes', 'n_parameters')) →
        DataFrame

    property detector

    get_from_config(to_get: str, if_not_available=None)

    get_samples()

    property halo_model

    property mass

    property n_parameters

    property nlive

    property notes

    result: dict = None

    result_summary() → DataFrame

    setup()

    property sigma

    summary() → DataFrame

class dddm.plotting.confidence_figures.ResultsManager(pattern=None, sampler='multinest')
    Bases: object
    add_result(path: str)

    apply_mask(mask)

    build_df()

    property df
        Lazy alias

    register_pattern(pattern, show_tqdm=True)

    result_cache: list = None

    result_df: DataFrame = None

class dddm.plotting.confidence_figures.SeabornPlot(result: DDDMResult)
    Bases: object
    best_fit() → tuple

    plot_bench(c='cyan', **kwargs)

    plot_best_fit(**kwargs) → None

```

```

plot_kde(**kwargs)

plot_samples(**kwargs) → None

plot_sigma_contours(nsigma=2, **kwargs)

property samples: ndarray

samples_to_df() → DataFrame

```

dddm.plotting.plot_basics module

Some basic functions for plotting et cetera. Used to for instance to check that the likelihood function is well behaved

```

dddm.plotting.plot_basics.error_bar_hist(ax, data, data_range=None, nbins=50, **kwargs)

dddm.plotting.plot_basics.get_color_from_range(val, _range=(0, 1), it=0)

dddm.plotting.plot_basics.hist_data(data, data_range=None, nbins=50)

dddm.plotting.plot_basics.ll_element_wise(x, y, clip_val=-10000.0)

dddm.plotting.plot_basics.open_pickle_figure(name)

dddm.plotting.plot_basics.pickle_dump_figure(name)

dddm.plotting.plot_basics.plot_spectrum(data, color='blue', label='label', linestyle='none',
                                         plot_error=True)

dddm.plotting.plot_basics.plt_ll_mass_det(det_class=<class 'dddm.detectors.examples.XenonSimple'>,
                                           bins=10, m=50, sig=1e-45)

dddm.plotting.plot_basics.plt_ll_mass_spec(det_class=<class 'dddm.detectors.examples.XenonSimple'>,
                                           bins=10, m=50, sig=1e-45)

dddm.plotting.plot_basics.plt_ll_sigma_det(det_class=<class 'dddm.detectors.examples.XenonSimple'>,
                                           bins=10, m=50, sig=1e-45)

dddm.plotting.plot_basics.plt_ll_sigma_mass(spec_clas, vary, det_class=<class
                                           'dddm.detectors.examples.XenonSimple'>, bins=10, m=50,
                                           sig=1e-45)

dddm.plotting.plot_basics.plt_ll_sigma_spec(det_class=<class
                                           'dddm.detectors.examples.XenonSimple'>, bins=10, m=50,
                                           sig=1e-45)

dddm.plotting.plot_basics.plt_priors(itot=100)

dddm.plotting.plot_basics.save_canvas(name, save_dir='./figures', dpi=200, tight_layout=False,
                                       pickle_dump=True)

    Wrapper for saving current figure

dddm.plotting.plot_basics.show_ll_function(npoints=10000.0, clip_val=-10000.0, min_val=0.1)

dddm.plotting.plot_basics.simple_hist(y: ndarray)

```

dddm.plotting.seaborn_utils module

” Small script to extract the results from seaborn to calculate confidence intervals

I’m sorry for this script, I wanted to have something robust but I couldn’t find it anywhere. Seaborn is doing a great job, so let’s use it’s functionality.

This work is mostly based on: <https://github.com/mwaskom/seaborn/blob/ff0fc76b4b65c7bcc1d2be2244e4ca1a92e4e740/seaborn/distributions.py>

```
dddm.plotting.seaborn_utils.one_sigma_area(x, y, clf=True, **kwargs)
```

Module contents

dddm.recoil_rates package

Submodules

dddm.recoil_rates.detector_spectrum module

Introduce detector effects into the expected detection spectrum

```
class dddm.recoil_rates.detector_spectrum.DetectorSpectrum(dark_matter_model: Union[SHM,
                                                                                   ShieldedSHM], experiment:
                                                                                   Experiment)
```

Bases: *GenSpectrum*

Convolve a recoil spectrum with the detector effects:

- background levels
- energy resolution
- energy threshold

