SPIKE Documentation

Release 1.0

December 03, 2014

Contents:

CHAPTER

ONE

INTRODUCTION

This is the beta version of the SPIKE program.

1.1 What is SPIKE ?

SPIKE is a program coming from a first-development oriented one named NPK-V2 that allows the processing, the display and the analysis of data-sets obtained from various Fourier-Transform spectroscopies. It stands for Spectrometry Processing Innovative KErnel.

For the moment, il handles the following data-sets

- NMR - 1D and 2D are fully supported
- FT-ICR - 1D and 2D are fully supported
- Orbitrap - 1D only
- other spectroscopies are being considered

Files can be imported from

- NMR : Bruker topspin
- FT-ICR : Bruker Apex
- Orbitrap : Thermofisher raw data

It allows to process datasets interactively from an ipython prompt or interactively using the processing.py batch program (aimed towad FT-ICR for the moment) The batch mode supports multiprocessing, both with MPI and natively on multi-core machines (still in-progress).Data-sets are handled in the HDF5 standard file-format, which allows virtually unlimited file size.

Version : this is 0.5 beta version

1.2 How do I get set up?

The program is in python 2.7. Look at the examples files (**eg_**.py) and at configuration files (.mscf) they contain valuable examples and some documentation. SPIKE requires the following libraries :

- numpy
- scipy
- matplotlib
- Qt / PySide

- Pytables
- mpi4py * ...

It has been successfully tested in the Enthought and anaconda link distributions.

1.3 Organisation of the Code

The main program is NPKData.py, which defines NPKData object on which everything is built. Spectroscopies are defined in the FTICR.py and Orbitrap.py code, which sub class NPKData It is prototyped as an NMR data-set, but this will change. Many programs contain routines tests (in an object unittest) that also serve as an example of use.

1.4 Main programs :

A small description of the files: - NPKData.py the main library, allows all processing for NMR experiments (1D, 2D and 3D) to be used as a library, in a stand-alone program or in ipython interactive session - FTICR.py an extension of NPKData for processing FT-ICR datasets (1D and 2D) - Orbitrap.py an extension of NPKData for processing Orbitrap datasets (1D)

> processing.py a stand alone program, written on the top of FTICR.py, allowing the efficient processing of FT-ICR 2D datasets, with no limit on the size of the final file Produces multi-resolution files syntax : python processing.py param_file.mscf visu2D.py an interactive tool for visualizing 2D FT-ICR multi-resolution files python visu2D.py param_file.mscf

1.5 Directories

Three main directories

- Algo contains algorithms to process data-sets (MaxEnt, Laplace, etc...) not everything active !
- File Importers for various file format for spectrometry, as well as the HDF5 SPIKE native format.
- Visu utilities for the Visu2D program

Some usage examples

- SPIKE_usage_eg example python programs using the various library available

Various codes

- Miscellaneous “en vrac”
- Display a small utility to choose either for regular matplotlib display of fake no-effect display (for tests)
- util set of low-level tools used all over in the code
- v1 a library implementing a compatibility with the NPKV_V1 program

example of configuration files

- process_eg.mscf
- test.mscf

and various utilities

- NPKConfigParser.py reads .mscf files

- NPKError.py generates error msg
- QC.py Quality Check
- Tests.py runs all tests
- dev_setup.py rolls a new version
- version.py defines version number
- init.py defines library
- rcpylint
- To_Do_list.txt
- QC.txt
- Release.txt

1.6 Authors and Licence

Authors for this code are :

Marc-André Delsuc - CNRS

Lionel Chiron - CNRS then NMRTEC then Casc4de

Marie-Aude Coutouly - NMRTEC

Covered code is provided under this license on an "as is" basis, without warranty of any kind, either expressed or implied, including, without limitation, warranties that the covered code is free of defects. The entire risk as to the quality and performance of the covered code is with you. Should any covered code prove defective in any respect, you (not the initial developer or any other contributor) assume the cost of any necessary servicing, repair or correction.

Downloading code and datasets from this page signifies acceptance of the hereunder License Agreement. The code distributed here is covered under the CeCILL licence.

CHAPTER

TWO

TUTORIAL

Few examples of how to use SPIKE. We begin first with simple import for both FTICR datasets and Orbitrap datasets then we show how to make more elaborated commands involving data treatment algorithms such as RECITAL and urQRd.

First, open in SPIKE directory a terminal and launch a IPython Notebook document writing:

    ipython notebook

We assume that the data are in a directory next to SPIKE directory named DATA_test.

