

---

# **oPDF API reference**

***Release 1.0***

**Jiaxin Han**

November 14, 2014



CONTENTS

<b>1</b>	<b>oPDF module</b>	<b>1</b>
1.1	Variables and Constants . . . . .	1
1.2	Classes . . . . .	2
<b>2</b>	<b>Utility functions</b>	<b>9</b>
	<b>Python Module Index</b>	<b>13</b>
	<b>Index</b>	<b>15</b>



## OPDF MODULE

Python interface to the C code for oPDF modelling.

It wraps the C functions into python classes with ctypes.

### 1.1 Variables and Constants

#### `oPDF.Globals`

Collection of global variables of the module, of class `globals_t`. It controls numerical precision, internal units, and cosmology.

#### `oPDF.VirTypes`

**Collection of virial definitions. It contains**

- `VirTH`: the spherical collapse prediction (i.e, Bryan & Norman 98 fitting).
- `VirB200`: the 200 times mean density definition.
- `VirC200`: the 200 times critical density definition.

#### `oPDF.HaloTypes`

**Collection of halo parametrizations. It contains**

- `NFWMC`: NFW halo parametrized by  $(M, c)$
- `NFWRhosRs`: NFW,  $(\rho_s, r_s)$
- `NFWPotsRs`: NFW,  $(\psi_s, r_s)$ , with  $\psi_s = 4\pi G \rho_s r_s^2$ .
- `CorePotsRs`: Cored Generalized NFW Potential (inner density slope=0), parametrized by  $(\psi_s, r_s)$
- `CoreRhosRs`: Cored GFW,  $(\rho_s, r_s)$
- `TMPMC`: Template profile,  $(M, c)$  parametrization
- `TMPPotScaleRScale`: Template,  $\psi_s/\psi_{s0}, r_s/r_{s0}$

#### `oPDF.Estimators`

**Collection of dynamical estimators. It contains**

- **RBinLike: binned radial likelihood.** Use `RBinLike.nbin` (integer) and `RBinLike.logscale` (True or False) to control the number and scale of bins.
- `AD`: Anderson-Darling distance.
- `MeanPhaseRaw`: Normalized mean phase deviation  $\bar{\Theta} = (\bar{\theta} - 0.5)/\sigma_\theta$ , to be compared to a standard normal variable.

- **MeanPhase**:  $\bar{\Theta}^2$ , to be compared to a chi-square variable.

## 1.2 Classes

### 1.2.1 Global Parameters

**class** `oPDF.globals_t`

global variables of the module. It controls numerical precision, internal units, and cosmology.

**get\_units()**

query the units

**set\_defaults()**

set default global parameters, including precision, cosmology and units

**set\_units** (*MassInMsunh=7300000000.0, LengthInKpch=0.73, VelInKms=1.0*)

set system of units. specify Mass in Msun/h, Length in kpc/h, Velocity in km/s

### 1.2.2 Halo

**class** `oPDF.Halo` (*halotype=NFWMC, virtype=C200, redshift=0.0, scales=None, TMPid=-1*)

a general halo describing the potential. It has the following properties

#### Variables

- **pars** – raw parameter values. do not change them manually, use `set_param()` to set them.
- **scales** – parameter scales. use `set_type()` to set them.
- **virtype** – virial definition
- **type** – parametrization type. One of `HaloTypes`.

Depending on the type of the halo, some of the following properties may be calculated during `set_param()`:

#### Variables

- **M** – mass
- **c** – concentration
- **Rv** – virial radius
- **Pots** –  $Pots=4*\pi*G*Rhos*Rs^2$ .
- **Rhos** – scale density for NFW
- **Rs** – scale radius
- **RScale** –  $Rs/Rs0$  for TMP profile
- **PotScale** –  $Pots/Pots0$  for TMP profile

define a halo by specifying the parametrization, virial definition and redshift of halo

**halotype**: halo parametrization, one of the `HaloTypes` members

**virtype**: virial definition, one of the `VirTypes` members

**redshift**: redshift of halo

scales: scales of halo parameters, array-like, of the same shape as parameters. default to all-ones if None. physical parameters will be  $\text{params} \times \text{scales}$

TMPIid: template id. only required when halotype is of template type

**get\_current\_TMPIid()**

get the id of the template currently loaded in the system.

this func can be used to check whether the loaded template is the template of the current halo, just in case the template does not match