```
static above_threshold(rates: ndarray, e_bin_edges: ndarray, e_thr: Union[float, int])
```

Apply threshold to the rates. We are right edge inclusive bin edges : **|bin0|bin1|bin2|** e_thr : | bin0 -> 0
bin1 -> fraction of bin1 > e_thr bin2 -> full content

Parameters

- **rates** – bins with the number of counts
- **e_bin_edges** – 2d array of the left, right bins
- **e_thr** – energy threshold

Returns

rates with energy threshold applied

dddm.recoil_rates.halo module

For a given detector get a WIMPrate for a given detector (not taking into account any detector effects)

class dddm.recoil_rates.halo.**SHM**(*v_0=None, v_esc=None, rho_dm=None*)

Bases: `object`

class used to pass a halo model to the rate computation must contain: :param v_esc – escape velocity (multiplied by units) :param rho_dm – density in mass/volume of dark matter at the Earth (multiplied by units) The standard halo model also allows variation of v_0 :param v_0 – v0 of the velocity distribution (multiplied by units) :function velocity_dist – function taking v,t giving normalised velocity distribution in earth rest-frame.

parameter_dict()

Return a dict of readable parameters of the current settings

velocity_dist(*v, t*)

Get the velocity distribution in units of per velocity, :param v: v is in units of velocity :return: observed velocity distribution at earth

dddm.recoil_rates.halo_shielded module

class dddm.recoil_rates.halo_shielded.**ShieldedSHM**(*location, file_folder='./verne_files', v_0=None, v_esc=None, rho_dm=None, log_cross_section=None, log_mass=None*)

Bases: `object`

class used to pass a halo model to the rate computation based on the earth shielding effect as calculated by Verne must contain:

:param v_esc – escape velocity (multiplied by units) :param rho_dm – density in mass/volume of dark matter at the Earth (multiplied by units)

The standard halo model also allows variation of v_0

:param v_0 – v0 of the velocity distribution (multiplied by units) :function velocity_dist – function taking v,t giving normalised

velocity distribution in earth rest-frame.

load_f()

load the velocity distribution. If there is no velocity
distribution shaved, load one.

Returns

parameter_dict()

Return a dict of readable parameters of the current settings

property rho_dm_nodim

property v_0_nodim

property v_esc_nodim

velocity_dist(*v, t*)

Get the velocity distribution in units of per velocity, :param v: v is in units of velocity :return: observed velocity distribution at earth

dddm.recoil_rates.spectrum module

class dddm.recoil_rates.spectrum.**GenSpectrum**(*dark_matter_model*: Union[SHM, ShieldedSHM],
experiment: Experiment)

Bases: `object`

property `darkelf_class`

get_bin_edges()

get_counts(*wimp_mass*: Union[int, float], *cross_section*: Union[int, float], *poisson*=False) → array

Parameters

- **wimp_mass** – wimp mass (not log)
- **cross_section** – cross-section of the wimp nucleon interaction (not log)
- **poisson** – type bool, add poisson True or False

Returns

array of counts/bin

get_data(*wimp_mass*: Union[int, float], *cross_section*: Union[int, float], *poisson*=False,
return_counts=False) → Union[DataFrame, ndarray]

Parameters

- **wimp_mass** – wimp mass (not log)
- **cross_section** – cross-section of the wimp nucleon interaction (not log)
- **poisson** – type bool, add poisson True or False
- **return_counts** – instead of a dataframe, return counts only

Returns

pd.DataFrame containing events binned in energy

required_detector_fields = ['name', 'material', 'type', 'exp_eff']

set_negative_to_zero(*counts*: ndarray)

spectrum_simple(*energy_bins*: Union[list, tuple, ndarray], *wimp_mass*: Union[int, float], *cross_section*:
Union[int, float])

Compute the spectrum for a given mass and cross-section :param wimp_mass: wimp mass (not log) :param
cross_section: cross-section of the wimp nucleon interaction

(not log)

Returns

returns the rate

Module contents

dddm.samplers package

Submodules

dddm.samplers.emcee module

Do a likelihood fit. The class MCMCStatModel is used for fitting applying the MCMC algorithm emcee.

MCMC is:

slower than the nestle package; and harder to use since one has to choose the ‘right’ initial parameters

Nevertheless, the walkers give great insight in how the likelihood-function is felt by the steps that the walkers make

```
class dddm.samplers.emcee.MCMCStatModel(wimp_mass: Union[float, int], cross_section: Union[float, int],
                                         spectrum_class: Union[DetectorSpectrum, GenSpectrum],
                                         prior: dict, tmp_folder: str, fit_parameters=('log_mass',
                                         'log_cross_section', 'v_0', 'v_esc', 'density', 'k'),
                                         detector_name=None, verbose=False, notes='default',
                                         nwalkers=50, nsteps=100, remove_frac=0.2, emcee_thin=15)
```

Bases: *StatModel*

run()

save_results(save_to_dir='emcee', force_index=False)

set_sampler(mult=True)

init the MCMC sampler

show_corner()

show_walkers()

dddm.samplers.multi_detectors module

```
class dddm.samplers.multi_detectors.CombinedMultinest(wimp_mass: Union[float, int], cross_section:
                                                         Union[float, int], spectrum_class:
                                                         List[Union[DetectorSpectrum,
                                                         GenSpectrum]]), prior: dict, tmp_folder: str,
                                                         results_dir: Optional[str] = None,
                                                         fit_parameters=('log_mass',
                                                         'log_cross_section', 'v_0', 'v_esc', 'density',
                                                         'k'), detector_name=None, verbose=False,
                                                         notes='default', nlive=1024, tol=0.1)
```

Bases: *_CombinedInference, MultiNestSampler*

```
class dddm.samplers.multi_detectors.CombinedNestle(wimp_mass: Union[float, int], cross_section:
                                                         Union[float, int], spectrum_class:
                                                         List[Union[DetectorSpectrum, GenSpectrum]]),
                                                         prior: dict, tmp_folder: str, results_dir:
                                                         Optional[str] = None,
                                                         fit_parameters=('log_mass', 'log_cross_section',
                                                         'v_0', 'v_esc', 'density', 'k'), detector_name=None,
                                                         verbose=False, notes='default', nlive=1024,
                                                         tol=0.1)
```

Bases: `_CombinedInference`, `NestleSampler`

