oPDF Tutorial

Release 1.0

Jiaxin Han

November 14, 2014

`oPDF` is a code for modelling the phase space distribution of steady-state tracers in spherical potentials. For more information, check the website.

Please consult the science paper on how it works.

You can use this tutorial interactively in ipython notebook by running

```
ipython notebook --pylab=inline
```

from the root directory of the oPDF code. This will open your browser, and you can click `tutorial.ipynb` in the opened webpage. If that does not work, then simply continue reading this document as a webpage. For the full API documentation, check here.

CHAPTER ONE

GETTING STARTED

1.1 prerequisites

The oPDF code depends on the following libraries:

- C libraries
  - GSL
  - HDF5
- Python libraries
  - numpy, scipy, matplotlib
  - iminuit (optional, only needed if you want to do NFW-likelihood fit to the density profile of dark matter. If you don't have it, you need to comment out the `iminuit` related imports in the header of oPDF.py.)

You can customize the makefile to specify how to compile and link against the GSL and HDF5 libraries, by specifying the `GSLINC,GSLLIB,HDFINC,HDFLIB` flags. ###build the library under the root directory of the code, run

```
make
```

This will generate the library `liboPDF.so`, the backend of the python module. Now you are all set up for the analysis. Open your python shell in the code directory, and get ready for the modelling. If you want to get rid of all the `*.o` files, you can clean them by

```
make clean
```

From now on, you should either work under the current directory, or have added the `oPDF` path to your `PYTHONPATH` before using `oPDF` in python. To add the path, do

```
export PYTHONPATH=$PYTHONPATH:$OPDF_DIR
```

in bash, or the following in csh:

```
setenv PYTHONPATH ${PYTHONPATH}:$OPDF_DIR
```

. Replace `$OPDF_DIR` with the actual root directory of the `oPDF` code above.

1.2 Prepare the data files

The data files are hdf5 files listing the positions and velocities of tracer particles, relative to the position and velocity of the center of the halo. The code comes with a sample file under data/:

- mockhalo.hdf5, a mock stellar halo. The potential is NFW with $M = 183.5017 \times 10^{10} M_{\odot}$, $c = 16.1560$, following the $\rho_{vir} = 200\rho_{crit}$ definition.

Compulsory datasets in a data file:

- `x, shape=[nx3], datatype=float32`. The position of each particle.
- `v, shape=[nx3], datatyep=float32`. The velocity of each particle.

Optional datasets:

- `PartMass, [nx1] or 1, float32`. This is the mass of particles. Assuming 1 if not specified.
- `SubID, [nx1], int32`. This is the subhalo id of each particle, for examination of the effects of subhaloes during the analysis.
- `HaloID, [nx1], int32`. This is the host halo id of each particles.

The units of the data can be specified when running the code.

Note: to construct a tracer sample for a halo, do not use FoF particles alone. Instead, make a spherical selection by including all the particles inside a given radius. These will include FoF particles, background particles, and particles from other FoFs. FoF selection should be avoided because it is an arbitrary linking of particles according to their separations, but not dynamics.

CHAPTER

TWO

A SIMPLE EXAMPLE: FIT THE MOCK HALO WITH RBINLIKE

2.1 Load the data

From the root directory of the code, do

```
from oPDF import *
```

```
datafile=rootdir+"/data/mockhalo.hdf5"
FullSample=Tracer(datafile)
Sample=FullSample.copy(0,1000)
```

```
30000 <oPDF.LP_Particle_t object at 0x4e23e60>
```

This will load the data into FullSample, and make a subsample of 1000 particles from the FullSample. You may want to do your analysis with the full sample. We extract the subsample just for illustrution purpose, to speed up the calculation in this tutorial.

2.2 Perform the fitting.

Now let's fit the data with the radial binned likelihood estimator with 10 logarithmic radial bins.

```
Estimators.RBinLike.nbin=10
x,fval,status=Sample.dyn_fit(Estimators.RBinLike)
print x,fval,status
```

```
[ 161.8486416    19.83133738] 5014.14688084 1
```

In one or two minutes, you will get the results above, where

- `x` is the best-fitting parameters
- `fval` is the maximum log-likelihood value
- `status` =1 means fitting is successful, =0 means fit failed.

