iminuit Documentation

Release 1.2

Piti Ongmongkolkul

November 08, 2016

Contents

1 What is iminuit? 3

2 In a nutshell 5
- 2.1 Installation . . . 5
  - 2.1.1 Dependencies . . . 5
  - 2.1.2 Stable version . . . 5
  - 2.1.3 Conda . . . 6
  - 2.1.4 Windows . . . 6
  - 2.1.5 Development version . . . 6
  - 2.1.6 Docs . . . 6
  - 2.1.7 Testing . . . 6
- 2.2 Full API Documentation . . . 7
  - 2.2.1 Quick Summary . . . 7
  - 2.2.2 Minuit . . . 7
  - 2.2.3 Utility Functions . . . 18
  - 2.2.4 Return Value Struct . . . 19
  - 2.2.5 Function Minimum Struct . . . 19
  - 2.2.6 Minos Error Struct . . . 20
  - 2.2.7 Minuit Parameter Struct . . . 20
  - 2.2.8 Function Signature Extraction Ordering . . . 21

3 Tutorial 23

4 API 25

5 Technical Stuff 27

6 You can help 29

Python Module Index 31

MINUIT from Python - Fitting like a boss

- Code: https://github.com/iminuit/iminuit
- Documentation: http://iminuit.readthedocs.org/
- Mailing list: https://groups.google.com/forum/#!forum/iminuit
- PyPI: https://pypi.python.org/pypi/iminuit
- License: LGPL (the iminuit source is MIT, but the bundled MINUIT is LGPL and thus the whole package is LGPL)

CHAPTER 1

What is iminuit?

Interactive IPython-friendly mimizer based on SEAL Minuit.

(It's included in the package, no need to install it separately.)

iminuit is designed from ground up to be fast, interactive and cython friendly. iminuit extract function signature very permissively starting from checking *func_code* down to last resort of parsing docstring (or you could tell iminuit to stop looking and take your answer). The interface is inspired heavily by PyMinuit and the status printout is inspired by ROOT Minuit. iminuit is mostly compatible with PyMinuit (with few exceptions). Existing PyMinuit code can be ported to iminuit by just changing the import statement.

CHAPTER 2

In a nutshell

```
from iminuit import Minuit
def f(x, y, z):
    return (x - 2) ** 2 + (y - 3) ** 2 + (z - 4) ** 2
m = Minuit(f)
m.migrad()
print(m.values)  # {'x': 2,'y': 3,'z': 4}
print(m.errors)  # {'x': 1,'y': 1,'z': 1}
```

If you are interested in fitting a curve or distribution, take a look at probfit.

2.1 Installation

iminuit works with Python 2.7 as well as 3.4 or later.

2.1.1 Dependencies

Like most Python packages, iminuit installation requires setuptools.

The following dependencies are optional:

- numpy
- ipython
- matplotlib
- pytest, pytest-cov
- Cython
- Sphinx, sphinx-rtd

TODO: describe better where which dependency is used.

2.1.2 Stable version

To install the latest stable version:

```
$ pip install iminuit
```

2.1.3 Conda

Conda packages for iminuit are available via the `astropy` channel at https://anaconda.org/astropy/iminuit

```
$ conda install -c astropy iminuit
```

2.1.4 Windows

For Windows, Christoph Gohlke made a nice windows binary to save you all from Windows compilation nightmare:

http://www.lfd.uci.edu/~gohlke/pythonlibs/#iminuit

2.1.5 Development version

To install the latest development version clone the repository from Github:

```
$ git clone https://github.com/iminuit/iminuit.git
$ cd iminuit
$ python setup.py install
```

2.1.6 Docs

You will need `sphinx` and `sphinx_rtd_theme`. They can be installed via

```
$ pip install sphinx
$ pip install sphinx_rtd_theme
```

To generate html docs locally, `iminuit` has to be available. To check if that is the case, and which version you're using, you can use this command:

```
$ python -c 'import iminuit; print(iminuit)'
```

One way to achieve this is to do this:

```
$ python setup.py build_ext --inplace
$ python setup.py develop
```

Another is to just install `iminuit` into `site-packages`:

```
$ python setup.py install
```

Once you have `iminuit` available, use these commands to build the docs:

```
$ cd doc
$ make html
```

The HTML output is here:

```
$ open _build/html/index.html
```

2.1.7 Testing

To run the tests you need to install pytest.