2.1 FTICR

- simple import of native dataset
- Show the FID
- Show the half truncated FID and full FID
- Doing FFT with zerofilling

2.1.1 simple import of native dataset

from File.Apex import Import_1D

import numpy as np

import matplotlib.pyplot as plt

from FTICR import FTICRData

Import from Apex

f = Import_1D("../DATA_test/angio_ms_000005.d")

2.1.2 Show the FID

f = Import_1D("../DATA_test/angio_ms_000005.d")

f.display(label = "FID")

2.1.3 Show the half truncated FID and full FID

f = Import_1D("../DATA_test/angio_ms_000005.d")

f.chsize(len(f.buffer)/2)

ff = f.copy()

ff.buffer = ff.buffer[:len(f.buffer)/2]/2

f.display(label = "FID")

f.display(label = "FID cut", new_fig = False)

2.1.4 Doing FFT with zerofilling

Classical FFT with apodisation and zerofilling.

FFT with zerofilling, processing cutting the pipes.

Here instead of writing a single long command with pipelines, the command is cut in many chunks. This can be used for performing intermediate operations not present in SPIKE.

2.2 Orbitrap

Some examples on how to use NPKv2.

- simple import of native dataset.
- simple FID handling, processing and display
- FFT with zerofilling

We begin first with simple import then we show how to make more elaborated commands involving data treatment algorithms such as RECITAL and urQRd.

2.2.1 simple import of native dataset

o = Import_1D("C:/Users/Egor/NPK_V2/DATA_test/ubiquitin_5_scan_res_30000_1.dat")

2.2.2 Show FID

o.display(label = "FID")

2.2.3 FFT with zerofilling

print o

o.apod_sin(maxi = 0.5).chsize(o.buffer.size*2).rfft().modulus()

o.units = 'm/z'

o.display(label = "zerofill x2")

2.2.4 FFT with zerofilling, processing cutting the pipes.

o = Import_1D(filename)

o.units = 'm/z'

o.apod_sin(maxi = 0.5)

o.chsize(o.buffer.size*4)

o.rfft()

o.modulus().display(label = "zerofill x4")

2.3 urQRd

is a preprocessing technique used for reducing the noise. The parameter k given to urQRd is related to the number of expected lines in the spectrum. It should be chosen 2 to 3 times larger than this expected number. Be carefull than the processing time **and** the memory footprint are both proportionnal to this value.

data.units = 'm/z'

data.urqrd(k = 300).rfft().modulus().display(label = "urQRd, rank = 300")

2.3.1 Additional tricks

IPython shortcuts.

there are *many* shortcuts and tricks in IPython, read the doc !

a couple of them are really helpfull for MS processing

- you can execute a cell by hitting *shift-return*
- you can get the documentation of any function by adding a ? at the end of its name, eg *o.rfft?*
- you can get all possible values by hitting the <*TAB*> key. Try for instance typing *o.* <*TAB*>

2.3.2 SPIKE arcanes

If needed, you can directly manipulate the numeric data held into the SPIKE dataset:

- the *.get_buffer()* method returns the underlying *numpy* array.
- The *.set_buffer()* method sets it, data can be real or complex.
- Do *.adapt_size()* afterwards if you changed the number of points.

It is also possible to use this sheet as a simple calculator, can be handy some time, for instance for checking charge state.

CHAPTER

THREE

CODE

3.1 NPKData

NPKData.py

Implement the basic mechanisms for NMR data-sets

Created by Marc-André and Marie-Aude on 2010-03-17. Copyright (c) 2010 IGBMC and NMRTEC. All rights reserved.

class NPKData.**Axis**(*size=64*, *itype=0*, *units='point'*)
hold information for one spectral axis used internally

check_zoom(*zoom*)
check whether a zoom window, given as (low,high) is valid - check low<high and within axis size - check that it starts on a real index in itype is complex return a boolean

get_sampling()
returns the sampling scheme contained in current axis

load_sampling(*filename*)
loads the sampling scheme contained in an external file file should contain index values, one per line, comment lines start with a # complex axes should be sampled by complex pairs, and indices go up to self.size1/2

sampling is loaded into self.sampling and self.sampling_info is a dictionnary with information

points_axis()
return axis in points units, actually 0..size-1

sampled
true is sampled axis

set_sampling(*sampling*)
sets the sampling scheme contained in current axis

typestr()
returns its type (real or complex) as a string

unit_axis()
returns an axis in the unit defined in self.units

class NPKData.**LaplaceAxis**(*size=64*, *dmin=1.0*, *dmax=10.0*, *dfactor=1.0*, *units='points'*)
hold information for one Laplace axis (DOSY) used internally

dtoi(*value*)
returns point value (i) from damping value (d)

itod(*value*)
returns damping value (d) from point value (i)

report()
hight level report

class NPKData.**NMRAxis**(*size=64*, *specwidth=6283.185307179586*, *offset=0.0*, *frequency=400.0*, *itype=0*, *units='points'*)
hold information for one NMR axis used internally

