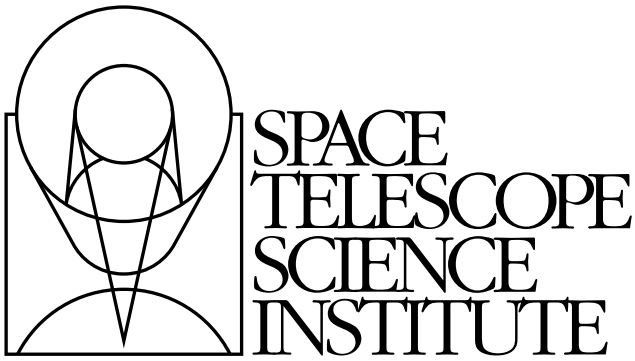

nictools Documentation

Release 1.0.6 (14-Aug-2012)

Warren Hack, Nadia Dencheva, Chris Sontag, Megan Sosey, Micha

May 03, 2013

CONTENTS

This package provides data processing tools for working with NICMOS data.

Contents:

CHAPTER

ONE

SAACLEAN

saaclean: Module for estimating and removing persistent CR signal due to a prior
SAA passage.

Usage
Normally used via the STSDAS task saaclean in the nicmos package. To use as pure python, create a params object to override any of the default parameters if desired, then invoke clean:

```
>>> mypars=saaclean.params(thresh=0.23)
>>> saaclean.clean('inputfile.fits','outputfile.fits',pars=mypars)
```

For more information
Additional user information, including parameter definitions and more examples, can be found in the help file for the STSDAS saaclean task, located in nicmos$doc/saaclean.hlp.

The algorithm and IDL prototype are described in the NICMOS ISR 2003-009, by Bergeron and Dickinson, available through the NICMOS webpage.

Dependencies

- numpy 1.0.2.dev3534 or higher
- pyfits v1.1b4 or higher
- imagestats v1.3 or higher

exception nictools.saaclean.**AlreadyDone**

exception nictools.saaclean.**BadThreshError**

exception nictools.saaclean.**InsuffImprovement**

exception nictools.saaclean.**NegScaleError**

exception nictools.saaclean.**NoPersistError**

class nictools.saaclean.**Domain**(*name*, *pixellist*, *range*)
Stores a list of pixels for a (typically high or low) signal domain

getmin()

striplowerthan(*factor*)
self.pp is defined in Exposure.getscales It contains the (bin, stddev, mode) for the statistical analysis.

striplowerthan(factor) examines the stddev column only, and replaces all values of the stddev that are less than factor*the zeroth bin, with the maximum stddev.

writeto(*filename*, *clobber=False*)

class nictools.saaclean.**Exposure**(*imgfile*, *nickname=None*)

Stores a collection of keywords and the image data for an exposure.

apply_domains(*saaper*, *badmask*, *noisethresh*, *appimage=None*)

apply_mask(*mask*)

dark_subtract(*dark*)

getmask(*dim=256*, *border=3*, *writename='mask.dat'*, *clobber=False*)

Computes a mask to use for pixels to omit

getscales(*saaper*, *mask*, *pars*)

pedskyish()

Performs something like the IRAF pedsky task, but with a bit more sophistication in handling the central row and column

update_header(*pars*, *tag='default'*, *header=None*)

Update the FITS header with all this good stuff we've done

writeto(*outname*, *clobber=False*)

class nictools.saaclean.**params**(*scale=0.54*, *wf1=0.7*, *wf2=0.3*, *stepsize=0.008*, *thresh=None*, *hirange=0.4*, *lorange=0.25*, *dofit=1*, *crthresh=0.3*, *noisethresh=1.0*, *binsigfrac=0.3*, *readsaaper='False'*, *writesaaper='True'*, *saaperfile='saaper.fits'*, *fitthresh='True'*, *histbinwidth=0.01*, *nclip=10*, *clobber='False'*, *flatsaaper='True'*, *flatsaaperfile=None*, *maskfile=None*, *darkpath=None*, *diagfile=None*)

nictools.saaclean.**clean**(*usr_calcfile*, *usr_targfile*, *usr_outfile*, *pars=None*)

nictools.saaclean.**create_saaper_header**(*im1*, *im2*, *saaper*)

nictools.saaclean.**flat_saaper**(*saaper*, *img*)

nictools.saaclean.**gausspoly_eval**(*coeffs*, *t*)

nictools.saaclean.**gausspoly_fit**(*thedata*, *guesscoeff*)

nictools.saaclean.**gausspoly_model**(*coeffs*, *t*)

nictools.saaclean.**get_dark_data**(*imgfile*, *darkpath*)

nictools.saaclean.**get_postsaa_darks**(*imgfile*)
Return the filenames containing the post-saa dark exposures, if present. Otherwise raise an exception and exit.