To run the iminuit tests for an installed version of the package:

```
python -m pytest --pyargs iminuit
```

To run the tests from the source folder (e.g. during pytest development), use this command:

```
$ make test
```

To get a coverage report from the tests:

```
$ make coverage
```

2.2 Full API Documentation

2.2.1 Quick Summary

These are the things you will use a lot:

util.describe(f[, verbose])	Try to extract the function argument names.
Minuit(fcn[, throw_nan, pedantic, frontend, ...])	Construct minuit object from given *fcn*
Minuit.migrad(self, int ncall=10000[, ...])	Run migrad.
Minuit.minos(self[, var, sigma])	Run minos for parameter *var*.
Minuit.values	Parameter values (dict: name -> value)
Minuit.args	Parameter value tuple
Minuit.errors	Parameter parabolic errors (dict: name -> error)
Minuit.get_merrors(self)	Dictionary of varname-> MinosError Struct
Minuit.fval	Last evaluated FCN value
Minuit.fitarg	fitarg: object
Minuit.mnprofile(self, vname[, bins, bound, ...])	Calculate minos profile around the specified range.
Minuit.draw_mnprofile(self, vname[, bins, ...])	Draw minos profile around the specified range.
Minuit.mncontour(self, x, y, int numpoints=20)	Minos contour scan.
Minuit.mncontour_grid(self, x, y[, bins, ...])	Compute gridded minos contour.
Minuit.draw_mncontour(self, x, y[, bins, ...])	Draw minos contour.

2.2.2 Minuit

class iminuit.**Minuit**(*fcn*, *throw_nan=False*, *pedantic=True*, *frontend=None*, *forced_parameters=None*, *print_level=1*, *errordef=None*, *grad_fcn=None*, ***kwds*)

Construct minuit object from given *fcn*

Arguments:

- **fcn**: the function to be optimized. Minuit automagically finds parameters names. More information about how Minuit detects function signature can be found in *Function Signature Extraction Ordering*

Builtin Keyword Arguments:

- **throw_nan**: set fcn to raise RuntimeError when it encounters *nan*. (Default False)
- **pedantic**: warns about parameters that do not have initial value or initial error/stepsize set.
- **frontend**: Minuit frontend. There are two builtin frontends.
  1. ConsoleFrontend which is design to print out to terminal.
  2. HtmlFrontend which is designed to give a nice output in IPython notebook session.

By Default, Minuit switch to HtmlFrontend automatically if it is called in IPython session. It uses ConsoleFrontend otherwise.

•**forced_parameters**: tell Minuit not to do function signature detection and use this argument instead. (Default None (automagically detect signature))

•**print_level**: set the print_level for this Minuit. 0 is quiet. 1 print out at the end of migrad/hesse/minos.

•**errordef**: Optional. Amount of increase in fcn to be defined as 1 σ. If None is given, it will look at *fcn.default_errordef()*. If *fcn.default_errordef()* is not defined or not callable iminuit will give a warning and set errordef to 1. Default None(which means errordef=1 with a warning).

•**grad_fcn**: Optional. Provide a function that calculates the gradient analytically and returns an iterable object with one element for each dimension. If None is given minuit will calculate the gradient numerically. (Default None)

Parameter Keyword Arguments:

Similar to PyMinuit. iminuit allows user to set initial value, initial stepsize/error, limits of parameters and whether parameter should be fixed or not by passing keyword arguments to Minuit. This is best explained through an example:

```
def f(x,y):
    return (x-2)**2 + (y-3)**2
```

•Initial value(varname):

```
#initial value for x and y
m = Minuit(f, x=1, y=2)
```

•Initial step size/error(fix_varname):

```
#initial step size for x and y
m = Minuit(f, error_x=0.5, error_y=0.5)
```

•Limits (limit_varname=tuple):

```
#limits x and y
m = Minuit(f, limit_x=(-10,10), limit_y=(-20,20))
```

•Fixing parameters:

```
#fix x but vary y
m = Minuit(f, fix_x=True)
```

Note: Tips: You can use python dictionary expansion to programatically change the fitting arguments.

```
kwdarg = dict(x=1., error_x=0.5)
m = Minuit(f, **kwdarg)
```

You can also obtain fit arguments from Minuit object to reuse it later too. *fitarg* will be automatically updated to the minimum value and the corresponding error when you ran migrad/hesse:

```
m = Minuit(f, x=1, error_x=0.5)
my_fitarg = m.fitarg
another_fit = Minuit(f, **my_fitarg)
```

---

migrad(*self*, *int ncall=10000*, *resume=True*, *int nsplit=1*, *precision=None*)
Run migrad.

Migrad is an age-tested(over 40 years old, no kidding), super robust and stable minimization algorithm. It even has wiki page. You can read how it does the magic at here.

Arguments:

•**ncall**: integer (approximate) maximum number of call before migrad stop trying. Default 10000.

•**resume**: boolean indicating whether migrad should resume from the previous minimizer attempt(True) or should start from the beginning(False). Default True.

•**split**: split migrad in to *split* runs. Max fcn call for each run is ncall/nsplit. Migrad stops when it found the function minimum to be valid or ncall is reached. This is useful for getting progress. However, you need to make sure that ncall/nsplit is large enough. Otherwise, migrad will think that the minimum is invalid due to exceeding max call (ncall/nsplit). Default 1(no split).

•**precision**: override miniut own's internal precision.

Return:

Function Minimum Struct, list of *Minuit Parameter Struct*

hesse(*self*)
Run HESSE.