```
class dddm.samplers.multi_detectors.CombinedUltraNest(wimp_mass: Union[float, int], cross_section:
    Union[float, int], spectrum_class:
    List[Union[DetectorSpectrum,
    GenSpectrum]], prior: dict, tmp_folder: str,
    results_dir: Optional[str] = None,
    fit_parameters=('log_mass',
    'log_cross_section', 'v_0', 'v_esc', 'density',
    'k'), detector_name=None, verbose=False,
    notes='default', nlive=1024, tol=0.1)
```

Bases: `_CombinedInference`, `UltraNestSampler`

dddm.samplers.nestle module

```
class dddm.samplers.nestle.NestleSampler(wimp_mass: Union[float, int], cross_section: Union[float, int],
    spectrum_class: Union[DetectorSpectrum, GenSpectrum],
    prior: dict, tmp_folder: str, results_dir: Optional[str] = None,
    fit_parameters=('log_mass', 'log_cross_section', 'v_0', 'v_esc',
    'density', 'k'), detector_name=None, verbose=False,
    notes='default', nlive=1024, tol=0.1)
```

Bases: `MultiNestSampler`

`get_summary()`

`run()`

`save_results(force_index=False)`

`show_corner()`

dddm.samplers.pymultinest module

Do a likelihood fit. The class `NestedSamplerStatModel` is used for fitting applying the bayesian algorithm `nestle/multinest`

```
class dddm.samplers.pymultinest.MultiNestSampler(wimp_mass: Union[float, int], cross_section:
    Union[float, int], spectrum_class:
    Union[DetectorSpectrum, GenSpectrum], prior:
    dict, tmp_folder: str, results_dir: Optional[str] =
    None, fit_parameters=('log_mass',
    'log_cross_section', 'v_0', 'v_esc', 'density', 'k'),
    detector_name=None, verbose=False,
    notes='default', nlive=1024, tol=0.1)
```

Bases: `StatModel`

`check_did_run()`

`check_did_save()`

`get_save_dir(force_index=False, _hash=None) → str`

```
get_summary()
```

```
log_prior_transform_nested(x, x_name)
```

```
log_probability_nested(parameter_vals, parameter_names)
```

Parameters

parameter_vals – the values of the model/benchmark considered as the truth

:param parameter_values: the values of the parameters that are being varied :param parameter_names: the names of the parameter_values :return:

```
run()
```

```
save_results(force_index=False)
```

```
show_corner()
```

Module contents

1.2.2 Submodules

1.2.3 dddm.context module

Setup the file structure for the software. Specifies several folders: software_dir: path of installation

```
dddm.context.base_context()
```

1.2.4 dddm.priors module

```
dddm.priors.get_priors(priors_from='Evans_2019')
```

Returns

dictionary of priors, type and values

1.2.5 dddm.statistics module

Statistical model giving likelihoods for detecting a spectrum given a benchmark to compare it with.

```
class dddm.statistics.StatModel(wimp_mass: Union[float, int], cross_section: Union[float, int],
                                spectrum_class: Union[DetectorSpectrum, GenSpectrum], prior: dict,
                                tmp_folder: str, fit_parameters=('log_mass', 'log_cross_section', 'v_0',
                                                                'v_esc', 'density', 'k'), detector_name=None, verbose=False,
                                notes='default')
```

Bases: `object`