Hz_axis()
return axis containing Hz values, can be used for display

extract(*(start*, *end)*)
redefines the axis parameters so that the new axe is extracted for the points [start:end]

freq_axis()
return axis containing Hz values, can be used for display

htoi(*value*)
returns point value (i) from Hz value (h)

htop(*value*)
returns ppm value (p) from Hz value (h)

itoh(*value*)
returns Hz value (h) from point value (i)

itop(*value*)
returns ppm value (p) from point value (i)

ppm_axis()
return axis containing ppm values, can be used for display

ptoh(*value*)
returns Hz value (h) from ppm value (p)

ptoi(*value*)
returns point value (i) from ppm value (p)

report()
high level reporting

class NPKData.**NPKData**(*dim=1*, *shape=None*, *buffer=None*, *name=None*, *debug=0*)
a working data used by the NPK package

The data is a numpy array, found in self.buffer can also be accessed directly d[i], d[i,j], ...

1D 2D and 3D are handled, 3 axes are defined : axis1 axis2 axis3 axes are defined as in NMR in 1D, every is in axis1 in 2D, the fastest varying dimension is in axis2, the slowest in axis1 in 3D, the fastest varying dimension is in axis3, the slowest in axis1 see axis_index typical properties and methods are : utilities:

.display() .check()

properties .itype .dim .size1, .size2, .size3 ...

moving data : .row(i) .col(i) .set_row(i) .set_col(i) .copy() .load() .save()

processing : .fft() .rfft() .modulus() .apod_xxx() sg() transpose() ...

arithmetics : .fill() .mult .add() also direct arithmetics : f = 2*d+e

all methods return self, so computation can be piped etc...

abs()
This command takes the absolute value of the current the data set

adapt_size()
adapt the sizes held in the axis objects to the size of the buffer TO BE CALLED each time the buffer size is modified otherwise strange things will happen

add(*otherdata*)
add the provided data : otherdata to the current one eg : data.add(otherdata) add content of otherdata to data buffer

can add NPKData and numbers

addbase(*constant*)
add a constant to the data

addfreq(*freq*, *amp=1.0*)
add to the current data-set (1D, 2D, 3D) a single frequency sinusoid characterized by its frequency (from axis.specwidth) and amplitude

addnoise(*noise*, *seed=None*)
add to the current data-set (1D, 2D, 3D) a white-gaussian, characterized by its level noise, and the random generator seed.

apod_apply(*axis*, *apod_buf*)
apply an apodisation, held into the buffer apod_buf

apod_em(*axis=0*, *lb=1.0*)
apply an exponential apodisation, lb is in Hz WARNING : different from common definition of apodisation

apod_gm(*axis=0*, *gb=1.0*)
apply an gaussian apodisation, gb is in Hz WARNING : different from common definition of apodisation

apod_sin(*axis=0*, *maxi=0*)
apply a sinebell apodisation maxi ranges from 0 to 0.5

apod_sq_sin(*axis=0*, *maxi=0*)
apply a squared sinebell apodisation maxi ranges from 0 to 0.5

apod_tm(*axis=0*, *tm1=0*, *tm2=0*)
apply a trapezoide apodisation, lb is in Hz WARNING : different from common definition of apodisation This commands applies a trapezoid filter function to the data- set. The function raises from 0.0 to 1.0 from the first point to point n1. The function then stays to 1.0 until point n2, from which it goes down to 0.0 at the last point. If in 2D or 3D then Fx tells on which axis to apply the filter.

apply_process(*axis_it*, *process*, *axis=0*, *mp=True*, *N_proc=None*)
scans through given data, using axis_it which is an iterator, applying process method (by its name) store results into self, along to axis if axis_it iterates over self, then processing is in-place

however it can iterate over an other data-set, thus importing the data

if self.dim is equal to axis_it().dim, then data are

if mp, does it in a multiprocessing fashion using multiprocessing.Pool() if N_proc is None, finds the optimum number itself.

axes(*axis*)
returns the required axis : 1, 2 or 3

centroid1d(*npoints=3*)
from peak lists determined with peak() realize a centroid fit of the peak summit and width, computes Full width at half maximum creates lists self.centered_peaks and self.width_peaks

Temporary so far, only based on regular sampling, not unit axis. ah-hoc structure, waiting for a real PEAK object

check(*warn=False*)
check basic internal validity raises exceptions unless warn is set to True - in which case, only warnings are issued can be used in pipes as it returns self if everything is ok

check1D()
true for a 1D

check2D()
true for a 2D

check3D()
true for a 3D

check_zoom(*zoom*)
check whether a zoom window, given as (low,high) or ((low1,high1),(low2,high2)) is valid - check low<high and within axis size - check that it starts on a real index in itype is complex return a boolean

chsize(*sz1=-1*, *sz2=-1*, *sz3=-1*)
Change size of data, zero-fill or truncate. DO NOT change the value of OFFSET and SPECW, so EXTRACT should always be preferred on spectra (unless you know exactly what your are doing).