HESSE estimates error matrix by the second derivative at the minimim. This error matrix is good if your χ^2 or likelihood profile is parabolic at the minimum. From my experience, most of the simple fits are.

`minos()` makes no parabolic assumption and scan the likelihood and give the correct error asymmetric error in all cases(Unless your likelihood profile is utterly discontinuous near the minimum). But, it is much more computationally expensive.

Returns:

list of *Minuit Parameter Struct*

minos(*self*, *var=None*, *sigma=1.0*, *unsigned int maxcall=1000*)
Run minos for parameter *var*.

If *var* is None it runs minos for all parameters

Arguments:

•**var**: optional variable name. Default None.(run minos for every variable)

•**sigma**: number of σ error. Default 1.0.

Returns:

Dictionary of varname to *Minos Error Struct* if minos is requested for all parameters.

args
Parameter value tuple

values
Parameter values (dict: name -> value)

errors
Parameter parabolic errors (dict: name -> error)

fitarg
fitarg: object Current Minuit state in form of a dict.

•name -> value

•error_name -> error

•fix_name -> fix

•limit_name -> (lower_limit, upper_limit)

This is very useful when you want to save the fit parameters and re-use them later. For example,:

```
m = Minuit(f, x=1)
m.migrad()
fitarg = m.fitarg

m2 = Minuit(f, **fitarg)
```

merrors
MINOS errors (dict).

Using this method is not recommended. It was added only for PyMinuit compatibility. Use *get_merrors()* instead, which returns a dictionary of name -> *Minos Error Struct* instead.

Dictionary entries for each parameter:

•(name,1.0) -> upper error

•(name,-1.0) -> lower error

fval
Last evaluated FCN value

See also:

get_fmin()

edm
Estimated distance to minimum.

See also:

get_fmin()

covariance
Covariance matrix (dict (name1, name2) -> covariance).

See also:

matrix()

gcc
Global correlation coefficients (dict : name -> gcc)

errordef
errordef: 'double' Amount of change in FCN that defines 1 $sigma$ error.

Default value is 1.0. *errordef* should be 1.0 for χ^2 cost function and 0.5 for negative log likelihood function.

This parameter is sometimes called `UP` in the MINUIT docs.

tol
tol: 'double' Tolerance.

One of the MIGRAD convergence criteria is `edm < edm_max`, where `edm_max` is calculated as `edm_max = 0.0001 * tol * UP`.

contour(*self*, *x*, *y*, *bins=20*, *bound=2*, *args=None*, *subtract_min=False*)
2D contour scan.

return contour of migrad result obtained by fixing all others parameters except **x** and **y** which are let to varied.

Arguments:

- **x** variable name for X axis of scan
- **y** variable name for Y axis of scan
- **bound** If bound is 2x2 array [[v1min,v1max],[v2min,v2max]]. If bound is a number, it specifies how many σ symmetrically from minimum (minimum+- bound*:math:*sigma*). Default 2
- **subtract_min** subtract_minimum off from return value. This makes it easy to label confidence interval. Default False.

Returns:

x_bins, y_bins, values

values[y, x] <– this choice is so that you can pass it to through matplotlib contour()

See also:

mncontour()

Note: If *subtract_min=True*, the return value has the minimum subtracted off. The value on the contour can be interpreted *loosely* as $i^2 \times$ up where i is number of standard deviation away from the fitted value *WITHOUT* taking into account correlation with other parameters that's fixed.

---

draw_contour(*self*, *x*, *y*, *bins=20*, *bound=2*, *args=None*, *show_sigma=False*)
Convenience wrapper for drawing contours.

The argument is the same as *contour()*. If *show_sigma=True'(Default), the label on the contour lines will show how many :math:*sigma' away from the optimal value instead of raw value.

Note: Like *contour()*, the error shown on the plot is not strictly the 1 σ contour since the other parameters are fixed.

See also:

contour() mncontour()

draw_mncontour(*self*, *x*, *y*, *bins=100*, *nsigma=2*, *numpoints=20*, *sigma_res=4*)
Draw minos contour.

Arguments:

- **x**, **y** parameter name
- **bins** number of bin in contour grid.
- **nsigma** number of sigma contour to draw
- **numpoints** number of points to calculate for each contour
- **sigma_res** number of sigma level to calculate MnContours. Default 4.

Returns:

x, y, gridvalue, contour

gridvalue is interorlated nsigma

draw_mnprofile (*self*, *vname*, *bins=30*, *bound=2*, *subtract_min=False*, *band=True*, *text=True*)
Draw minos profile around the specified range.

It is obtained by finding Migrad results with **vname** fixed at various places within **bound**.

Arguments:

•**vname** variable name to scan

•**bins** number of scanning bin. Default 30.

•**bound** If bound is tuple, (left, right) scanning bound. If bound is a number, it specifies how many σ symmetrically from minimum (minimum+- bound* σ). Default 2.

•**subtract_min** subtract_minimum off from return value. This makes it easy to label confidence interval. Default False.