nictools.saaclean.**getdark**(*camera*, *tdkfile*, *darkpath*)
Get the right dark file for a given NICMOS camera. This is definitely not the right way to do this.

nictools.saaclean.**make_saaper**(*imgfile*, *pars*, *crthresh=1*)

nictools.saaclean.**median**(*a*)

nictools.saaclean.**osfn**(*filename*)
Return a filename with iraf syntax and os environment names substituted out

nictools.saaclean.**parabola_min**(*thedata*, *startguess*)

nictools.saaclean.**parabola_model**(*coeffs*, *t*)

nictools.saaclean.**smartopen**(*fname*, *mode*, *clobber=True*)
Allows specifying a clobber behavior

nictools.saaclean.**thresh_from_gausspoly_fit**(*saa*, *parbinwidth=0.5*, *nclip=3*, *diagfile=None*, *clobber=False*)
Some massaging of the SAApersistence image histogram is performed in order to obtain an optimal fit. Unfortunately this involves some magic numbers taken from the IDL code.

nictools.saaclean.**writeimage**(*image*, *filename*, *header=None*, *comment=None*, *clobber=False*)

CHAPTER

TWO

NIC_REM_PERSIST

class nictools.nic_rem_persist.**NicRemPersist**(*calcfile*, *targfile*, *verbosity=1*, *persist_lo=None*, *used_lo=None*, *persist_model=None*, *persist_mask=None*, *run_stdout=None*)

Remove bright earth persistence persistence from CAL file of NICMOS data (on which pedsub has been run) using a med

based on correction calculated from PED file.

Notes

The syntax for using this class under Python or PyRAF:

```
--> nrp = nic_rem_persist.NicRemPersist('n9r7b2bjq_ped.fits', persist_model = 'persistring.fits'
          used_lo = .3, persist_lo = 0.8)
--> nrp.persist()
```

The linux command line syntax for calling this module:

```
hal> ./nic_rem_persist.py 'n9r7b2bjq_ped.fits','n9r7b2bjq_cal.fits' -d 'same_as_persistring.fits
```

The full description of the parameters required by this class is as follows.

constructor

Parameters

calcfile : string

name of ped file

targfile : string

name of cal file

persist_lo : float

minimum allowed value of the persistence

used_lo : float

minimum allowed value of the fraction of pixels used

persist_model : string

filename containing persistence frame (ring median of)

persist_mask : string

filename containing pixel mask

run_stdout : file handle

open trailer file (pipeline use only)

persist()
remove persistence due to the full bright Earth.

print_pars()
Print input parameters and calculated values.

nictools.nic_rem_persist.**check_cl_pars**(*calcfile*, *targfile*, *persist_lo*, *used_lo*)
When run from linux command line, verify that each parameter is valid.

Parameters
calcfile : string

name of ped file

targfile : string

name of cal file

persist_lo : float

minimum allowed value of the persistence

used_lo : float

minimum allowed value of the fraction of pixels used

Returns
persist_lo, used_lo : float, float

nictools.nic_rem_persist.**check_py_pars**(*self*, *calcfile*, *targfile*, *persist_lo*, *used_lo*, *persist_model*, *persist_mask*)
When run under python, check validity of input parameters. For unspecified *_lo parameters, the user will be given the option of typing in a value or accepting the default value. For an unspecified model(mask) file, will try to get file name from PMODFILE(PMSKFILE) from input file header, otherwise will get default value from persutil.

Parameters
calcfile : string

name of PED file

targfile : string

name of CAL file

persist_lo : float

minimum allowed value of the persistence

used_lo : float

minimum allowed value of the fraction of pixels used

persist_model : string

filename containing persistence frame (ring median of)

persist_mask : string

filename containing pixel mask

Returns
persist_lo, used_lo : float

persist_model, persist_mask : string

nictools.nic_rem_persist.**iterstatc**(*clip*, *d*)

version of nicmos iterstat: calculate sigma-clipped mean
and standart deviation

Parameters
clip : float

number of std to use in sigma clipping

d : float

array of values to sigma clip

Returns
clipped mean, clipped std : float, float

nictools.nic_rem_persist.**make_footprint**(*rin*, *rout*)

Make an annular mask footprint for use in ndimage.median_filter to create a ring median image.
Sets all pixels between rin and rout to 1.