col(*i*)
returns a 1D extracted from the current 2D at position 0<=i<=size2-1

conjg(*axis=0*)
take the inverse conjugate of the buffer

conv_n_p()
realises the n+p to SH conversion

copy()
return a copy of itself

diag(*direc='F12'*)
In 2D, extracts the diagonal of the 2D and put into the 1D buffer.

In 3D, extracts one diagonal plane of the 3D cube, chosen with the direc parameter and put it into the 2D buffer direct values are :

"F12" is the F1=F2 diagonal "F23" is the F2=F3 diagonal "F13" is the F1=F3 diagonal

dim
returns the dimension of data : 1 2 or 3 (for 1D 2D or 3D)

display(*scale=1.0*, *absmax=0.0*, *show=False*, *label=None*, *new_fig=True*, *axis=None*, *mode3D=False*, *zoom=None*, *xlabel='_def_'*, *ylabel='_def_'*, *figure=None*)
not so quick and dirty display using matplotlib or mlab - still a first try

scale allows to increase the vertical scale of display absmax overwrite the value for the largest point, which will not be computed

display is scaled so that the largest point is first computed (and stored in absmax), and then the value at absmax/scale is set full screen

show will call plot.show() at the end, allowing every declared display to be shown on-screen useless in ipython

label add a label text to plot xlabel, ylabel : axes label (default is self.units - use None to remove) axis used as axis if present, axis length should match experiment length

in 2D, should be a pair (xaxis,yaxis)

new_fig will create a new window if set to True (default) (active only is figure==None) mode3D use malb 3D display instead of matplotlib contour for 2D display zoom is a tuple defining the zoom window (left,right) or ((F1_limits),(F2_limits)) figure if not None, will be used directly to display instead of using its own

can actually be called without harm, even if no graphic is available, it will just do nothing.

display_peaks (*axis=None*, *peak_label=False*, *zoom=None*, *show=False*)
displays peaks generated with peak()

extract (*[[x1, y1]]*)
extract([x1, y1], [x2, y2]) or extract([x1, y1, x2, y2]) etc...

Permits to extract a portion of the data. Data can then be processed as a regular data-set. EXTRACT changes the value of OFFSET and SPECW accordingly.

•extract(x1,y1) for 1D datasets.

•extract(x1, y1, x2, y2) for 2D datasets.

see also : chsize

f (*x*, *y*)
used by 3D display

fastclean (*nsigma=2.0*, *nbseg=20*, *axis=0*)
set to zeros all points below nsigma times the noise level This allows the corresponding data-set, once stored to file, to be considerably more compressive.

nsigma: float the ratio used, typically 1.0 to 3.0 (higher compression)

nbseg: int the number of segment used for noise evaluation, see util.signal_tools.findnoiselevel

axis: int the axis on which the noise is evaluated, default is fastest varying dimension

fft (*axis=0*)
computes the complex Fourier transform,

takes complex time domain data and returns complex frequency domain data

see test_axis for information on axis

fftr (*axis=0*)
computes the alternate Fourier transform,

takes complex time domain data and returns real frequency domain data

see test_axis for information on axis

flip ()
on a 2D with axis2.itype==1 and axis1.itype==0 copies the imaginary from on axis to the other after this, we have

axis2.itype==0 and axis1.itype==1 size1 is doubled size2 is halved

Useful for complex FT this is the opposite of flop()

>>>bb=NPKData(buffer=array([[0., 1., 2., 3.],[4., 5., 6., 7.],[8., 9., 10., 11.],[12., 13., 14., 15.]])) >>>print bb.buffer array([[0., 1., 2., 3.],

[4., 5., 6., 7.], [8., 9., 10., 11.], [12., 13., 14., 15.]])