•**band** show green band to indicate the increase of fcn by *errordef*. Default True.

•**text** show text for the location where the fcn is increased by *errordef*. This is less accurate than *minos()*. Default True.

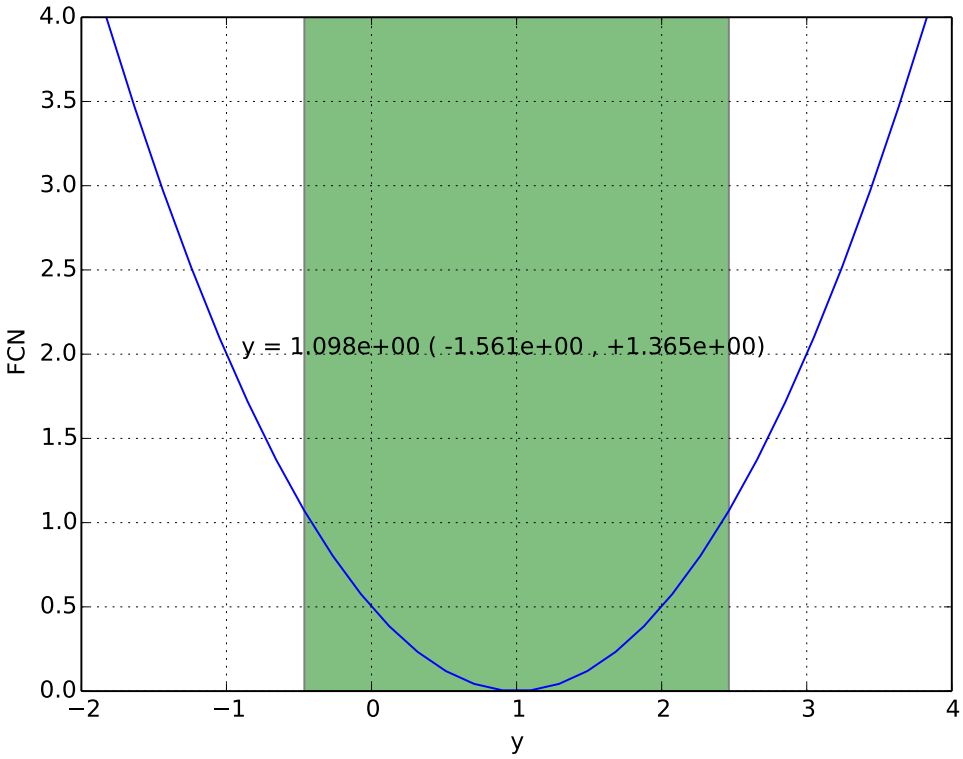
Returns:

bins(center point), value, migrad results

```
from iminuit import Minuit


def f(x, y, z):
    return (x - 1) ** 2 + (y - x) ** 2 + (z - 2) ** 2


m = Minuit(f, print_level=0, pedantic=False)
m.migrad()
m.draw_mnprofile('y')
```



draw_profile (*self*, *vname*, *bins=100*, *bound=2*, *args=None*, *subtract_min=False*, *band=True*, *text=True*)
A convenient wrapper for drawing profile using matplotlib.

Note: This is not a real minos profile. It's just a simple 1D scan. The number shown on the plot is taken from the green band. They are not minos error. To get a real minos profile call *mnprofile()* or *draw_mnprofile()*

Arguments:

In addition to argument listed on *profile()*. draw_profile take these addition argument:

- **band** show green band to indicate the increase of fcn by *errordef*. Note again that this is NOT minos error in general. Default True.
- **text** show text for the location where the fcn is increased by *errordef*. This is less accurate than *minos()* Note again that this is NOT minos error in general. Default True.

See also:

mnprofile() draw_mnprofile() profile()

get_fmin (*self*)
Current FunctionMinimum Struct

get_initial_param_state (*self*)
Initial setting in form of MinuitParameter Struct

get_merrors (*self*)
Dictionary of varname-> MinosError Struct

get_num_call_fcn(*self*)
Total number of calls to FCN (not just the last operation)

get_param_states(*self*)
List of current MinuitParameter Struct for all parameters

grad_fcn
Gradient function of the cost function

is_clean_state(*self*)
Check if minuit is in a clean state, ie. no migrad call

is_fixed(*self*, *vname*)
Check if variable *vname* is (initially) fixed

latex_initial_param(*self*)
Build `iminuit.latex.LatexTable` for initial parameter

latex_matrix(*self*)
Build `LatexFactory` object with the correlation matrix

latex_param(*self*)
build `iminuit.latex.LatexTable` for current parameter

list_of_fixed_param(*self*)
List of (initially) fixed parameters

list_of_vary_param(*self*)
List of (initially) float varying parameters

matrix(*self*, *correlation=False*, *skip_fixed=True*)
Error or correlation matrix in tuple or tuples format.

matrix_accurate(*self*)
Check if covariance (of the last migrad) is accurate

merrors
MINOS errors (dict).