```
allow_multiple_detectors = False
```

```
property bench_is_set
```

```
benchmark_values = None
```

```
check_spectrum()
```

Lazy alias for eval_spectrum

property density: `Union[int, float]`

`eval_benchmark()`

`eval_spectrum(values: Union[list, tuple, ndarray], parameter_names: Union[List[str], Tuple[str]])`

For given values and parameter names, return the spectrum one would have with these parameters. The values and parameter names should be array like objects of the same length. Usually, one fits either two ('log_mass', 'log_cross_section') or five parameters ('log_mass', 'log_cross_section', 'v_0', 'v_esc', 'density'). :param values: array like object of :param parameter_names: names of parameters :return: a spectrum as specified by the parameter_names

`get_logger(tmp_folder, verbosity)`

`known_parameters = ('log_mass', 'log_cross_section', 'v_0', 'v_esc', 'density')`

property log_cross_section

property log_mass

`log_prior(value, variable_name)`

Compute the prior of variable_name for a given value :param value: value of variable name :param variable_name: name of the 'value'. This name should be in the config of the class under the priors with a similar content as the priors as specified in the get_prior function. :return: prior of value

`log_probability(parameter_vals, parameter_names)`

Parameters

- **parameter_vals** – the values of the model/benchmark considered as the truth
- **parameter_names** – the names of the parameter_values

Returns

`read_priors_mean(prior_name) → Union[int, float]`

`set_benchmark()`

Set up the benchmark used in this statistical model. Likelihood of other models can be evaluated for this 'truth'

`set_fit_parameters(params)`

Write the fit parameters to the config

`set_models()`

Update the dm model with with the required settings from the prior

`total_log_prior(parameter_vals, parameter_names)`

For each of the parameter names, read the prior

Parameters

- **parameter_vals** – the values of the model/benchmark considered as the truth
- **parameter_names** – the names of the parameter_values

Returns

property v_0: `Union[int, float]`

property v_esc: `Union[int, float]`

1.2.6 dddm.test_utils module

`dddm.test_utils.test_context()`

just returns the base contexts, might be different one day

1.2.7 dddm.utils module

Basic functions for saving et cetera

`dddm.utils.deterministic_hash(thing, length=10)`

Return a base32 lowercase string of length determined from hashing a container hierarchy

`dddm.utils.exporter(export_self=False)`

Export utility modified from <https://stackoverflow.com/a/41895194> Returns export decorator, `__all__` list stolen from <https://github.com/AxFoundation/strax/blob/d3608efc77acd52e1d5a208c3092b6b45b27a6e2/strax/utils.py#46>

`dddm.utils.is_installed(module)`

Try to import <module>, return False if not installed

`dddm.utils.is_windows()`

`dddm.utils.print_versions(modules=('dddm', 'numpy', 'numba', 'wimprates'), print_output=True, include_python=True, return_string=False, include_git=True)`

Print versions of modules installed.

Parameters

- **modules** – Modules to print, should be str, tuple or list. E.g. `print_versions(modules=('numpy', 'dddm',))`
- **return_string** – optional. Instead of printing the message, return a string
- **include_git** – Include the current branch and latest commit hash

Returns

optional, the message that would have been printed

`dddm.utils.to_str_tuple(x: Union[str, bytes, list, tuple, Series, ndarray]) → Tuple[str]`

Convert any sensible instance to a tuple of strings stolen from <https://github.com/AxFoundation/strax/blob/d3608efc77acd52e1d5a208c3092b6b45b27a6e2/strax/utils.py#242>

1.2.8 Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

d

- `ddd`, 19
- `ddd.context`, 17
- `ddd.detectors`, 9
 - `ddd.detectors.examples`, 3
 - `ddd.detectors.experiment`, 5
 - `ddd.detectors.super_cdms`, 5
 - `ddd.detectors.xenon_nt`, 9
- `ddd.plotting`, 12
 - `ddd.plotting.confidence_figures`, 9
 - `ddd.plotting.plot_basics`, 11
 - `ddd.plotting.seaborn_utils`, 12
- `ddd.priors`, 17
- `ddd.recoil_rates`, 15
 - `ddd.recoil_rates.detector_spectrum`, 12
 - `ddd.recoil_rates.halo`, 13
 - `ddd.recoil_rates.halo_shielded`, 13
 - `ddd.recoil_rates.spectrum`, 14
- `ddd.samplers`, 17
 - `ddd.samplers.emcee`, 15
 - `ddd.samplers.multi_detectors`, 15
 - `ddd.samplers.nestle`, 16
 - `ddd.samplers.pymultinest`, 16
- `ddd.statistics`, 17
- `ddd.test_utils`, 19
- `ddd.utils`, 19

INDEX

A

`above_threshold()` (`dddm.recoil_rates.detector_spectrum.DetectorSpectrum` static method), 12

`add_result()` (`dddm.plotting.confidence_figures.ResultsManager` method), 10

`allow_multiple_detectors` (`dddm.statistics.StatModel` attribute), 17

`apply_mask()` (`dddm.plotting.confidence_figures.ResultsManager` method), 10

`ArgonSimple` (class in `dddm.detectors.examples`), 3

B

`background_function()` (`dddm.detectors.examples.ArgonSimple` method), 3

`background_function()` (`dddm.detectors.examples.GermaniumSimple` method), 4

`background_function()` (`dddm.detectors.examples.XenonSimple` method), 4