>>>bb.axis2.itype=1 >>>bb.flip() >>>print bb.buffer array([[0., 2.],

[1., 3.], [4., 6.], [5., 7.], [8., 10.], [9., 11.], [12., 14.], [13., 15.]])

flipphase(*ph0*, *ph1*, *axis=1*)
equivalent to flip(); phase();flop() but much faster apply a phase correction along F1 axis of a 2D. on 2D where axis1.itype = 0 and axis2.itype = 1 using pairs of columns as real and imaginary pair phase corrections are in degree

flop()
on a 2D with axis2.itype==0 and axis1.itype==1 copies the imaginary from on axis to the other after this, we have

axis2.itype==1 and axis1.itype==0 size1 is halved size2 is doubled

Useful for complex FT this is the opposite of flip()

get_buffer(*copy=False*)
returns a view or a copy of the numpy buffer containing the NPKData values dtype is either real or complex if axis is complex. remarks :

•default is a view, if you want a copy, simply do d.get_buffer(copy=True)

•if you use a view, do not modify the size, nor the dtype

•see set_buffer()

WARNING - In nD with n>1 and if NPKData is hypercomplex, only the fastest (n) axis is considered, all other imaginary parts are left as real.

ifft(*axis=0*)
computes the inverse of fft(), takes complex frequency domain data and returns complex time domain data

see test_axis for information on axis

ifftr(*axis=0*)
computes the inverse of fftr, takes real frequency domain data and returns complex time domain data

see test_axis for information on axis

irfft(*axis=0*)
computes the inverse of rfft(), takes complex frequency domain data and returns real time domain data

see test_axis for information on axis

itype
returns complex type of each axes coded as single number, using NPKv1 code

linear_interpolate(*xpoints*, *axis='F2'*)
"compute and applies a linear function as a baseline correction

load(*name*)
load data from a file

load_txt(*name*)
load 1D data in texte, single column, no unit - with attributes as pseudo comments

mean(*zone*)
computes mean value in the designed spectral zone Consider array as real even if itype is 1

median()
Executes a median filter on the data-set (1D or 2D).a window of x points (or y by x in 2D) is moved along the data set, the point are ordered, and the indexth point is taken as the new point for the data set.

minus()
Sets to zero the positive part of the data set see also : minus, zeroing

modulus()
takes the modulus of the dataset depends on the value f axis(i).itype

mult(*multiplier*)
Multiply data-set by a scalar eg : d.mult(alpha) multiplies d buffer by alpha

mult_by_vector(*axis*, *vector*, *mode='real'*)
multiply the data-set by a vector, along a given axis if mode == "real", does it point by point regardles of itype if mode == "complex" uses axis.itype to determine how handle complex values

in all cases vector can be real or complex

peak(*pos_neg=1*, *threshold=0.1*, *offset=None*)
first trial for peakpicker 1D only pos_neg = 1 / -1 / 0 : type of peaks positive / negative / threshold = minimum level, as absolute value self.peaks : index of the peaks self.peaks_ordered : index of the ordered peaks from maximum to minimum.

peaks2d(*threshold=0.1*, *zoom=None*, *value=False*)
Extract peaks from 2d Array dataset if value is True, return the magnitude at position (x,y)

phase(*ph0*, *ph1*, *axis=0*)
apply a phase correction along given axis phase corrections are in degree

plane(*axis*, *i*)
returns a 2D extracted from the current 3D at position 0<=i<=size1-1

plus()
Sets to zero the negative part of the data set see also : minus, zeroing

proj(*axis=0*, *projtype='s'*)
returns a projection of the dataset on the given axis projtype determines the algorithm :

"s" is for skyline projection (the highest point is retained) "m" is for mean,

real(*axis=0*)
This command extract the real part of the current the data set considered as complex. axis is not needed in 1D, can be F1, F2 or F12 in 2D, and can be F1, F2, F3, F12, F13, F23, or F123 in 3D.

recital(*finalsize*, *iterations=2*, *miniteration=100*, *noise=None*, *scale_noise=1.0*, *mfista=False*, *axis=0*)
Apply the recital resolution enhancement algorithm to the data held in buffer final size of produced spectrum is finalsized noise iteration (outer loop) and iterations (inner loop) can be used to force default values

time is proportionnal to iterations x miniteration stands for Resolution EnhanCement by ITerative ALgorithm.

input: finalsize : targeted size. iterations : maximal number of lambda modifications. miniteration : maximal number of iterations for each lambda value. noise : noise level measured outside the algorithm for performing the Soft Thresholding scale_noise : : scaling of the noise level measured in Recital itself.

report()
reports itself

reverse(*axis=0*)
reverse the order of the current data-set (i.e. first points are last, last points are first). If dataset is complex, REVERSE will reverse the complex vector (2 by 2).

revf(*axis=0*)
Processes FID data-sets by multiplying by -1 2 points out of 4. Permits to preprocess Bruker FIDs in Dim 2 (Bruker trick) before RFT, or permits to bring back zero frequency in the center for some other data formats

rfft(*axis=0*)
computes the real Fourier transform, takes real time domain data and returns complex frequency domain data

see test_axis for information on axis

row(*i*)
returns a 1D extracted from the current 2D at position 0<=i<=size1-1

save(*name*)
save data to a file

save_csv(*name*)
save 1D data in csv, in 2 columns : x, y x values are conditions by the .units attribute data attributes are stored as pseudo comments

data can be read back with File.csv.Import_1D()

save_txt(*name*)
save 1D data in texte, single column, no unit - with attributes as pseudo comments

set_buffer(*buff*)
modify the internal buffer of the NPKData. allows real or complex arrays to be used remarks