Using this method is not recommended. It was added only for PyMinuit compatibility. Use *`get_merrors()`* instead, which returns a dictionary of name -> *Minos Error Struct* instead.

Dictionary entries for each parameter:

- (name,1.0) -> upper error
- (name,-1.0) -> lower error

merrors_struct
merrors_struct: object MINOS error calculation information (dict name -> struct)

migrad_ok(*self*)
Check if minimum is valid

mncontour(*self*, *x*, *y*, *int numpoints=20*, *sigma=1.0*)
Minos contour scan.

A proper n **sigma** contour scan. This is the line where the minimum of fcn with x,y is fixed at points on the line and letting the rest of variable varied is change by **sigma** * errordef^2 . The calculation is very very expensive since it has to run migrad at various points.

Note: See http://wwwasdoc.web.cern.ch/wwwasdoc/minuit/node7.html

Arguments:

- **x** string variable name of the first parameter
- **y** string variable name of the second parameter
- **numpoints** number of points on the line to find. Default 20.
- **sigma** number of sigma for the contour line. Default 1.0.

Returns:

x minos error struct, y minos error struct, contour line

contour line is a list of the form [[x1,y1]...[xn,yn]]

mncontour_grid(*self*, *x*, *y*, *bins=100*, *nsigma=2*, *numpoints=20*, *int sigma_res=4*, *edges=False*)

Compute gridded minos contour.

Arguments:

- **x**, **y** parameter name
- **bins** number of bins in the grid. The boundary of the grid is selected automatically by the minos error computed. Default 100.
- **nsigma** number of sigma to draw. Default 2
- **numpoints** number of points to calculate mncontour for each sigma points(there are sigma_res*nsigma total)
- **sigma_res** number of sigma level to calculate MnContours
- **edges** Return bin edges instead of mid value(pass True if you want to draw it using pcolormesh)

Returns:

xgrid, ygrid, sigma, rawdata

rawdata is tuple of (x,y,sigma_level)

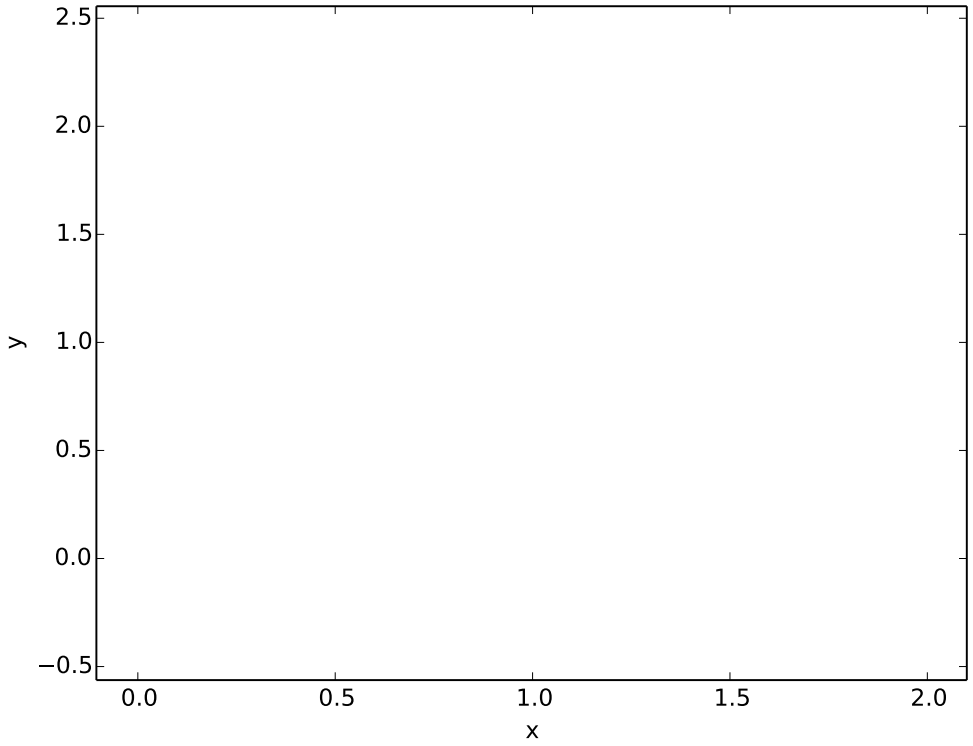
See also:

draw_mncontour()

```
from iminuit import Minuit


def f(x, y, z):
    return (x - 1) ** 2 + (y - x) ** 2 + (z - 2) ** 2


m = Minuit(f, print_level=0, pedantic=False)
m.migrad()
m.draw_mncontour('x', 'y')
```



mnprofile (*self*, *vname*, *bins=30*, *bound=2*, *subtract_min=False*)
Calculate minos profile around the specified range.

That is Migrad minimum results with **vname** fixed at various places within **bound**.