`background_function()` (`dddm.detectors.experiment.Experiment` method), 5

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsHvGeMigdal` method), 5

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsHvGeNr` method), 6

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsHvSiMigdal` method), 6

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsHvSiNr` method), 6

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal` method), 7

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsIzipGeNr` method), 7

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal` method), 8

`background_function()` (`dddm.detectors.super_cdms.SuperCdmsIzipSiNr` method), 8

`background_function()` (`dddm.detectors.xenon_nt.XenonNtMigdal` method), 9

`background_function()` (`dddm.detectors.xenon_nt.XenonNtNr` method), 9

`base_context()` (in module `dddm.context`), 17

`bench_is_set` (`dddm.statistics.StatModel` property), 17

`benchmark_values` (`dddm.statistics.StatModel` attribute), 17

`best_fit()` (`dddm.plotting.confidence_figures.SeabornPlot` method), 10

`build_df()` (`dddm.plotting.confidence_figures.ResultsManager` method), 10

C

`check.did_run()` (`dddm.samplers.pymultinest.MultiNestSampler` method), 16

`check.did_save()` (`dddm.samplers.pymultinest.MultiNestSampler` method), 16

`check_spectrum()` (`dddm.statistics.StatModel` method), 17

`CombinedMultinest` (class in `dddm.samplers.multi_detectors`), 15

`CombinedNestle` (class in `dddm.samplers.multi_detectors`), 15

`CombinedUltraNest` (class in `dddm.samplers.multi_detectors`), 16

`config` (`dddm.detectors.experiment.Experiment` property), 5

`config_summary()` (`dddm.plotting.confidence_figures.DDDMResult` method), 10

`cut_efficiency` (`dddm.detectors.examples.ArgonSimple` attribute), 3

`cut_efficiency` (`dddm.detectors.examples.GermaniumSimple` attribute), 4

cut_efficiency(<i>dddm.detectors.examples.XenonSimple</i> <i>attribute</i>), 4	<i>dddm.recoil_rates.detector_spectrum</i> module, 12
cut_efficiency(<i>dddm.detectors.experiment.Experiment</i> <i>attribute</i>), 5	<i>dddm.recoil_rates.halo</i> module, 13
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvGeMigdal</i> <i>attribute</i>), 5	<i>dddm.recoil_rates.halo_shielded</i> module, 13
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvGeNr</i> <i>attribute</i>), 6	<i>dddm.recoil_rates.spectrum</i> module, 14
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvSiMigdal</i> <i>attribute</i>), 6	<i>dddm.samplers</i> module, 17
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvSiNr</i> <i>attribute</i>), 7	<i>dddm.samplers.emcee</i> module, 15
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvSiNr</i> <i>attribute</i>), 7	<i>dddm.samplers.multi_detectors</i> module, 15
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvSiNr</i> <i>attribute</i>), 7	<i>dddm.samplers.nestle</i> module, 16
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvSiNr</i> <i>attribute</i>), 8	<i>dddm.samplers.pymultinest</i> module, 16
cut_efficiency(<i>dddm.detectors.super_cdms.SuperCdmsHvSiNr</i> <i>attribute</i>), 8	<i>dddm.statistics</i> module, 17
cut_efficiency(<i>dddm.detectors.xenon_nt.XenonNtMigdal</i> <i>attribute</i>), 9	<i>dddm.test_utils</i> module, 19
cut_efficiency(<i>dddm.detectors.xenon_nt.XenonNtNr</i> <i>attribute</i>), 9	<i>dddm.utils</i> module, 19
	<i>DDDMResult</i> (class in <i>dddm.plotting.confidence_figures</i>), 9
D	
darkelf_class(<i>dddm.recoil_rates.spectrum.GenSpectrum</i> property), 14	<i>density</i> (<i>dddm.statistics.StatModel</i> property), 17
<i>dddm</i> module, 19	<i>detection_efficiency</i> (<i>dddm.detectors.examples.ArgonSimple</i> attribute), 3
<i>dddm.context</i> module, 17	<i>detection_efficiency</i> (<i>dddm.detectors.examples.GermaniumSimple</i> <i>attribute</i>), 4
<i>dddm.detectors</i> module, 9	<i>detection_efficiency</i> (<i>dddm.detectors.examples.XenonSimple</i> attribute), 4
<i>dddm.detectors.examples</i> module, 3	<i>detection_efficiency</i> (<i>dddm.detectors.experiment.Experiment</i> attribute), 5
<i>dddm.detectors.experiment</i> module, 5	<i>detection_efficiency</i> (<i>dddm.detectors.super_cdms.SuperCdmsHvGeMigdal</i> <i>attribute</i>), 5
<i>dddm.detectors.super_cdms</i> module, 5	<i>detection_efficiency</i> (<i>dddm.detectors.super_cdms.SuperCdmsHvGeNr</i> <i>attribute</i>), 6
<i>dddm.detectors.xenon_nt</i> module, 9	<i>detection_efficiency</i> (<i>dddm.detectors.super_cdms.SuperCdmsHvSiMigdal</i> <i>attribute</i>), 6
<i>dddm.plotting</i> module, 12	<i>detection_efficiency</i> (<i>dddm.detectors.super_cdms.SuperCdmsHvSiNr</i> <i>attribute</i>), 7
<i>dddm.plotting.confidence_figures</i> module, 9	<i>detection_efficiency</i> (<i>dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal</i> <i>attribute</i>), 7
<i>dddm.plotting.plot_basics</i> module, 11	
<i>dddm.plotting.seaborn_utils</i> module, 12	
<i>dddm.priors</i> module, 17	
<i>dddm.recoil_rates</i> module, 15	

detection_efficiency (dddm.detectors.super_cdms.SuperCdmsIzipGeNr attribute), 7
 detection_efficiency (dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal attribute), 8