Parameters
rin : int

inner radius

rout : int

outer radius

Returns
mask : ndarray

nictools.nic_rem_persist.**median**(*y*, *mask*)
Return the median of the array y, ignoring masked elements.

Parameters
y : float

array of values [2d]

mask : int

array of ones or zeros (0 indicates a good value) [2d]

Returns
median_y : float

median of y, ignoring masked elements

CHAPTER

THREE

CALTEMPFROMBIAS

class nictools.CalTempFromBias.**CalTempFromBias**(*input_file*, *edit_type=None*, *hdr_key=None*, *err_key=None*, *nref_par=None*, *force=None*, *noclean=False*, *dry_run=1*, *verbosity=0*)

Calculate the temperature from the bias for a given filename.

Notes

Basic syntax for using this class is:

```
tfb = CalTempFromBias( filename, edit_type=edit_type, hdr_key=hdr_key, err_key=err_key,
      nref_par=nref_par, force=force, noclean=noclean, dry_run=dry_run, verbosity=verbosity)
[temp, sigma, winner, in_flag, dry_run ]= tfb.calctemp()
tfb.print_pars()
stat = tfb.update_header( temp, sigma, winner)
```

The full set of parameters for the methods:

constructor

Parameters

input_file : string

name of the file or filelist to be processed

edit_type : string type of file to update

hdr_key : string

name of keyword to update in file

err_key : string

name of keyword for error estimate

nref_par : string

name of the directory containing the nonlinearity file

force : string

name of algorithm whose value is to be returned

noclean : {'True', 'False'}

flag to force use of UNCLEANed 0th read.

dry_run : {0,1} [Default: 1]

flag to force not writing to header

verbosity : {0,1,2}

verbosity level (0 for quiet, 1 verbose, 2 very verbose)

calctemp()

Calculate the temperature from the bias for the given input file

Returns

temp : float

sig : float

winner : int

in_flag : str

print_pars()

Print parameters used.

update_header(*temp*, *sig*, *winner*, *edit_type=None*, *hdr_key=None*, *err_key=None*)

Update header method

Parameters

temp : float

calculated temperature

sig : float

standard deviation of calculated temperature

winner : int

algorithm used

edit_type : string

type of file to be updated

hdr_key : string

name of keyword to update in file

err_key : string

name of keyword for error estimate

Returns

status : int (not None for failure due to key not being specified)

nictools.CalTempFromBias.**do_blind**(*camera*, *quads*, *verbosity*)

Calculate temperature using the blind correction

Parameters

camera : int

number of camera

quads : float

value of quad from quadmean

verbosity : int

level of veborsity

Returns

temp, sig : float, float

nictools.CalTempFromBias.**do_quietest**(*camera*, *quads*, *verbosity*)

Calculate temperature using the quietest quad correction

Parameters

camera : int

number of camera

quads : float

value of quad from quadmean

verbosity : int

level of veborsity

Returns

temp, sig : float

nictools.CalTempFromBias.**poly**(*var_x*, *coeffs*)

Return linear polynomial with given coefficients.

var_x

[scalar, vector, or array] input argument

coeffs

[float] vector of polynomial coefficients

nictools.CalTempFromBias.**quadmean**(*im*, *border*)

This function computes the mean in the 4 quadrants of an input NxM array

Parameters

im : ndarray

input rectangular array

border : int

border size (in pixels) aroud the perimeter of each quad to be excluded from the mean

Returns

quads : float

Notes

Following Eddie's convention, the quads are numbered as follows:

```
|------|-----|
|  Q4  |  Q3 |
|      |     |   as seen in the standard NICMOS/HST data format.
|------|-----|
|  Q1  |  Q2 |
|      |     |
|------|-----|
```

optionally, you can specify a border, in pixels, around the perimeter of EACH QUAD to be excluded from the mean

CHAPTER

FOUR

FINESKY

class nictools.finesky.**Makemedmask**(*thresh=None*, *medfile=None*, *callist=None*, *verbosity=0*)