**mass(r)**

cumulative mass profile

**pot(r)**

potential

**set\_param(pars=[1.0, 1.0])**

set the parameters of the halo

pars: parameters describing the halo

**set\_type(halotype=NFWMC, virtype=C200, redshift=0.0, scales=None, TMPIid=-1)**

set the parametrization, virial definition and redshift of halo

halotype: halo parametrization, one of the [HaloTypes](#) members

virtype: virial definition, one of the [VirTypes](#) members

redshift: redshift of halo

scales: scales of halo parameters, array-like, of the same shape as parameters. default to ones if not specified. physical parameters will be  $\text{params} \times \text{scales}$

### 1.2.3 Tracer

**class oPDF.Tracer(datafile=None, rmin=None, rmax=None, shuffle=True)**

Tracer: a population of tracer particles.

#### Variables

- **halo** – the halo (potential, :py:class:oPDF.Halo) for the tracer.
- **lnL** – likelihood or distance for the sample from the previous likelihood calculation, depending on estimator.
- **nP** – number of particles.
- **mP** – average particle mass.
- **data** – particle data, numpy record array format. It includes the following fields: ('haloid', 'subid', 'flag', 'w', 'r', 'K', 'L2', 'L', 'x', 'v', 'E', 'T', 'vr', 'theta', 'rlim')

#### Note:

- The *w* field is the particle mass in units of the average particle mass. These are all ones if no particle mass is given in the datafile.
- the *haloid* and *subid* fields are only filled if you have *SubID* and *HaloID* datasets in the datafile when loading.
- The *E*, 'theta' and *rlim* fields are the energy, phase-angle, and radial limits (peri and apo-center distances) of the orbits. These depend on the potential, and are only filled when you have done some calculation in a halo, or have filled them explicitly with `set_phase()`.

---

**Note:** The following members are provided for information, but do not manually assign to them. use `radial_count()` and `radial_cut()` to set them.

---

### Variables

- **nbin\_r** – number of radial bins.
- **FlagRLogBin** – whether radial binning is in logspace.
- **RadialCount** – counts in radial bins.
- **rmin** – lower radial cut.
- **rmax** – upper radial cut.

it loads a tracer from the datafile.

optionally, can apply radial cut given by rmin and rmax

---

**Note:** by default, the tracer particles will be shuffled after loading, for easy creation of subsamples by copying later. to keep the original ordering of particles, set `shuffle=False`

---

**NFW\_fit** (*x0=[1, 1], minuittol=1*)

to fit an NFW density PDF with maximum likelihood. results will be printed on screen. also return minuit result and the minuit minimizer.

..note: This is only intended for fitting the Dark Matter density profile to get the NFW parameters. The tracer particle mass should have been properly assigned or adjusted, so that  $m_P \times \text{number\_density} = \text{physical\_density}$ . If you have sampled  $n$  particles from the full sample of  $n_0$  particles, remember to adjust the  $m_P$  of the sample to be  $m_{P0} \times n_0/n$ , so that total mass is conserved.

**NFW\_like** (*pars=[1, 1]*)

NFW log-likelihood. the halo should have been set to one of the NFW types before calling this.

pars are the parameters to be passed to the halo.

return  $\log(\text{likelihood})$

**TSprof** (*pars, proxy='L', nbin=100, estimator=MeanPhaseRaw*)

calculate the likelihood inside equal-count bins of proxy.

return the loglike or f.o.m. for the estimator in each bin, and the bin edges.

proxy and nbin can also be of  $\text{len}>1$ ; in that case, use `self.Views[i].Views[j].lnL` and `self.Views[i].Views[j].proxybin` to get the likelihood and bins in each node

**TSprofCum** (*pars, proxy='r', bins=100, estimator=AD*)

cumulative TS profile. reuturn bin edges, ts, counts

**clean** ()

release the C-allocated memory for the tracer

**copy** (*offset=0, n=0*)

create a subsample by copying  $n$  particles starting from offset. if  $n==0$ , then copy all the particles starting from offset.

return the subsample



**create\_nested\_views** (*viewtypes='EL', nbins=[10, 10]*)

create nested views, i.e., create views according to first proxy, then create sub-views for each view according to the second proxy and so on.

viewtypes can be one, two or more proxies, e.g. 'E','EL','LEr'.

len(nbins) must match len(viewtypes).

the energy need to be set before calling if creating E views

**create\_views** (*n=10, proxy='L'*)

sort the particles according to proxy, and divide into n equal-size subsamples sequentially. these subsamples does not copy the particle data, but only points to the corresponding segments of data in the parent sample, so they are called views, and can be accessed through `Tracer.Views[i]` from the parent sample. the energy need to have been set before calling if proxy is E