That's it! You have got the best-fitting $M = 161.8 \times 10^{10} M_\odot$ and $c = 19.83$. ### Estimate significances How does that compare to the real parameters of $M = 183.5017 \times 10^{10} M_\odot$, $c = 16.1560$? Not too far, but let's check the likelihood ratio of the two models

```
x0=[183.5017,16.1560]
f0=Sample.likelihood(x0, Estimators.RBinLike)
likerat=2*(fval-f0)
print likerat
```

```
1.0933322551
```

So we got a likelihood ratio of 1.09. How significant is that? According to Wilks' theorem, if the data follows the null model (with the real parameters), then the likelihood ratio between the best-fit and the null would follow a χ^2 distribution. Since we have two free parameters, we should compare our likelihood ratio to a $\chi^2(dof = 2)$ distribution. We can obtain the pval from the survival function of a χ^2 distribution, and convert that to a Guassian significance level. This is automatically done by the Chi2Sig() utility function

```
from myutils import Chi2Sig
significance=Chi2Sig(likerat, dof=2)
print significance
```

```
0.555026554617
```

So the best-fitting differs from the real parameters by 0.56σ, which is not significant at all. In other words, the best-fitting parameters are quite consistent with the real parameters!

2.3 Confidence Contour

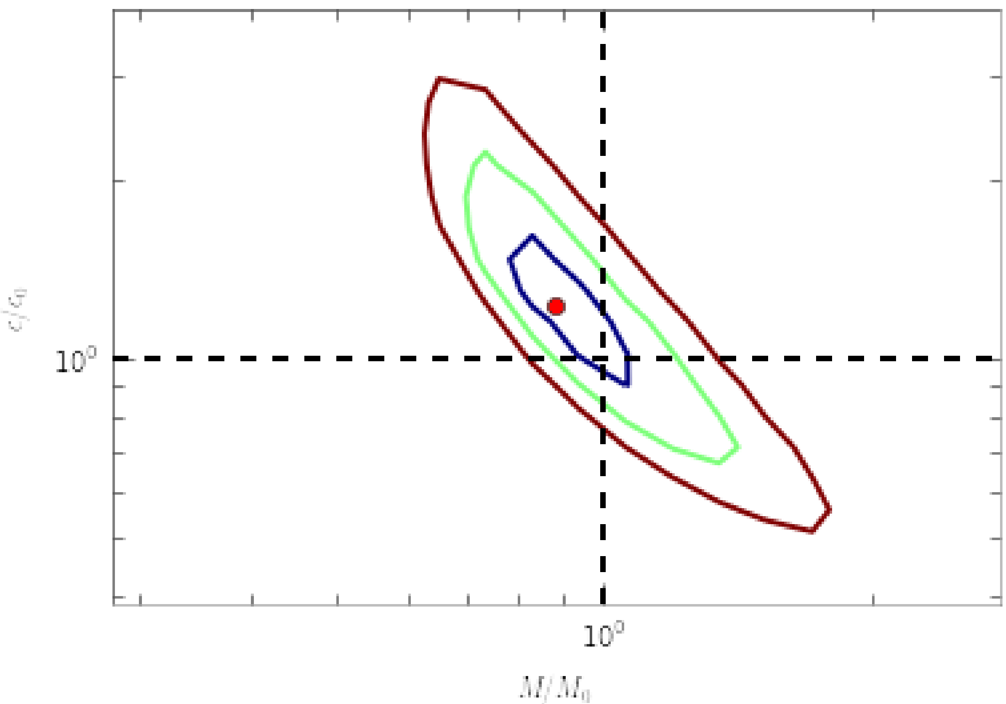
Following the same philosophy for the significance levels, we can start to define confidence contours formed by points that differ from the best-fitting parameters by a given significance level. This is done by scanning a likelihood surface and then converting it to a significance surface. For example, below we scan 20×20 grids around the best-fitting parameters `x`, inside a box spanning from `log10(x)-dx` to `log10(x)+dx` in each dimension. For the confidence levels of RBinLike, we can provide the maximum likelihood value that we obtained above, to save the function from searching for maxlike itself. Be prepared that the scan can be slow.

```
m,c,sig=Sample.scan_confidence(Estimators.RBinLike, x, ngrids=[20,20], dx=[0.5,0.5], logscale=True,
```

Now let's plot them in units of the real parameter values:

```
plt.contour(m/x0[0],c/x0[1],sig,levels=[1,2,3]) #1,2,3sigma contours.
plt.plot(x[0]/x0[0],x[1]/x0[1],'ro') #the best-fitting
plt.plot(plt.xlim(),[1,1], 'k--', [1,1], plt.ylim(),  'k--')# the real parameters
plt.loglog()
plt.xlabel(r'$M/M_0$')
plt.ylabel(r'$c/c_0$')
```

```
<matplotlib.text.Text at 0x4fa4fd0>
```