Create and output a median mask from cal files and blt files

Notes

Syntax for using this class:

```
m_mask = finesky.Makemedmask( medfile='medout2.fits', callist='/hal/data2/dev/nicmos_ped/inlist1
         thresh = 0.7,verbosity = 1)
m_mask.makemask()
```

Full set of parameters for class methods are as follows.

constructor

Parameters

thresh : real

threshold used in making mask

medfile : string

name of output masked median image

callist : string

name of text file containing cal file names

verbosity : {0,1,2}

verbosity level (0 for quiet, 1 verbose, 2 very verbose)

makemask()

Make and output mask

print_pars()

Print parameters.

nictools.finesky.**write_to_file**(*data*, *filename*, *hdr*, *verbosity*)

Write data to specified filename with specified header

Parameters

data : ndarray

numpy array

filename : string

name of output file

hdr : pyfits Header object

header for output file

verbosity : {0,1,2}

verbosity level (0 for quiet, 1 verbose, 2 very verbose)

CHAPTER

FIVE

PUFTCORR

puftcorr: Module for estimating and removing "Mr. Staypuft" signal from
a NICMOS exposure.

Usage

Normally used via the STSDAS task puftcorr in the nicmos package. To use as pure python, just invoke the clean method:

```
>>> puftcorr.clean('inputfile.fits','outputfile.fits')
```

For more information

Additional user information, including parameter definitions and more examples, can be found in the help file for the STSDAS puftcorr task, located in nicmos$doc/puftcorr.hlp.

The algorithm and IDL prototype were developed by L.Bergeron, but never made publicly available.

Dependencies

- numpy v1.0.2dev3534 or higher
- pyfits v1.1b4 or higher
- convolve v2.0 or higher
- ndimage v2.0 or higher

exception nictools.puftcorr.**NoPuftError**

class nictools.puftcorr.**InputFile**(*imgfile*)
Stores a collection of keywords and the header for an exposure.

class nictools.puftcorr.**Readout**(*input*, *sampnum*)

class nictools.puftcorr.**params**(*camera*)

nictools.puftcorr.**clean**(*usr_imgfile*, *usr_outfile*)

nictools.puftcorr.**get_corr**(*im*, *pars*)

nictools.puftcorr.**get_totsig**(*im*, *la*)

nictools.puftcorr.**osfn**(*filename*)
Return a filename with iraf syntax and os environment names substituted out

CHAPTER

SIX

RNLINCORR

rnlincor: Module to correct for the countrate-dependent nonlinearity in
a NICMOS image.

Usage

Normally used via the STSDAS task rnlincor in the nicmos package. To use as pure python, just invoke the run method:

```
>>> rnlincor.run('inputfile.fits','outputfile.fits')
```

It may also be run from the shell:

```
% rnlincor.py infile.fits [outfile.fits] [--nozpcorr]
```

For more information

Additional user information, including parameter definitions and more examples, can be found in the help file for the STSDAS rnlincor task, located in nicmos$doc/rnlincor.hlp.

This task is based on prototype code developed by R. de Jong. The algorithm is described in more detail in ISR NICMOS 2006-003 by de Jong.

Dependencies

- numpy v1.0.2dev3534 or higher
- pyfits v1.1b4 or higher

class nictools.rnlincor.**FitsRowObject**(*fitsrecord*)
Class to facilitate working with single table rows.

nictools.rnlincor.**check_infile**(*infile*)
Open the input file and check all the things that can go wrong. If we pass all the tests, return the handle to the open file to pass to the main routine.

nictools.rnlincor.**expandname**(*pattern*)
Select the latest file that matches the pattern in the directory specified in the pattern

nictools.rnlincor.**getcurve**(*fname*, *col1='wavelength'*, *col2='correction'*, *pad=0*)
Gets a 2-column table from the first extension of a FITS file. Defaults to "wavelength" and "correction" for column names, but others can be specified. If pad keyword is nonzero, the wavelength table will be extended by the pad amount in each direction.

nictools.rnlincor.**getrow**(*photmode*, *corrname*)
Pick out the correction parameters from the proper row of the table located in corrname, based on photmode. Return an object that contains the row fields as attributes.