 detection_efficiency (dddm.detectors.super_cdms.SuperCdmsIzipSiNr attribute), 8
 detection_efficiency (dddm.detectors.xenon_nt.XenonNtMigdal attribute), 9
 detection_efficiency (dddm.detectors.xenon_nt.XenonNtNr attribute), 9
 detector (dddm.plotting.confidence_figures.DDDMResult property), 10
 detector_hash (dddm.detectors.experiment.Experiment attribute), 5
 detector_name (dddm.detectors.examples.ArgonSimple attribute), 3
 detector_name (dddm.detectors.examples.GermaniumSimple attribute), 4
 detector_name (dddm.detectors.examples.XenonSimple attribute), 4
 detector_name (dddm.detectors.experiment.Experiment attribute), 5
 detector_name (dddm.detectors.super_cdms.SuperCdmsHvGeMigdal attribute), 5
 detector_name (dddm.detectors.super_cdms.SuperCdmsHvGeNr attribute), 6
 detector_name (dddm.detectors.super_cdms.SuperCdmsHvSiMigdal attribute), 6
 detector_name (dddm.detectors.super_cdms.SuperCdmsHvSiNr attribute), 7
 detector_name (dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal attribute), 7
 detector_name (dddm.detectors.super_cdms.SuperCdmsIzipGeNr attribute), 7
 detector_name (dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal attribute), 8
 detector_name (dddm.detectors.super_cdms.SuperCdmsIzipSiNr attribute), 8
 detector_name (dddm.detectors.xenon_nt.XenonNtMigdal attribute), 9
 detector_name (dddm.detectors.xenon_nt.XenonNtNr attribute), 9
 DetectorSpectrum (class in dddm.recoil_rates.detector_spectrum), 12
 deterministic_hash() (in module dddm.utils), 19
 df (dddm.plotting.confidence_figures.ResultsManager property), 10
 E
 e_max_kev (dddm.detectors.experiment.Experiment attribute), 5
 e_min_kev (dddm.detectors.experiment.Experiment attribute), 5
 effective_exposure (dddm.detectors.experiment.Experiment property), 5
 energy_threshold_kev (dddm.detectors.examples.ArgonSimple attribute), 3
 energy_threshold_kev (dddm.detectors.examples.GermaniumSimple attribute), 4
 energy_threshold_kev (dddm.detectors.examples.XenonSimple attribute), 4
 energy_threshold_kev (dddm.detectors.experiment.Experiment attribute), 5
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsHvGeMigdal property), 5
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsHvGeNr attribute), 6
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsHvSiMigdal property), 6
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsHvSiNr attribute), 7
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal property), 7
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsIzipGeNr attribute), 7
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal property), 8
 energy_threshold_kev (dddm.detectors.super_cdms.SuperCdmsIzipSiNr attribute), 8
 energy_threshold_kev (dddm.detectors.xenon_nt.XenonNtMigdal attribute), 9
 energy_threshold_kev (dddm.detectors.xenon_nt.XenonNtNr attribute), 9
 error_bar_hist() (in module dddm.plotting.plot_basics), 11
 eval_benchmark() (dddm.statistics.StatModel method), 18
 eval_spectrum() (dddm.statistics.StatModel method),

18

`Experiment` (class in `dddm.detectors.experiment`), 5

`exporter()` (in module `dddm.utils`), 19

`exposure_tonne_year` (`dddm.detectors.examples.ArgonSimple` attribute), 3

`exposure_tonne_year` (`dddm.detectors.examples.GermaniumSimple` attribute), 4

`exposure_tonne_year` (`dddm.detectors.examples.XenonSimple` attribute), 4

`exposure_tonne_year` (`dddm.detectors.experiment.Experiment` attribute), 5

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsHvGeMigdal` attribute), 5

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsHvGeNr` attribute), 6

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsHvSiMigdal` attribute), 6

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsHvSiNr` attribute), 7

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal` attribute), 7

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsIzipGeNr` attribute), 7

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal` attribute), 8

`exposure_tonne_year` (`dddm.detectors.super_cdms.SuperCdmsIzipSiNr` attribute), 8

G

`GenSpectrum` (class in `dddm.recoil_rates.spectrum`), 14

`GermaniumSimple` (class in `dddm.detectors.examples`), 4

`get_bin_edges()` (`dddm.recoil_rates.spectrum.GenSpectrum` method), 14

`get_color_from_range()` (in module `dddm.plotting.plot_basics`), 11

`get_counts()` (`dddm.recoil_rates.spectrum.GenSpectrum` method), 14

`get_data()` (`dddm.recoil_rates.spectrum.GenSpectrum` method), 14

`get_from_config()` (`dddm.plotting.confidence_figures.DDDMResult` method), 10

`get_logger()` (`dddm.statistics.StatModel` method), 18

`get_priors()` (in module `dddm.priors`), 17

`get_samples()` (`dddm.plotting.confidence_figures.DDDMResult` method), 10

`get_save_dir()` (`dddm.samplers.pymultinest.MultiNestSampler` method), 16

`get_summary()` (`dddm.samplers.nestle.NestleSampler` method), 16

`get_summary()` (`dddm.samplers.pymultinest.MultiNestSampler` method), 16

H

`halo_model` (`dddm.plotting.confidence_figures.DDDMResult` property), 10

`hist_data()` (in module `dddm.plotting.plot_basics`), 11

I

`interaction_type` (`dddm.detectors.examples.ArgonSimple` attribute), 3