2.4 Phase Images

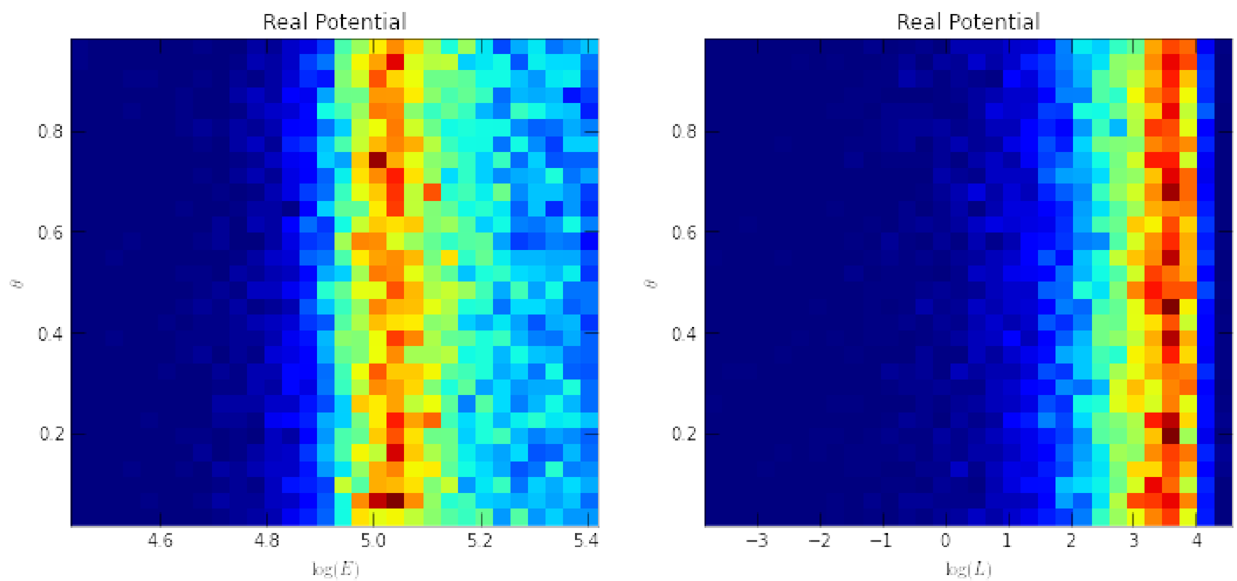
How does the data look in (θ, E, L) space? We can create images showing the distribution of particles in these coordinates. These images give a direct visualization of how uniformly the tracer are distributed along θ-direction, on different (E, L) orbits. They are quite useful for spotting deviations from steady-stateness in particular regions in phase space, for example, to examine local deviations caused by subhaloes.

To avoid having too few particles in each pixel we will start by drawing a larger sample as NewSample, and then plot the images adopting the real potential with parameters `x0`.

```
NewSample=FullSample.copy(0,20000)
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
NewSample.phase_image(x0, proxy='E')
plt.title('Real Potential')
plt.subplot(1,2,2)
NewSample.phase_image(x0, proxy='L')
plt.title('Real Potential')
```

```
<matplotlib.text.Text at 0x524ee90>
```