nictools.rnlincor.**parrun**(*parfile*)

nictools.rnlincor.**rnlincor**(*infile*, *outfile*, ***opt*)
The main routine

nictools.rnlincor.**run**(**args*, ****inopt*)

nictools.rnlincor.**set_default_options**(*inopt*)

nictools.rnlincor.**update_data**(*f*, *imgext*, *img*, *mul*)

nictools.rnlincor.**update_header**(*f*, *alpha*, *zpcorr*, *zpratio=None*)
Update all the header keywords

CHAPTER

SEVEN

READTDD

The readTDD.py is a helper module used to extract the linear dark and amp glow components from a NICMOS time dependent dark file.

author
Christopher Hanley

dependencies
stsci.tools.fileutil

class nictools.readTDD.**darkobject**(*hdulist*)
darkobject: This class takes as input a pyfits hdulist object. The linear dark and amp glow noise componenets are then extracted from the hdulist.

getampglow()
getampglow: darkobject method which us used to return the amp glow component from a NICMOS temperature dependent dark file.

getampglowheader()
getampglowheader: darkobject method used to return the header information of the amp glow entension of a TDD file.

getlindark()
getlindark: darkobject method which is used to return the linear dark component from a NICMOS temperature dependent dark file.

getlindarkheader()
getlindarkheader: darkobject method used to return the header information of the linear dark entension of a TDD file.

nictools.readTDD.**fromcalfile**(*filename*)
fromcalfile: function that returns a darkobject instance given the name of a cal.fits file as input. If there is no TEMPFILE keyword in the primary header of the cal.fits file or if the file specified by TEMPFILE cannot be found, a None object is returned.

CHAPTER

EIGHT

MAKEMEDMASK

class nictools.makemedmask.**Makemedmask**(*thresh=None*, *medfile=None*, *callist=None*, *verbosity=0*)

Create and output a median mask from cal files and blt files

Notes

Syntax for using this class:

```
m_mask = makemedmask.Makemedmask( medfile='medout2.fits', callist='/hal/data2/dev/nicmos_ped/inl
        thresh = 0.7,verbosity = 1)
m_mask.makemask()
```

Full signatures for methods are as follows:

constructor

Parameters

thresh : float

threshold used in making mask

medfile : string

name of output masked median image

callist : string

name of text file containing cal file names

verbosity : {0,1,2}

verbosity level (0 for quiet, 1 verbose, 2 very verbose)

makemask()

Make and output mask

print_pars()

Print parameters.

nictools.makemedmask.**write_to_file**(*data*, *filename*, *hdr*, *verbosity*)

Write data to specified filename with specified header

Parameters

data : ndarray

numpy array

filename : string

name of output file

hdr : pyfits Header object

header for output file

verbosity : {0,1,2}

verbosity level (0 for quiet, 1 verbose, 2 very verbose)

CHAPTER

NINE

UTILITIES

There are a number of modules which provide utility functions for use in this package.

9.1 fsutil

nictools.fsutil.**all_printMsg**(*message*, *level=1*)

Parameters
message : string

message to print

level : int

verbosity level

nictools.fsutil.**checkVerbosity**(*level*)

Parameters
level : int

level of verbosity

Returns
level : bool

true if verbosity is at least as great as level.

nictools.fsutil.**printMsg**(*message*, *level=0*)

Parameters
message : string

message to print

level : int

verbosity level

nictools.fsutil.**setCallist**(*callist_value*)
Copy callist to a variable that is global for this file.

Parameters
callist_value : string

name of file listing cal files

nictools.fsutil.**setMedfile**(*medfile_value*)
Copy medfile to a variable that is global for this file.

Parameters
medfile_value : string

name of output file for masked median image

nictools.fsutil.**setThresh**(*thresh_value*)
Copy thresh to a variable that is global for this file.

Parameters
thresh_value : float

level of threshold

nictools.fsutil.**setVerbosity**(*verbosity_level*)
Copy verbosity to a variable that is global for this file.