**destroy\_views** ()

erase any views from a tracer

**dyn\_fit** (*estimator=RBinLike, x0=[1, 1], xtol=0.001, ftol\_abs=0.01, maxiter=500, verbose=0*)

dynamical fit with the given estimator

**Parameters** estimator(Estimator): estimator to use. select one from [Estimators](#).

x0(array-like): initial parameter values

xtol: tolerance in x to consider convergence

ftol\_abs: tolerance in function values to consider convergence.

convergence is reached when both  $dx < xtol$  and  $df < ftol\_abs$  between subsequent steps in the search.

maxiter: maximum number of iterations

verbose: whether to print during each step.

### Returns

[**x**, **fval**, **status\_success**] x(array): the best fit parameter

fval(float): log-likelihood or fig of merit, depending on estimator

status\_success(bool): whether the search converged successfully, 1 if yes, 0 if no.

**gen\_bin** (*bintype, nbin=30, logscale=True, equalcount=False*)

return bin edges. divide into nbin bins, with nbin+1 edges.

**like\_eval** (*estimator*)

evaluate likelihood or fig of merit with the given estimator in the attached halo. one has to call [set\\_phase\(\)](#) before this.

**likelihood** (*pars, estimator, auto\_rbin=True*)

calculate likelihood. automatically prepare binning, orbits and eval like.

**load** (*datafile*)

load particles from datafile

**nested\_views\_like** (*estimator=AD*)

evaluate likelihood in at the deepest views, and return the sum of them. The likelihood for each view is also available in `Views[i].lnL`

**phase\_density** (*proxy='E', bins=100, method='hist', logscale=False, weight=False, return\_data=False*)

estimate density in proxy-theta space

**phase\_image** (*pars, proxy, bins=30, logscale=True*)

plot an image of the particle distribution in proxy-theta space

**plot\_TSprof** (*pars, proxy='L', nbin=100, estimator=MeanPhaseRaw, xtype='percent-phys',  
linestyle='r-'*)

plot the TS profile

xtype: can be one of 'percent', 'physical', and 'percent-phys'.

when xtype='percent', plot the x-axis with percents.

if xtype='phys', plot x-axis with physical values.

if xtype='percent-phys', plot xaxis in percent scale but label with physical values.

**predict\_radial\_count** (*nbin=100, logscale=True*)

predict radial counts according to oPDF.

`set_phase()` must have been called prior to calling this.

return predicted counts.

**print\_data** (*i=0*)

print information of particle i

**radial\_count** (*nbin=10, logscale=True*)

bin the particles radially, to be used for radial likelihood calculation. The histogram will be recorded in `Tracer.RadialCount[]`.

**radial\_cut** (*rmin=None, rmax=None*)

cut the tracer with bounds [rmin, rmax]. if only rmin or rmax is given, the other bound is not changed.

**resample** (*seed=1024*)

create a bootstrap sample (sampling with replacement) of the same size from tracer return the new sample

**scan\_confidence** (*estimator, x0, ngrids=[10, 10], dx=[0.5, 0.5], logscale=False, maxlike=None*)

scan significance levels around parameter value x0.

it scans ngrids linear bins from x0-dx to x0+dx if logscale=False, or ngrids log bins from log10(x0)-dx to log10(x0)+dx if logscale=True.

If maxlike is given, it is interpreted as the global maximum log-likelihood, and is used to determine significance for RBinLike estimator; otherwise the maximum likelihood is automatically scanned for RBinLike.

**scan\_like** (*estimator, x, y*)

scan a likelihood surface.

x,y are the vectors specifying the binning along x and y dimensions

**return the likelihood value z on grids, to be used for contour plots as**

```
>>> contour(x, y, z)
```

**select** (*flags*)

select particles according to flags array, to create a subsample

**set\_energy** ()

determine particle energy in the attached halo

**set\_orbits** (*set\_phase=True*)

prepare particle orbits inside the attached halo

**set\_phase: whether to calculate the phase-angle of each particle.** phase angle is needed by AD and MeanPhase estimators, but not by RBinLike see `estimator.need_phase` for each estimator.