`interaction_type` (`dddm.detectors.examples.GermaniumSimple` attribute), 4

`interaction_type` (`dddm.detectors.examples.XenonSimple` attribute), 4

`interaction_type` (`dddm.detectors.experiment.Experiment` attribute), 5

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsHvGeMigdal` attribute), 6

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsHvGeNr` attribute), 6

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsHvSiMigdal` attribute), 6

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsHvSiNr` attribute), 7

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal` attribute), 7

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsIzipGeNr` attribute), 7

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal` attribute), 8

`interaction_type` (`dddm.detectors.super_cdms.SuperCdmsIzipSiNr` attribute), 8

`interaction_type` (`dddm.detectors.xenon_nt.XenonNtMigdal` attribute), 9

`interaction_type` (`dddm.detectors.xenon_nt.XenonNtNr` attribute), 9

`is_installed()` (in module `dddm.utils`), 19

`is_windows()` (in module `dddm.utils`), 19

K

`known_parameters` (`dddm.statistics.StatModel` attribute), 18

L

`ll_element_wise()` (in module `dddm.plotting.plot_basics`), 11

load_f() (*dddm.recoil_rates.halo_shielded.ShieldedSHM* *MultiNestSampler* (class in *dddm.samplers.pymultinest*), 13
 location (*dddm.detectors.examples.ArgonSimple* *attribute*), 3
 location (*dddm.detectors.examples.GermaniumSimple* *attribute*), 4
 location (*dddm.detectors.examples.XenonSimple* *attribute*), 4
 location (*dddm.detectors.experiment.Experiment* *attribute*), 5
 log_cross_section (*dddm.statistics.StatModel* *property*), 18
 log_mass (*dddm.statistics.StatModel* *property*), 18
 log_prior() (*dddm.statistics.StatModel* *method*), 18
 log_prior_transform_nested() (*dddm.samplers.pymultinest.MultiNestSampler* *method*), 17
 log_probability() (*dddm.statistics.StatModel* *method*), 18
 log_probability_nested() (*dddm.samplers.pymultinest.MultiNestSampler* *method*), 17

M

mass (*dddm.plotting.confidence_figures.DDDMResult* *property*), 10
 MCMCStatModel (class in *dddm.samplers.emcee*), 15
 module
 dddm, 19
 dddm.context, 17
 dddm.detectors, 9
 dddm.detectors.examples, 3
 dddm.detectors.experiment, 5
 dddm.detectors.super_cdms, 5
 dddm.detectors.xenon_nt, 9
 dddm.plotting, 12
 dddm.plotting.confidence_figures, 9
 dddm.plotting.plot_basics, 11
 dddm.plotting.seaborn_utils, 12
 dddm.priors, 17
 dddm.recoil_rates, 15
 dddm.recoil_rates.detector_spectrum, 12
 dddm.recoil_rates.halo, 13
 dddm.recoil_rates.halo_shielded, 13
 dddm.recoil_rates.spectrum, 14
 dddm.samplers, 17
 dddm.samplers.emcee, 15
 dddm.samplers.multi_detectors, 15
 dddm.samplers.nestle, 16
 dddm.samplers.pymultinest, 16
 dddm.statistics, 17
 dddm.test_utils, 19
 dddm.utils, 19

N

n_energy_bins (*dddm.detectors.experiment.Experiment* *attribute*), 5
 n_parameters (*dddm.plotting.confidence_figures.DDDMResult* *property*), 10
 NestleSampler (class in *dddm.samplers.nestle*), 16
 nlive (*dddm.plotting.confidence_figures.DDDMResult* *property*), 10
 notes (*dddm.plotting.confidence_figures.DDDMResult* *property*), 10

O

one_sigma_area() (in module *dddm.plotting.seaborn_utils*), 12
 open_pickle_figure() (in module *dddm.plotting.plot_basics*), 11

P

parameter_dict() (*dddm.recoil_rates.halo.SHM* *method*), 13
 parameter_dict() (*dddm.recoil_rates.halo_shielded.ShieldedSHM* *method*), 13
 pickle_dump_figure() (in module *dddm.plotting.plot_basics*), 11
 plot_bench() (*dddm.plotting.confidence_figures.SeabornPlot* *method*), 10
 plot_best_fit() (*dddm.plotting.confidence_figures.SeabornPlot* *method*), 10
 plot_kde() (*dddm.plotting.confidence_figures.SeabornPlot* *method*), 10
 plot_samples() (*dddm.plotting.confidence_figures.SeabornPlot* *method*), 11
 plot_sigma_contours() (*dddm.plotting.confidence_figures.SeabornPlot* *method*), 11
 plot_spectrum() (in module *dddm.plotting.plot_basics*), 11
 plt_ll_mass_det() (in module *dddm.plotting.plot_basics*), 11
 plt_ll_mass_spec() (in module *dddm.plotting.plot_basics*), 11
 plt_ll_sigma_det() (in module *dddm.plotting.plot_basics*), 11
 plt_ll_sigma_mass() (in module *dddm.plotting.plot_basics*), 11
 plt_ll_sigma_spec() (in module *dddm.plotting.plot_basics*), 11
 plt_priors() (in module *dddm.plotting.plot_basics*), 11
 print_versions() (in module *dddm.utils*), 19

R

`read_priors_mean()` (`dddm.statistics.StatModel` method), 18

`register_pattern()` (`dddm.plotting.confidence_figures.ResultsManager` method), 10

`required_detector_fields` (`dddm.recoil_rates.spectrum.GenSpectrum` attribute), 14

`resolution()` (`dddm.detectors.examples.ArgonSimple` method), 3

`resolution()` (`dddm.detectors.examples.GermaniumSimple` method), 4

`resolution()` (`dddm.detectors.examples.XenonSimple` method), 4

`resolution()` (`dddm.detectors.experiment.Experiment` method), 5

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsHvGeMigdal` method), 6

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsHvGeNr` method), 6

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsHvSiMigdal` method), 6

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsHvSiNr` method), 7