Parameters
verbosity_level : int

level of verbosity

9.2 opusutil

nictools.opusutil.**CloseTrl**()

nictools.opusutil.**FileToList**(*filename*)

nictools.opusutil.**OpenTrl**(*filespec*)

nictools.opusutil.**PrintMsg**(*level*, *msg*, *module_name=''*)

nictools.opusutil.**RemoveIfThere**(*filename*)

nictools.opusutil.**ResourceToMap**(*filename*)

nictools.opusutil.**StretchFile**(*stretched_filename*)

nictools.opusutil.**UsingLvl**(*level*)

9.3 persutil

nictools.persutil.**all_printMsg**(*message*, *level=1*)

nictools.persutil.**checkVerbosity**(*level*)
Return true if verbosity is at least as great as level.

nictools.persutil.**getOptions**()

nictools.persutil.**getPersist_lo**(*calcfile*)

Get value of persist_lo from BEPVALLO in the persistence model file PMODFILE or from a default specified by this module. This will only be called if user did not specify a value on the command-line.

Parameters

calcfile : string

input ped file

Returns

persist_lo : float

nictools.persutil.**getPersist_mask**(*calcfile*)

Get name of persistence mask from PMSKFILE in the input file This will only be called if user did not specify a model on the command-line.

Parameters

calcfile : string

input ped file

Returns

persist_mask : string

nictools.persutil.**getPersist_model**(*calcfile*)

Get name of persistence model from PMODFILE in the input file. This will only be called if user did not specify a model on the command-line.

Parameters

calcfile : string

input ped file

Returns

persist_model : string

nictools.persutil.**getUsed_lo**(*calcfile*)

Get value of used_lo from BEPUSELO in the persistence model file PMODFILE or from a default specified by this module. This will only be called if user did not specify a value on the command-line.

Parameters

calcfile : string

input ped file

Returns

used_lo : float

nictools.persutil.**printMsg**(*message*, *level=0*)

nictools.persutil.**setVerbosity**(*verbosity_level*)

Copy verbosity to a variable that is global for this file. argument: verbosity_level - an integer value indicating the level of verbosity

9.4 SP_FirstDerivatives

Automatic differentiation for functions with any number of variables

Instances of the class DerivVar represent the values of a function and its partial X{derivatives} with respect to a list of variables. All common mathematical operations and functions are available for these numbers. There is no restriction on the type of the numbers fed into the code; it works for real and complex numbers as well as for any Python type that implements the necessary operations.

This module is as far as possible compatible with the n-th order derivatives module Derivatives. If only first-order derivatives are required, this module is faster than the general one.

Example:

```
print sin(DerivVar(2))
```

produces the output:

```
(0.909297426826, [-0.416146836547])
```

The first number is the value of sin(2); the number in the following list is the value of the derivative of sin(x) at x=2, i.e. cos(2).

When there is more than one variable, DerivVar must be called with an integer second argument that specifies the number of the variable.

Example:

```
>>>x = DerivVar(7., 0)
>>>y = DerivVar(42., 1)
>>>z = DerivVar(pi, 2)
>>>print (sqrt(pow(x,2)+pow(y,2)+pow(z,2)))

produces the output

>>>(42.6950770511, [0.163953328662, 0.98371997197, 0.0735820818365])
```

The numbers in the list are the partial derivatives with respect to x, y, and z, respectively.

Note: It doesn't make sense to use DerivVar with different values for the same variable index in one calculation, but there is no check for this. I.e.:

```
>>>print DerivVar(3, 0)+DerivVar(5, 0)

produces

>>>(8, [2])
```

but this result is meaningless.

class nictools.SP_FirstDerivatives.**DerivVar**(*value*, *index=0*, *order=1*)

Numerical variable with automatic derivatives of first order

Parameters

value : int or float

the numerical value of the variable

index : int

the variable index, which serves to distinguish between variables and as an index for the derivative lists. Each explicitly created instance of DerivVar must have a unique index.

order : int

the derivative order, must be zero or one

Raises :

——— :

ValueError: if order < 0 or order > 1 :

arccos()

arcsin()

arctan()

arctan2(*other*)

cos()

cosh()

exp()

gamma()

log()

log10()

sign()

sin()

sinh()

sqrt()

tan()

tanh()

nictools.SP_FirstDerivatives.**DerivVector**(*x*, *y*, *z*, *index=0*)

Parameters

x : float or int

x component of the vector

y : float or int

y component of the vector

z : float or int

z component of the vector

index : int

the DerivVar index for the x component. The y and z components receive consecutive indices.