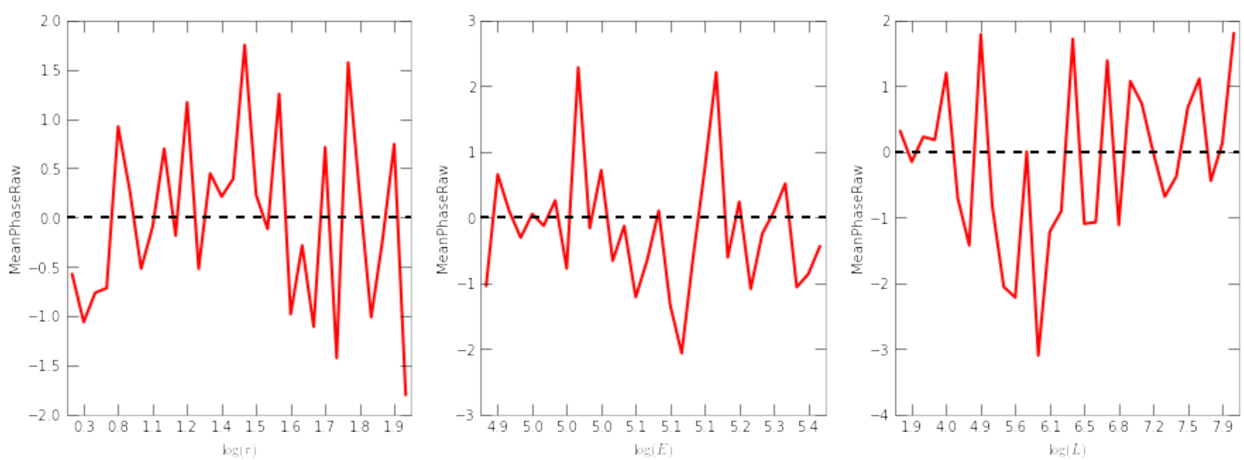


We can see that the particle distribution is indeed uniform (roughly given the current resolution) along the θ-direction, irrespective of the energy and angular momentum.

2.5 TS profiles

If you want a more quantitative view of how much deviation there is at each E, L or even r, you can plot the mean phase deviation or AD distance (Test Statistics, or TS) inside different (E, L, r) bins.

```
nbin=30
x0=[183.5017,16.1560]
plt.figure(figsize=(15,5))
for i,proxy in enumerate('rEL'):
    plt.subplot(1,3,i+1)
    NewSample.plot_TSprof(x0, proxy, nbin)
    plt.plot(plt.xlim(), [0,0], 'k--')
```



See, the mean phase deviations are within 3:math:*sigma* everywhere. Note that the raw mean phase is a standard normal variable if the tracer is in steady-state under the potential.

CHAPTER

THREE

CUSTOMIZING THE ANALYSIS

3.1 Estimators

There are several predefined estimators to choose from when you need an estimator as a parameter. These are listed as members of `Estimators`. In most cases, you can freely choose from the following when an estimator is required.

- Estimators.RBinLike
- Estimators.AD
- Estimators.MeanPhase
- Estimators.MeanPhaseRaw (same as MeanPhase but returns the un-squared mean phase deviation, so it is a standard normal variable instead of a chi-square for MeanPhase).

For RBinLike, you can also customize the number of radial bins and whether to bin in linear or log scales. For example, the following will change the RBinLike to use 20 linear bins.

```
Estimators.RBinLike.nbin=20
Estimators.RBinLike.logscale=False
```

3.2 units

The system of units is specified in three fundamental units: Mass[$M_\odot/h$], Length[kpc:math:/h], Velocity[km/s]. You can query the current units with

```
Globals.get_units()
```

```
Mass  : 7300000000.0 Msun/h
Length: 0.73 kpc/h
Vel   : 1.0 km/s
```

```
(7300000000.0, 0.73, 1.0)
```

These units are typical for a simulated tracer in units of `[1e10Msun/h, kpc/h, km/s]` with `h=0.73`. The oPDF code does not need to know the value of the hubble constant, as long as the units are correctly specified. It is the user's responsibility to make sure that his/her units are consistent with his assumed hubble parameter.

If you want to change the system of units, you must do it immediately after importing the `oPDF` module, to avoid inconsistency with units of previously loaded tracers. For example, to change the units to (1e10Msun/h,kpc/h,km/s),

```
from oPDF import *
Globals.set_units(1e10,1,1)
```

The user should only use Globals.set_units() to change the units, which automatically updates several interal constants related to units. Never try to change the internal unit variables (e.g., Globals.units.MassInMsunh) manually.

3.3 cosmology

The cosmology parameters (Ω_{M0}, $\Omega_{\Lambda0}$)can be accessed through

```
print Globals.cosmology.OmegaM0, Globals.cosmology.OmegaL0
```

```
0.3 0.7
```

To change the cosmology to (0.25, 0.75), simply do

```
Globals.cosmology.OmegaM0=0.25
Globals.cosmology.OmegaL0=0.75
```

Again this is advised to be done in the beginning, to avoid inconsistency in the calculations.