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal` method), 7

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsIzipGeNr` method), 8

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal` method), 8

`resolution()` (`dddm.detectors.super_cdms.SuperCdmsIzipSiNr` method), 8

`resolution()` (`dddm.detectors.xenon_nt.XenonNtMigdal` method), 9

`resolution()` (`dddm.detectors.xenon_nt.XenonNtNr` method), 9

`result` (`dddm.plotting.confidence_figures.DDDMResult` attribute), 10

`result_cache` (`dddm.plotting.confidence_figures.ResultsManager` attribute), 10

`result_df` (`dddm.plotting.confidence_figures.ResultsManager` attribute), 10

`result_summary()` (`dddm.plotting.confidence_figures.DDDMResult` method), 10

`ResultsManager` (class in `dddm.plotting.confidence_figures`), 10

`rho_dm_nodim` (`dddm.recoil_rates.halo_shielded.ShieldedSHM` property), 13

`run()` (`dddm.samplers.emcee.MCMCStatModel` method), 15

`run()` (`dddm.samplers.nestle.NestleSampler` method), 16

`run()` (`dddm.samplers.pymultinest.MultiNestSampler` method), 17

S

`samples` (`dddm.plotting.confidence_figures.SeabornPlot` property), 11

`sample_variance_df()` (`dddm.plotting.confidence_figures.SeabornPlot` method), 11

`save_canvas()` (in module `dddm.plotting.plot_basics`), 11

`save_results()` (`dddm.samplers.emcee.MCMCStatModel` method), 15

`save_results()` (`dddm.samplers.nestle.NestleSampler` method), 16

`save_results()` (`dddm.samplers.pymultinest.MultiNestSampler` method), 17

`SeabornPlot` (class in `dddm.plotting.confidence_figures`), 10

`set_benchmark()` (`dddm.statistics.StatModel` method), 18

`set_fit_parameters()` (`dddm.statistics.StatModel` method), 18

`set_models()` (`dddm.statistics.StatModel` method), 18

`set_negative_to_zero()` (`dddm.recoil_rates.spectrum.GenSpectrum` method), 14

`set_sampler()` (`dddm.samplers.emcee.MCMCStatModel` method), 15

`setup()` (`dddm.plotting.confidence_figures.DDDMResult` method), 10

`ShieldedSHM` (class in `dddm.recoil_rates.halo_shielded`), 13

`SHM` (class in `dddm.recoil_rates.halo`), 13

`show_corner()` (`dddm.samplers.emcee.MCMCStatModel` method), 15

`show_corner()` (`dddm.samplers.nestle.NestleSampler` method), 16

`show_corner()` (`dddm.samplers.pymultinest.MultiNestSampler` method), 17

`show_ll_function()` (in module `dddm.plotting.plot_basics`), 11

`show_walkers()` (`dddm.samplers.emcee.MCMCStatModel` method), 15

`sigma` (`dddm.plotting.confidence_figures.DDDMResult` property), 10

`SimpleHist` (in module `dddm.plotting.plot_basics`), 11

`spectrum_simple()` (`dddm.recoil_rates.spectrum.GenSpectrum` method), 14

`StatModel` (class in `dddm.statistics`), 17

`summary()` (`dddm.plotting.confidence_figures.DDDMResult` method), 10

`SuperCdmsHvGeMigdal` (class in `dddm.detectors.super_cdms`), 5

`SuperCdmsHvGeNr` (class in `dddm.detectors.super_cdms`), 6

`SuperCdmsHvSiMigdal` (class in `dddm.detectors.super_cdms`), 6

<code>dddm.detectors.super_cdms), 6</code>		
<code>SuperCdmsHvSiNr (class</code>	<i>in</i>	X
<code>dddm.detectors.super_cdms), 6</code>		<code>XenonNtMigdal (class in dddm.detectors.xenon_nt), 9</code>
<code>SuperCdmsIzipGeMigdal (class</code>	<i>in</i>	<code>XenonNtNr (class in dddm.detectors.xenon_nt), 9</code>
<code>dddm.detectors.super_cdms), 7</code>		<code>XenonSimple (class in dddm.detectors.examples), 4</code>
<code>SuperCdmsIzipGeNr (class</code>	<i>in</i>	
<code>dddm.detectors.super_cdms), 7</code>		
<code>SuperCdmsIzipSiMigdal (class</code>	<i>in</i>	
<code>dddm.detectors.super_cdms), 8</code>		
<code>SuperCdmsIzipSiNr (class</code>	<i>in</i>	
<code>dddm.detectors.super_cdms), 8</code>		

T

`target_material (dddm.detectors.examples.ArgonSimple attribute), 4`

`target_material (dddm.detectors.examples.GermaniumSimple attribute), 4`

`target_material (dddm.detectors.examples.XenonSimple attribute), 4`

`target_material (dddm.detectors.experiment.Experiment attribute), 5`

`target_material (dddm.detectors.super_cdms.SuperCdmsHvGeMigdal attribute), 6`

`target_material (dddm.detectors.super_cdms.SuperCdmsHvGeNr attribute), 6`

`target_material (dddm.detectors.super_cdms.SuperCdmsHvSiMigdal attribute), 6`

`target_material (dddm.detectors.super_cdms.SuperCdmsHvSiNr attribute), 7`

`target_material (dddm.detectors.super_cdms.SuperCdmsIzipGeMigdal attribute), 7`

`target_material (dddm.detectors.super_cdms.SuperCdmsIzipGeNr attribute), 8`

`target_material (dddm.detectors.super_cdms.SuperCdmsIzipSiMigdal attribute), 8`

`target_material (dddm.detectors.super_cdms.SuperCdmsIzipSiNr attribute), 8`

`test_context() (in module dddm.test_utils), 19`

`to_str_tuple() (in module dddm.utils), 19`

`total_log_prior() (dddm.statistics.StatModel method), 18`

V

`v_0 (dddm.statistics.StatModel property), 18`

`v_0_nodim (dddm.recoil_rates.halo_shielded.ShieldedSHM property), 13`

`v_esc (dddm.statistics.StatModel property), 18`

`v_esc_nodim (dddm.recoil_rates.halo_shielded.ShieldedSHM property), 13`

`velocity_dist() (dddm.recoil_rates.halo.SHM method), 13`

`velocity_dist() (dddm.recoil_rates.halo_shielded.ShieldedSHM method), 13`