Arguments:

- **vname** name of variable to scan
- **bins** number of scanning bins. Default 30.
- **bound** If bound is tuple, (left, right) scanning bound. If bound isa number, it specifies how many σ symmetrically from minimum (minimum+- bound* σ). Default 2
- **subtract_min** subtract_minimum off from return value. This makes it easy to label confidence interval. Default False.

Returns:

bins(center point), value, migrad results

narg
Number of arguments

ncalls
Number of FCN call of last migrad / minos / hesse run.

np_matrix (*self*, *correlation=False*, *skip_fixed=True*)
Error or correlation matrix in numpy array format.

The name of this function was chosen to be analogous to `matrix()`, it returns the same information in a different format.

Note that a `numpy.ndarray` is returned, not a `numpy.matrix`

parameters
Parameter name tuple

pos2var
Map variable position to name

print_all_minos(*self*)
Print all minos errors (and its states)

print_fmin(*self*)
Print current function minimum state

print_initial_param(*self*, ***kwds*)
Print initial parameters

print_level
print_level: object Print level.

- 0: quiet
- 1: print stuff the end
- 2: 1+fit status during call

Yes I know the case is wrong but this is to keep it compatible with PyMinuit.

print_matrix(*self*, ***kwds*)
Show error_matrix

print_param(*self*, ***kwds*)
Print current parameter state.

Extra keyword arguments will be passed to frontend.print_param.

profile(*self*, *vname*, *bins=100*, *bound=2*, *args=None*, *subtract_min=False*)
Calculate cost function profile around specify range.

Arguments:

- **vname** variable name to scan
- **bins** number of scanning bin. Default 100.
- **bound** If bound is tuple, (left, right) scanning bound. If bound is a number, it specifies how many σ symmetrically from minimum (minimum+- bound* σ). Default 2
- **subtract_min** subtract_minimum off from return value. This makes it easy to label confidence interval. Default False.

Returns:

bins(center point), value

See also:

mnprofile()

set_errordef(*self*, *double errordef*)
Set error parameter 1 for χ^2 and 0.5 for log likelihood.

See page 37 of http://hep.fi.infn.it/minuit.pdf

set_print_level(*self*, *lvl*)
Set print level.

- 0 quiet
- 1 normal
- 2 paranoid
- 3 really paranoid

set_strategy (*self*, *value*)
Set strategy.

- 0 = fast
- 1 = default
- 2 = slow but accurate

set_up (*self*, *double errordef*)
Alias for *set_errordef()*

strategy
strategy: 'unsigned int' Strategy integer code.

- 0 fast
- 1 default
- 2 slow but accurate

var2pos
Map variable name to position

2.2.3 Utility Functions

:mod:util module provides describe function and various function to manipulate fitarguments. IMinuit utility functions and classes.

iminuit.util.**describe** (*f*, *verbose=False*)
Try to extract the function argument names.

See also:

Function Signature Extraction Ordering

class iminuit.util.**Struct** (***kwds*)
A Struct is a Python dict with tab completion.

Example:

```
>>> s = Struct(a=42)
>>> s['a']
42
>>> s.a
42
```

iminuit.util.**fitarg_rename** (*fitarg*, *ren*)
Rename variable names in `fitarg` with rename function.

```
#simple renaming
fitarg_rename({'x':1, 'limit_x':1, 'fix_x':1, 'error_x':1},
    lambda pname: 'y' if pname=='x' else pname)
#{'y':1, 'limit_y':1, 'fix_y':1, 'error_y':1},

#prefixing
```

```
    figarg_rename({'x':1, 'limit_x':1, 'fix_x':1, 'error_x':1},
        lambda pname: 'prefix_'+pname)
    #{'prefix_x':1, 'limit_prefix_x':1, 'fix_prefix_x':1, 'error_prefix_x':1}
```

iminuit.util.**true_param**(*p*)
Check if p is a parameter name, not a limit/error/fix attributes.

iminuit.util.**param_name**(*p*)
Extract parameter name from attributes.

Examples:

- fix_x -> x
- error_x -> x
- limit_x -> x

iminuit.util.**extract_iv**(*b*)
Extract initial value from fitargs dictionary.

iminuit.util.**extract_limit**(*b*)
Extract limit from fitargs dictionary.

iminuit.util.**extract_error**(*b*)
Extract error from fitargs dictionary.

iminuit.util.**extract_fix**(*b*)
extract fix attribute from fitargs dictionary

iminuit.util.**remove_var**(*b*, *exclude*)
Exclude variable in exclude list from b.

iminuit.util.**arguments_from_docstring**(*doc*)
Parse first line of docstring for argument name.

Docstring should be of the form `min(iterable[, key=func])`.