3.4 parametrization of the potential

The default parameterization of the potential is a NFW potential with mass and concentration parameters. You can change the parametrization of the halo associated with your tracer. For example, if you want to fit for (ρ_s, r_s) instead of (M, c), then

```
Sample.halo.set_type(halotype=HaloTypes.NFWRhosRs)
```

Available types are listed as members of the `HaloTypes` objects, including:

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

To use template profiles, you have to creates them first, in the form of $(r, \psi, \rho(< r))$ arrays and the real r_s parameter to be added to C/TemplateData.h. You need to recompile the C library once this is done. PotentialProf.py can help you in generating the templates from DM distributions.

If you use template profiles, you also need to specify the template id, to tell the code which template in TemplateData.h to use. For example,

```
Sample.halo.set_type(halotype=HaloTypes.TMPPotScaleRScale, TMPid=5)
```

You can also change the virial definition and redshift of the halo, for example:

```
Sample.halo.set_type(virtype=VirTypes.B200, redshift=0.1)
```

When fitting for the potential, it is always a good choice to adjust the scales of parameters so that the numerical values of the parameters are of order 1. oPDF allows you to change the scale of the parameters. The physical values of the parameters will be the raw parameters times the scale of parameters. By default, the scales are all set to unity. We can change them as

```
Sample.halo.set_type(scales=[100,10])
```

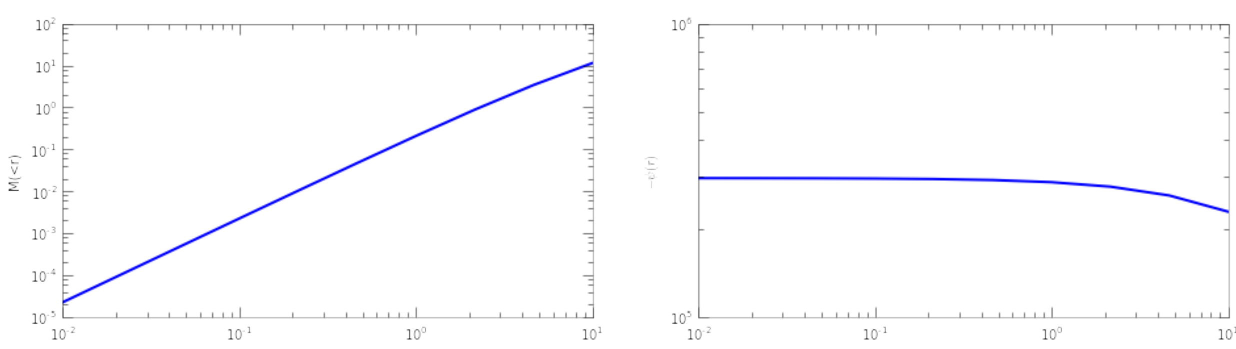
Now if we fit the Sample again with the RBinLike estimator, instead of `x=[ 161.8 19.8]`, we will get `x=[1.618 1.98]` as the best fit, but the physical values are not changed.

3.4.1 Halos

Each tracer is associated with a halo. You can also work with a seperate halo object. There are several methods associated with a halo object. You can set_type(), set_param(), get the mass and potential profiles

```
halo=Halo(halotype=HaloTypes.NFWMC)
halo.set_param([180,13])
r=np.logspace(-2,1,10)
plt.figure(figsize=(16,4))
plt.subplot(121)
plt.loglog(r, halo.mass(r))
plt.xlabel('r')
plt.ylabel('M(<r)')
plt.subplot(122)
plt.loglog(r, -halo.pot(r))
plt.xlabel('r')
plt.ylabel(r'$-\psi(r)$')
```

```
<matplotlib.text.Text at 0x5e48510>
```



3.5 selecting and cutting

This applies a radial cut from 1 to 100 in system unit :

```
Sample.radial_cut(1,100)
```

This creates a subsample by selecting high angular momentum (L>1e4) particles:

```
SubSample=Sample.select(Sample.data['L']>1e4)
```

All the particle data can be accessed from the record array Sample.data. You can do similar selections (and many other operations) on any available fields of the data. Have a look at the datatype or `Particle_t._fields_` to see the available fields

```
print Sample.data.dtype.names
```

```
('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. The average particle mass is `Sample.mP`. 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 the set_phase() function, e.g.,

```
Sample.set_phase(x0)
print Sample.data['E'][10]
print Sample.data['theta'][35]
```