Returns

vector : Scientific.Geometry.VectorModule.Vector

a vector whose components are DerivVar objects

nictools.SP_FirstDerivatives.**isDerivVar**(*x*)

Parameters

x : :

an arbitrary object

Returns

result : bool

True if x is a DerivVar object, False otherwise

9.5 SP_LeastSquares

Non-linear least squares fitting

9.5.1 Examples

Usage example:

```
from Scientific.N import exp

def f(param, t):
    return param[0]*exp(-param[1]/t)

data_quantum = [(100, 3.445e+6),(200, 2.744e+7),
                (300, 2.592e+8),(400, 1.600e+9)]
data_classical = [(100, 4.999e-8),(200, 5.307e+2),
                  (300, 1.289e+6),(400, 6.559e+7)]

print leastSquaresFit(f, (1e13,4700), data_classical)

def f2(param, t):
    return 1e13*exp(-param[0]/t)

print leastSquaresFit(f2, (3000.,), data_quantum)
```

exception nictools.SP_LeastSquares.**IterationCountExceededError**

nictools.SP_LeastSquares.**leastSquaresFit**(*model*, *parameters*, *data*, *max_iterations=None*, *stopping_limit=0.005*)

General non-linear least-squares fit using the X{Levenberg-Marquardt} algorithm and X{automatic differentiation}.

model
[function] the function to be fitted. It will be called with two parameters: the first is a tuple containing all fit parameters, and the second is the first element of a data point (see below). The return value must be a number. Since automatic differentiation is used to obtain the derivatives with respect to the parameters, the function may only use the mathematical functions known to the module FirstDerivatives.

parameters
[tuple of numbers] a tuple of initial values for the fit parameters

data
[list] a list of data points to which the model is to be fitted. Each data point is a tuple of length two or three. Its first element specifies the independent variables of the model. It is passed to the model function as its first parameter, but not used in any other way. The second element of each data point tuple is the number that the return value of the model function is supposed to match as well as possible. The third element (which defaults to 1.) is the statistical variance of the data point, i.e. the inverse of its statistical weight in the fitting procedure.

Returns
fitlist : list

a list containing the optimal parameter values

chisq : float

chi-squared value describing the quality of the fit

nictools.SP_LeastSquares.**polynomialLeastSquaresFit**(*parameters*, *data*)
Least-squares fit to a polynomial whose order is defined by the number of parameter values.

Note: This could also be done with a linear least squares fit from L{LinearAlgebra}

Parameters
parameters : tuple

a tuple of initial values for the polynomial coefficients

data : list

the data points, as for L{leastSquaresFit}

9.6 SP_numpy

nictools.SP_numpy.**int_sum**(*a*, *axis=0*)

nictools.SP_numpy.**zeros_st**(*shape*, *other*)

9.7 tfbutil

nictools.tfbutil.**all_printMsg**(*message*, *level=1*)

nictools.tfbutil.**checkVerbosity**(*level*)
Return true if verbosity is at least as great as level.

nictools.tfbutil.**printMsg**(*message*, *level=0*)

nictools.tfbutil.**setDry_run**(*dry_run_value*)
Copy dry_run to a variable that is global for this file.

Parameters
dry_run : string

string that is either True or False

nictools.tfbutil.**setEdit_type_key**(*edit_type_value*)
Copy edit_type_key to a variable that is global for this file.

Parameters
edit_type_key : string

a string for the keyword name to write

nictools.tfbutil.**setErr_key**(*err_key_value*)
Copy err_key to a variable that is global for this file.

Parameters
err_key : string

a string for the keyword for the error estimate

nictools.tfbutil.**setForce**(*force_value*)
Copy force to a variable that is global for this file.

Parameters
force : string

string that is either None, Q, B, or A

nictools.tfbutil.**setHdr_key**(*hdr_key_value*)
Copy hdr_key to a variable that is global for this file.

Parameters
hdr_key : string

a string for the keyword name to write

nictools.tfbutil.**setNoclean**(*noclean_value*)
Copy no_clean to a variable that is global for this file.

Parameters
no_clean : string

string that is either True or False

nictools.tfbutil.**setNref**(*nref_value*)
Copy nref to a variable that is global for this file.

Parameters
nref : string

string for name of directory containing nonlinearity file

nictools.tfbutil.**setVerbosity**(*verbosity_level*)
Copy verbosity to a variable that is global for this file.

Parameters
verbosity_level L int :

an integer value indicating the level of verbosity

CHAPTER

TEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

n

n

I

L

M

N

O

P

Q

R

S

T

U

W

Z