- see get_buffer()

set_col(*i*, *d1D*)
set into the current 2D the given 1D, as the column at position 0<=i<=size2-1

set_row(*i*, *d1D*)
set into the current 2D the given 1D, as the row at position 0<=i<=size1-1

sg(*window_size*, *order*, *deriv=0*, *axis=0*)
applies saviski-golay of order filter to data window_size : int

the length of the window. Must be an odd integer number.

order [int] the order of the polynomial used in the filtering. Must be less than *window_size* - 1.

deriv: int the order of the derivative to compute (default = 0 means only smoothing)

axis: int the axis on which the filter is to be applied, default is fastest varying dimension

sg2D(*window_size*, *order*, *deriv=None*)
applies a 2D saviski-golay of order filter to data window_size : int

the length of the square window. Must be an odd integer number.

order [int] the order of the polynomial used in the filtering. Must be less than *window_size* - 1.

deriv: None, 'col', or 'row'. 'both' mode does not work. the direction of the derivative to compute (default = None means only smoothing)

can be applied to a 2D only.

size1
returns the size of the F1 spectral axis in 1D 2D and 3D i.e. the unique axis in 1D, the slowest axis in 2D and 3D warning, if data along axis is complex, the size is twice the number of complex pairs

size2
returns the size of the F2 spectral axis in 2D and 3D i.e. the slowest axis in 2D and the intermediate in 3D warning, if data along axis is complex, the size is twice the number of complex pairs

size3
returns the size of the F3 spectral axis in 3D i.e. the slowest axis in 3D warning, if data along axis is complex, the size is twice the number of complex pairs

spline_interpolate(*xpoints*, *axis='F2'*, *kind=3*)
compute and applies a spline function as a baseline correction

std(*zone*)
computes standard deviation in the designed spectral zone Computes value on the real part only ** CHANGED ON July 2012 **

swap(*axis=0*)
swap both parth to complex this is the opposite of swa() >>>aa=NPKData(buffer=arange(8.)) >>>aa.axis1.itype=1 >>>print aa.buffer array([0., 1., 2., 3., 4., 5., 6., 7.]) >>>print aa.swa().buffer array([0., 4., 1., 5., 2., 6., 3., 7.])

test_axis(*axis=0*)
tests on axis

in 1D, axis is not used axis has to be 1 or "F1"

in 2D, axis is either 2 for horizontal / faster incremented dimension == "F2" or 1 for the other dimension == "F1" defaut is 2

in 3D, axis is 3, 2 or 1 with 3 the faster incremented and 1 the slower == F3 F2 F1 defaut is 3

alternativaly, you may use the strings "F1", "F2" or "F3" BUT not F12 F23 as 0 is rest to default

transpose(*axis=0*)
Transposes the 2D matrix or planes of the 3D cube. The sizes of the matrix must be a power of two for this command to be used. After transposition, the two dimensions are completely permuted

axis is used in 3D to tell which submatrices should be transposed

see also : sym chsize modifysize

units
copy units to all the axes

unswap(*axis=0*)
this is the opposite of swap() >>>aa=NPKData(buffer=arange(8.)) >>>aa.axis1.itype=1 >>>print aa.buffer array([0., 1., 2., 3., 4., 5., 6., 7.]) >>>print aa.unswa().buffer array([0., 2., 4., 6., 1., 3., 5., 7.])

urqrd(*k*, *orda=None*, *iterations=1*, *axis=0*)
Apply urQRd denoising to data k is about 2 x number_of_expected_lines Manages real and complex cases. Handles the case of hypercomplex for denoising of 2D FTICR for example.

xcol(*start=0*, *stop=None*, *step=1*)
an iterator over columns of a 2D so for c in matrix.xcol():

do something with c...

is equivalent to for i in range(matrix.size2): # i.e. all cols

c = matrix.col(i) do something with c...

you can limit the range by giving start, stop and step arguments - using the same syntax as xrange()

on hypercomplex data matrix.xcol(step=matrix.axis2.itype+1) will step only on cols associated to the real point

xrow(*start=0*, *stop=None*, *step=1*)
an iterator over rows of a 2D so for r in matrix.xrow():

do something with r...

is equivalent to for i in range(matrix.size1): # i.e. all rows

r = matrix.row(i) do something with r...

you can limit the range by giving start, stop and step arguments - using the same syntax as xrange()

on hypercomplex data matrix.xrow(step=matrix.axis1.itype+1) will step only on rows associated to the real point

zeroing(*threshold*)
Sets to zero points below threshold (in absolute value) see also : plus, minus

zf(*zf1=None*, *zf2=None*, *zf3=None*)
Zerofill data by adding zeros. for a dataset of length size, will add zeros up to zf*size

do nothing by default unless axis is sampled, in which case, missing unsampled points are replaced by 0.0

zoom(*dim*, **args*)
The basic command for defining region of interest window

•if n=0, zoom mode is off.