```
9201829.25741
0.982207688183
```

3.6 Extending the code

- To add new types of potential
- in C/halo.h: add your HaloType identifier in HaloType_t
- in C/halo.c:
  - write your halo initializer in halo_set_param()
  - write your potential function in halo_pot()
  - optionally, write your cumulative mass profile in halo_mass(), and add any initilization in halo_set_type() if needed.
- in oPDF.py:

  - add your newly defined halotype to the following line `HaloTypes=NamedEnum(...`

- To add new template profiles
- Generate your template in the form of $(r, \psi, \rho(< r))$ arrays, and append to C/PotentialTemplate in TemplateData.h
- Append the scale radius of the new template to TemplateScale in C/TemplateData.h. This is only used if you want to use TMPMC parametrization. In this case the scale radius must be the radius where the logrithmic slope of density equals to -2.
- $\rho(< r)$ and r_s are only needed if you want to use TMPMC parametrization. If you only want to use TMPPotScaleRScale parametrization, you can fill $\rho(< r)$ and r_s with ones or any value.
- To add new estimators
- check C/models.c

You need to recompile the C library once this is done. PotentialProf.py can help you in generating the templates from DM distributions.

CHAPTER FOUR

ADDITIONAL FEATURES

4.1 Parallel jobs

The C backend of oPDF is fully parallelized with `OpenMP` for parallel computation on shared memory machines. To control the number of threads used, for example to use 16 threads, set the environment variable

```
export OMP_NUM_THREADS=16
```

in bash or

```
setenv OMP_NUM_THREADS 16
```

in csh before running.

When submitting python scripts containing `oPDF` calculations to a batch system on a server, try to submit to a shared memory node and request more than one CPUs on the node to make use of the parallel power.

4.2 Memory management

Each loaded tracer is associated with a memory block in C. If you are certain you no longer need the tracer, you can clean it to free up memory. For example,

```
NewSample.clean()
```

will clear our previously created NewSample. If you know you only need the tracer for certain operations, you can automate the loading and cleaning process by using `with` statement:

```
with Tracer(datafile) as TempSample:
    NewSample=TempSample.copy(0,100)
```

```
30000 <oPDF.LP_Particle_t object at 0x564f3b0>
```

This will load the datafile into `TempSample`, create `NewSample` from `TempSample`, and clear `TempSample` when exiting the `with` block.

4.3 Bootstrap sampling

To create bootstrap samples (sample with replacement), just sample with a different seed each time

```
BSSample=Sample.resample(seed=123)
```

4.4 NFW-likelihood

To fit a spatial distribution of particles to an NFW profile (e.g., fitting the distribution of dark matter particles in a halo)

```
Sample.NFW_fit()
```

```
**********************************************************************
-------------------------------------------------------------------------------------------
fval = 7019.816934338302 | nfcn = 68 | ncalls = 68
edm = 1.6793878043880015e-05 (Goal: 5e-05) | up = 0.5
-------------------------------------------------------------------------------------------
|          Valid |    Valid Param | Accurate Covar |         Posdef |    Made Posdef |
-------------------------------------------------------------------------------------------
|           True |           True |           True |           True |          False |
-------------------------------------------------------------------------------------------
|     Hesse Fail |        Has Cov |      Above EDM |                |  Reach calllim |
-------------------------------------------------------------------------------------------
|          False |           True |          False |             `' |          False |
-------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------
|      | Name  |  Value    | Para Err |   Err-    |   Err+    |  Limit-   |  Limit+   |
----------------------------------------------------------------------------------------------
|    0 |     m =  15.43    |  0.5686  |           |           |           |           |
|    1 |     c =  11.99    |  1.183   |           |           |           |           |
----------------------------------------------------------------------------------------------

**********************************************************************
['m', `c']
((1.0, -0.20669988381300106), (-0.20669988381300106, 1.0))
       |    0    1
--------------------
m    0 | 1.00 -0.21
c    1 | -0.21 1.00
--------------------
```

```
(({'hesse_failed': False, 'has_reached_call_limit': False, 'has_accurate_covar': True, 'has_posdef_c
  [{'is_const': False, 'name': 'm', 'has_limits': False, 'value': 15.43079877133447, 'number': 0L, '
   {'is_const': False, 'name': 'c', 'has_limits': False, 'value': 11.989314056141591, 'number': 1L,
 <iminuit._libiminuit.Minuit at 0x5c3f2f0>)
```

In order for this to make sense, `Sample` should be loaded with dark matter particles of equal particle mass given in `Sample.mP`, and the number density profile times `Sample.mP` should give the physical density profile.

You also need the `iminuit` python package before you can use this function. If you don't have that, you need to comment out the `iminuit` related imports in the header of `oPDF.py`.