It can also parse cython docstring of the form `Minuit.migrad(self[, int ncall_me =10000, resume=True, int nsplit=1])`

2.2.4 Return Value Struct

iminuit uses various structs as return value. This section lists the struct and all it's field

2.2.5 Function Minimum Struct

They are usually return value from `Minuit.get_fmin()` and `Minuit.migrad()` Function Mimum Struct has the following attributes:

- *fval*: FCN minimum value
- *edm*: Estimated Distance to Minimum
- *nfcn*: Number of function call in last mimizier call
- *up*: UP parameter. This determine how minimizer define 1 σ error
- ***is_valid*: Validity of function minimum. This is defined as**
  - has_valid_parameters

  - and not has_reached_call_limit
  - and not is_above_max_edm

- ***has_valid_parameters*: Validity of parameters. This means:**
  1. The parameters must have valid error(if it's not fixed). Valid error is not necessarily accurate.
  2. The parameters value must be valid
- *has_accurate_covariance*: Boolean indicating whether covariance matrix is accurate.
- *has_pos_def_covar*: Positive definiteness of covariance
- *has_made_posdef_covar*: Whether minimizer has to force covariance matrix to be positive definite by adding diagonal matrix.
- *hesse_failed*: Successfulness of the hesse call after minimizer.
- *has_covaraince*: Has Covariance.
- *is_above_max_edm*: Is EDM above 0.0001*tolerance*up? The convergence of migrad is defined by EDM being below this number.
- *has_reached_call_limit*: Whether the last minimizer exceeds number of FCN calls it is allowd.

2.2.6 Minos Error Struct

Minos Error Struct is used in return value from `Minuit.minos()`. You can also call `Minuit.get_merrors()` to get accumulated dictionary all minos errors that has been calculated. It contains various minos status:

- *lower*: lower error value
- *upper*: upper error value
- *is_valid*: Validity of minos error value. This means *lower_valid* and *upper_valid*
- *lower_valid*: Validity of lower error
- *upper_valid*: Validity of upper error
- *at_lower_limit*: minos calculation hits the lower limit on parameters
- *at_upper_limit*: minos calculation hits the upper limit on parameters
- *lower_new_min*: found a new minimum while scanning cost function for lower error value
- *upper_new_min*: found a new minimum while scanning cost function for upper error value
- *nfn*: number of call to FCN in the last minos scan
- *min*: the value of the parameter at the minimum

2.2.7 Minuit Parameter Struct

Minuit Parameter Struct is return value from `Minuit.hesse()` You can, however, access the latest parameter by calling `Minuit.get_param_states()`. Minuit Parameter Struct has the following attrubutes:

- *number*: parameter number
- *name*: parameter name
- *value*: parameter value
- *error*: parameter parabolic error(like those from hesse)

- *is_fixed*: is the parameter fixed
- *is_const*: is the parameter a constant(We do not support const but you can alway use fixing parameter instead)
- *has_limits*: parameter has limits set
- *has_lower_limit*: parameter has lower limit set. We do not support one sided limit though.
- *has_upper_limit*: parameter has upper limit set.
- *lower_limit*: value of lower limit for this parameter
- *upper_limit*: value of upper limit for this parameter

2.2.8 Function Signature Extraction Ordering

1. Using `f.func_code.co_varnames`, `f.func_code.co_argcount` All functions that is defined like:

```
def f(x,y):
    return (x-2)**2+(y-3)**2
```

or:

```
f = lambda x,y: (x-2)**2+(y-3)**2
```

Has these two attributes.

2. Using `f.__call__.func_code.co_varnames`, `f.__call__.co_argcount`. Minuit knows how to dock off self parameter. This allow you to do things like encapsulate your data with in fitting algorithm:

```
class MyChi2:
    def __init__(self, x, y):
        self.x, self.y = (x,y)
    def f(self, x, m, c):
        return m*x + c
    def __call__(self,m,c):
        return sum([(self.f(x,m,c)-y)**2
                    for x,y in zip(self.x ,self.y)])
```

3. If all fail, Minuit will call `inspect.getargspec` for function signature. Builtin C functions will fall into this category since they have no signature information. `inspect.getargspec` will parse docstring to get function signature.

This order is very similar to PyMinuit signature detection. Actually, it is a superset of PyMinuit signature detection. The difference is that it allows you to fake function signature by having func_code attribute in the object. This allows you to make a generic functor of your custom cost function. This is how probfit was written:

```
f = lambda x,m,c: m*x+c
#the beauty here is that all you need to build
#a Chi^2 is just your function and data
class GenericChi2:
    def __init__(self, f, x, y):
        self.f = f
        args = describe(f)#extract function signature
        self.func_code = Struct(
                co_varnames = args[1:],#dock off independent param
                co_argcount = len(args)-1
            )
```

```
        def __call__(self, *arg):
            return sum((self.f(x,*arg)-y)**2 for x,y in zip(self.x, self.y))

    m = Minuit(GenericChi2(f,x,y))
```

Note: If you are unsure what minuit will parse your function signature as , you can use *describe()* which returns tuple of argument names minuit will use as call signature.