•if n=1, zoom mode is on,

class NPKData.**NPKDataTests**(*methodName='runTest'*)

•Testing NPKData basic behaviour -

test_NUS_sampling()
NUS example removing the sampling noise

test_dampingunit()
test itod and dtoi

test_fft()

•Testing FFT methods -

test_flatten()
test the flatten utility

test_hypercomplex_modulus()
Test of hypercomplex modulus

test_load()

•Testing load methods

test_math()

•Testing math methods -

test_peaks1d()
test 1D peak picker

test_peaks2d()
test 2D peak picker

test_recital_synthetic()
Test recital on synthetic dataset.

test_superresolution()
NUS example removing the sampling noise white_noise.jpg

NPKData.**as_cpx**(*arr*)
interpret arr as a complex array useful to move between complex and real arrays (see as_float)

```
>>> print as_cpx(np.arange(4.0))
[ 0.+1.j  2.+3.j]
```

NPKData.**as_float**(*arr*)
interpret arr as a float array useful to move between complex and real arrays (see as_float)

```
>>> print as_float(np.arange(4)*(1+1j))
[ 0.  0.  1.  1.  2.  2.  3.  3.]
```

NPKData.**conj_ip**(*a*)
computes conjugate() in-place

```
>>> conj_ip(np.arange(4)*(1+1j))
[ 0.-0.j  1.-1.j  2.-2.j  3.-3.j]
```

NPKData.**copyaxes**(*inp*, *out*)
copy axes values from NPKDAta in to out.

internal use

NPKData.**flatten**(**arg*)
flatten recursively a list of lists

>>>print flatten(((1,2), 3, (4, (5,), (6,7)))) [1, 2, 3, 4, 5, 6, 7]

NPKData.**hypercomplex_modulus**(*arr*, *size1*, *size2*)
Calculates the modulus of an array of hypercomplex numbers. input:

arr : hypercomplex array size1 : size counting horizontally each half quadrant. size2 : siez counting vertically each half quadrant.

eg: arr = np.array([[1, 4],[3, 7],[1, 9],[5, 7]]) is an hypercomplex with size1 = 2 and size2 = 2

NPKData.**warning**(*msg*)
issue a warning message to the user

3.2 File formats

3.2.1 Apex

Utility to Handle Apex files

File.Apex.**Import_1D**(*folder*, *outfile=''*)
Entry point to import 1D spectra It returns a FTICRData It writes a HDF5 file if an outfile is mentionned

File.Apex.**Import_2D**(*folder*, *outfile=''*, *F1specwidth=None*)
Entry point to import 2D spectra It returns a FTICRData It writes a HDF5 file if an outfile is mentionned

File.Apex.**Ser2D_to_H5f**(*sizeF1*, *sizeF2*, *filename='ser'*, *outfile='H5f.h5'*, *chunks=None*)
Charge any ser file directly in H5f file

File.Apex.**get_param**(*param*, *names*, *values*)
From params, this function returns the value of the given param

File.Apex.**locate_acquisition**(*folder*)
From the given folder this function return the absolute path to the apexAcquisition.method file It should always be in a subfolder

File.Apex.**read_2D**(*sizeF1*, *sizeF2*, *filename='ser'*)
Reads in a Apex 2D fid

sizeF1 is the number of fid sizeF2 is the number of data-points in the fid uses array

File.Apex.**read_3D**(*sizeF1*, *sizeF2*, *sizeF3*, *filename='ser'*)
Ebauche de fonction

Reads in a Apex 3D fid

uses array

File.Apex.**read_param**(*filename*)
Open the given file and retrieve all parameters written initially for apexAcquisition.method NC is written when no value for value is found

structure : <param><name>C_MsmsE</name><value>0.0</value></param>

read_param returns values in a dictionnary

File.Apex.**read_scan**(*filename*)
Function that returns the number of scan that have been recorded It is used to see wether the number of recorded points correspond to the L_20 parameter

File.Apex.**write_ser**(*bufferdata*, *filename='ser'*)
Write a ser file from FTICRData

Utility to import and export data in text and csv files

all functions compress transparently if the filenales end with .gz Marc-André adapted from some Lionel code