**set\_phase** (*pars*, *need\_theta=True*)

prepare the phases for phase-related calculations such as `like_eval` or `phase_density`

**shuffle** (*seed=1024*)

shuffle particles randomly.

seed: optional, seeds the random number generator for the shuffle

**sort** (*proxy*, *offset=0*, *n=0*)

sort the particles according to proxy

proxy can be 'E', 'L', 'r' or 'flag'.

**offset, n: optional, sort n particles starting from offset.** `n=0` means sort all particles starting from offset.

**squeeze** ()

remove `P.flag==0` (`data['flag']==0`) particles



## UTILITY FUNCTIONS

These utility functions may or may not be related to the oPDF method. Some of them are just general-purpose plotting or monitoring functions.

`myutils.AD2Sig (AD)`  
convert AndersonDarling TS to sigma

`myutils.ADSurvFunc (AD)`  
survival function for AD test on uniform distributions

`myutils.Chi2Sig (x, dof)`  
convert chi-square value to significance level, for dof degrees of freedom

`myutils.P2Sig (pval)`  
convert pval to sigma

**class** `myutils.ProgressMonitor (total_steps, total_show=100, init_show=0)`  
monitor progress of your loops  
  
init\_show: initial value progress percentage, set to 0 if no reason total\_steps: maximum iteration steps to monitor  
total\_show: number of revealing times

**monitor\_progress** (*current\_step*)  
put this inside loop to monitor progress, to print the percent of job finished.

`myutils.Sig2TS (sig, dof=1)`  
convert a sigma value to the 2\*likelihood ratio

`myutils.contour_handle (color, linestyle='solid')`  
return a patch object to be used for labelling patch objects in legends

`myutils.create31fig (sharex=True, sharey=False, figsize=(8, 8))`  
create a figure with 3 tightly packed subplots

`myutils.density_of_points (data, bins=100, method='kde', weights=None)`  
estimate density of points with kde or histogram2d

data: should be shape [2,n] array

bins: can be an integer or [nx,ny] for number of bins, an ndarray or a list of two arrays for bin edges

method: 'kde' or 'hist', kernel-density-estimate or 2d-histogram estimate

weights: whether to use weights or not. currently only supports hist method.

**return:** (X,Y,Z) ready to be used for contour plots as contour(X, Y, Z). X and Y are mid points of the bins on which Z is calculated.

`myutils.get_extent (X, Y)`  
get extent for X,Y vectors or meshgrids.

**the output is (xmin,xmax,ymin,ymax), the edge-padded boudaries**, assuming X,Y specifies the mid points of bins and uniformly spaced.

can be used to specify extent for imshow()

`myutils.percent2level (p, z)`  
convert percentiles to levels

`myutils.percentile_contour (X, Y, Z, percents=0.683, colors=None, fill=False, linestyle='solid', **kwargs)`  
plot contour at specific percentile levels

X,Y can be both 2-d arrays as Z, or 1-d array specifying the column(horizontally varying) and row coordinates for Z.

percents can be a list, specify the contour percentile levels

colors should be a tuple, e.g, (r,)

fill: bool, whether to plot filled contours

kwargs specify linestyles

**return:** a handle artist of the same linestyle (but not the contour object) to be used in legends

`myutils.plot_circle (cen=[0, 0], r=1, **kwargs)`  
plot a circle

`myutils.plot_cov_ellipse (cov, pos, nstd=1, fill=False, ax=None, **kwargs)`

Plots an *nstd* sigma error ellipse based on the specified covariance matrix (*cov*). Additional keyword arguments are passed on to the ellipse patch artist.

Parameters

**cov** : The 2x2 covariance matrix to base the ellipse on

**pos** [The location of the center of the ellipse. Expects a 2-element] sequence of [x0, y0].

**nstd** [The radius of the ellipse in numbers of standard deviations.] Defaults to 1 standard deviations.

**ax** : The axis that the ellipse will be plotted on. Defaults to the current axis.

Additional keyword arguments are pass on to the ellipse patch.

**Returns** A matplotlib ellipse artist

`myutils.shiftedColorMap (cmap, start=0, midpoint=0.5, stop=1.0, name='shiftedcmap')`

Function to offset the “center” of a colormap. Useful for data with a negative min and positive max and you want the middle of the colormap’s dynamic range to be at zero

Input

**cmap** : The matplotlib colormap to be altered

**start** [Offset from lowest point in the colormap’s range.] Defaults to 0.0 (no lower ofset). Should be between 0.0 and *midpoint*.