CHAPTER 3

Tutorial

All the tutorials are in tutorial directory. You can view them online too:

- Quick start
- Hard Core Cython tutorial. If you need to do a huge likelihood fit that needs speed, this is for you. If you don't care, just use probfit. It's a fun read though I think.

CHAPTER 4

API

See *Full API Documentation*

CHAPTER 5

Technical Stuff

Using it as a black box is a bad idea. Here are some fun reads; the order is given by the order I think you should read.

- Wikipedia for Quasi Newton Method and DFP formula. The magic behind MIGRAD.
- Variable Metric Method for Minimization William Davidon 1991
- A New Approach to Variable Metric Algorithm (R.Fletcher 1970)
- Original Paper: MINUIT - A SYSTEM FOR FUNCTION MINIMIZATION AND ANALYSIS OF THE PARAMETER ERRORS AND CORRELATIONS by Fred James and Matts Roos.

CHAPTER 6

You can help

Github allows you to contribute to this project very easily just fork the repository, make changes and submit a pull request.

Here's the list of concrete open issues and feature requests: https://github.com/iminuit/iminuit

More generally any contribution to the docs, tests and package itself is welcome!

- Documentation. Tell us what's missing, what's incorrect or misleading.
- Tests. If you have an example that shows a bug or problem, please file an issue!
- Performance. If you are a C/cython/python hacker go ahead and make it faster.

i

iminuit.util, 18

A

args (iminuit.Minuit attribute), 9
arguments_from_docstring() (in module iminuit.util), 19

C

contour() (iminuit.Minuit method), 10
covariance (iminuit.Minuit attribute), 10

D

describe() (in module iminuit.util), 18
draw_contour() (iminuit.Minuit method), 11
draw_mncontour() (iminuit.Minuit method), 11
draw_mnprofile() (iminuit.Minuit method), 12
draw_profile() (iminuit.Minuit method), 13

E

edm (iminuit.Minuit attribute), 10
errordef (iminuit.Minuit attribute), 10
errors (iminuit.Minuit attribute), 9
extract_error() (in module iminuit.util), 19
extract_fix() (in module iminuit.util), 19
extract_iv() (in module iminuit.util), 19
extract_limit() (in module iminuit.util), 19

F

fitarg (iminuit.Minuit attribute), 9
fitarg_rename() (in module iminuit.util), 18
fval (iminuit.Minuit attribute), 10

G

gcc (iminuit.Minuit attribute), 10
get_fmin() (iminuit.Minuit method), 13
get_initial_param_state() (iminuit.Minuit method), 13
get_merrors() (iminuit.Minuit method), 13
get_num_call_fcn() (iminuit.Minuit method), 13
get_param_states() (iminuit.Minuit method), 14
grad_fcn (iminuit.Minuit attribute), 14

H

hesse() (iminuit.Minuit method), 9

I

iminuit.util (module), 18
is_clean_state() (iminuit.Minuit method), 14
is_fixed() (iminuit.Minuit method), 14

L

latex_initial_param() (iminuit.Minuit method), 14
latex_matrix() (iminuit.Minuit method), 14
latex_param() (iminuit.Minuit method), 14
list_of_fixed_param() (iminuit.Minuit method), 14
list_of_vary_param() (iminuit.Minuit method), 14

M

matrix() (iminuit.Minuit method), 14
matrix_accurate() (iminuit.Minuit method), 14
merrors (iminuit.Minuit attribute), 10, 14
merrors_struct (iminuit.Minuit attribute), 14
migrad() (iminuit.Minuit method), 9
migrad_ok() (iminuit.Minuit method), 14
minos() (iminuit.Minuit method), 9
Minuit (class in iminuit), 7
mncontour() (iminuit.Minuit method), 14
mncontour_grid() (iminuit.Minuit method), 15
mnprofile() (iminuit.Minuit method), 16

N

narg (iminuit.Minuit attribute), 16
ncalls (iminuit.Minuit attribute), 16
np_matrix() (iminuit.Minuit method), 16

P

param_name() (in module iminuit.util), 19
parameters (iminuit.Minuit attribute), 17
pos2var (iminuit.Minuit attribute), 17
print_all_minos() (iminuit.Minuit method), 17
print_fmin() (iminuit.Minuit method), 17
print_initial_param() (iminuit.Minuit method), 17
print_level (iminuit.Minuit attribute), 17
print_matrix() (iminuit.Minuit method), 17
print_param() (iminuit.Minuit method), 17

profile() (iminuit.Minuit method), 17

R

remove_var() (in module iminuit.util), 19

S

set_errordef() (iminuit.Minuit method), 17
set_print_level() (iminuit.Minuit method), 17
set_strategy() (iminuit.Minuit method), 18
set_up() (iminuit.Minuit method), 18
strategy (iminuit.Minuit attribute), 18
Struct (class in iminuit.util), 18

T

tol (iminuit.Minuit attribute), 10
true_param() (in module iminuit.util), 19

V

values (iminuit.Minuit attribute), 9
var2pos (iminuit.Minuit attribute), 18