**midpoint** [The new center of the colormap. Defaults to ] 0.5 (no shift). Should be between 0.0 and 1.0. In general, this should be  $1 - \text{vmax}/(\text{vmax} + \text{abs}(\text{vmin}))$  For example if your data range from -15.0 to +5.0 and you want the center of the colormap at 0.0, *midpoint* should be set to  $1 - 5/(5 + 15)$  or 0.75

**stop** [Offset from highets point in the colormap’s range.] Defaults to 1.0 (no upper ofset). Should be between *midpoint* and 1.0.

Credit: <http://stackoverflow.com/questions/7404116/defining-the-midpoint-of-a-colormap-in-matplotlib>

`myutils.skeleton(x, y, nbin=10, alpha=0.683, weights=None)`

to divide x into bins and give estimation of center and variance of y inside each bin

**input:** x,y: column vectors to extract skeleton from

nbin: number of bins or bin edges for x

alpha: confidence level for boundary estimation





**m**

myutils, 9

**o**

oPDF, 1



## A

AD2Sig() (in module myutils), 9  
 ADSurvFunc() (in module myutils), 9

## C

Chi2Sig() (in module myutils), 9  
 clean() (oPDF.Tracer method), 4  
 contour\_handle() (in module myutils), 9  
 copy() (oPDF.Tracer method), 4  
 create31fig() (in module myutils), 9  
 create\_nested\_views() (oPDF.Tracer method), 4  
 create\_views() (oPDF.Tracer method), 5

## D

density\_of\_points() (in module myutils), 9  
 destroy\_views() (oPDF.Tracer method), 5  
 dyn\_fit() (oPDF.Tracer method), 5

## E

Estimators (in module oPDF), 1

## G

gen\_bin() (oPDF.Tracer method), 5  
 get\_current\_TMPid() (oPDF.Halo method), 3  
 get\_extent() (in module myutils), 9  
 get\_units() (oPDF.globals\_t method), 2  
 Globals (in module oPDF), 1  
 globals\_t (class in oPDF), 2

## H

Halo (class in oPDF), 2  
 HaloTypes (in module oPDF), 1

## L

like\_eval() (oPDF.Tracer method), 5  
 likelihood() (oPDF.Tracer method), 5  
 load() (oPDF.Tracer method), 5

## M

mass() (oPDF.Halo method), 3  
 monitor\_progress() (myutils.ProgressMonitor method), 9

myutils (module), 9

## N

nested\_views\_like() (oPDF.Tracer method), 5  
 NFW\_fit() (oPDF.Tracer method), 4  
 NFW\_like() (oPDF.Tracer method), 4

## O

oPDF (module), 1

## P

P2Sig() (in module myutils), 9  
 percent2level() (in module myutils), 10  
 percentile\_contour() (in module myutils), 10  
 phase\_density() (oPDF.Tracer method), 5  
 phase\_image() (oPDF.Tracer method), 5  
 plot\_circle() (in module myutils), 10  
 plot\_cov\_ellipse() (in module myutils), 10  
 plot\_TSprof() (oPDF.Tracer method), 6  
 pot() (oPDF.Halo method), 3  
 predict\_radial\_count() (oPDF.Tracer method), 6  
 print\_data() (oPDF.Tracer method), 6  
 ProgressMonitor (class in myutils), 9

## R

radial\_count() (oPDF.Tracer method), 6  
 radial\_cut() (oPDF.Tracer method), 6  
 resample() (oPDF.Tracer method), 6

## S

scan\_confidence() (oPDF.Tracer method), 6  
 scan\_like() (oPDF.Tracer method), 6  
 select() (oPDF.Tracer method), 6  
 set\_defaults() (oPDF.globals\_t method), 2  
 set\_energy() (oPDF.Tracer method), 6  
 set\_orbits() (oPDF.Tracer method), 6  
 set\_param() (oPDF.Halo method), 3  
 set\_phase() (oPDF.Tracer method), 6  
 set\_type() (oPDF.Halo method), 3  
 set\_units() (oPDF.globals\_t method), 2  
 shiftedColorMap() (in module myutils), 10  
 shuffle() (oPDF.Tracer method), 7

Sig2TS() (in module myutils), [9](#)  
skeleton() (in module myutils), [11](#)  
sort() (oPDF.Tracer method), [7](#)  
squeeze() (oPDF.Tracer method), [7](#)

## T

Tracer (class in oPDF), [3](#)  
TSprof() (oPDF.Tracer method), [4](#)  
TSprofCum() (oPDF.Tracer method), [4](#)

## V

VirTypes (in module oPDF), [1](#)