File.csv.**Import_1D**(*filename*, *column=0*, *delimiter=', '*)
import a 1D file stored as csv header as comments (#) parameters in pseudocomments :

#$key value

then one value per line column and delimiter as in load()

class File.csv.**NPKDataTests**(*methodName='runTest'*)

•Testing NPKData basic behaviour -

File.csv.**load**(*filename*, *column=0*, *delimiter=', '*)
load 1D data from txt or csv file, attribute are in pseuo-coments startin with #$ value are in columns, separated by delimiter - only the columun given in arg will be loaded column = 0 is fine for text files column = 1 is fine for csv files with units returns a numpy buffer and an attribute dictionary

File.csv.**save**(*data*, *filename*, *delimiter=', '*)
save 1D data in txt, single column, no unit - with attributes as pseudo comments

File.csv.**save_unit**(*data*, *filename*, *delimiter=', '*)
save 1D data in csv, in 2 columns, with attributes as pseudo comments

3.2.2 GifaFile

GifaFile.py

Created by Marc-André on 2010-03-17. Copyright (c) 2010 IGBMC. All rights reserved.

This module provides a simple access to NMR files in the Gifa format.

class File.GifaFile.**GifaFile**(*fname*, *access='r'*, *debug=0*)

defines the interface to simply (read/write) acces Gifa v4 files standard methods are load() and save()

standard sequence to read is F = GifaFile(filename,"r") B = F.get_data() # B is a NPKdata F.close()

or F = GifaFile(filename,"r") F.load() B = F.data # B is a NPKdata F.close()

and to write F = GifaFile(filename,"w") F.set_data(B) # where B is a NPKdata; do not use F.data = B F.save() F.close()

The file consists of a header (of size headersize) and data The header is handled as a dictionnary self.header data is handled as a NPKdata self.data

so numpy ndarray are in self.data.buffer

byte_order
pour intel

close()
closes the associated file

copyaxesfromheader(*n_axis*)
get values from axis "n_axis" from header, and creates and returns a new (NMRAxis) axis with this values itype is not handled (not coded per axis in header) used internally

copydiffaxesfromheader()
get values from axis "n" from header, and creates and returns a new (LaplaceAxis) axis with this values used internally

dim
dimensionality of the dataset 1 2 or 3

get_data()
returns the NPKdata attached to the (read) file

itype
Real/complex type of the dataset in 1D : 0 : real 1: complex in 2D : 0 : real on both;

1 : complex on F2 2 : complex on F1 3 : complex on both

in 3D [0][real on all;] 1 : complex on F3 2 : complex on F2 3 : complex on F3-F2 4 : complex on F1 5 : complex on F1-F3 6 : complex on F1-F2 7 : complex on all

load()
creates a NPKdata loaded with the file content

load_header()
load the header from file and set-up every thing

nblock1
number of data block on disk along F1 axis

nblock2
number of data block on disk along F2 axis

nblock3
number of data block on disk along F3 axis

read_header()
return a dictionnary of the file header internal use

readc()
read a file in Gifa format, and returns the binary buffer as a numpy array internal use - use load()

report()
prints a little debugging report

save()
save the NPKdata to the file

set_data(*buff*)
sets the NPKdata attached to the (to be written) file

setup_header()
setup file header, from self.data

size1
size along the F1 axis (either 1D, or slowest varyong axis in nD)

size2
size along the F2 axis (fastest varying in 2D)

size3
size along the F3 axis (fastest varying in 3D)

szblock1
size of data block on disk along F1 axis

szblock2
size of data block on disk along F2 axis

szblock3
size of data block on disk along F3 axis

write_header()
write file header setup_header() should have been called first

write_header_line(*key*)
write into the header the entry key returns the number of byte written internal use

writec()
write a file in Gifa format internal use - use save()

class File.GifaFile.**GifaFileTests**(*methodName='runTest'*)

•Testing GifaFile on various 1D and 2D files -

tempfile = <module 'tempfile' from '/Users/chiron/anaconda/lib/python2.7/tempfile.pyc'>

3.2.3 HDF5File

HDF5File.py

Created by Marc-André Delsuc, Marie-Aude Coutouly on 2011-07-13. Copyright (c) 2011 __NMRTEC__. All rights reserved.

API dealing with HDF5File. For now it is non surclassing tables, you have to use *.hf. to access all tables functionalities

class File.HDF5File.**HDF5File**(*fname*, *access='r'*, *info=None*, *nparray=None*, *fticrd=None*, *debug=0*)
defines the interface to simply (read/write) access HDF5 files standard methods are load() and save()

standard sequence to read is H = HDF5File(filename,"r") B = H.get_data() # B is a FTICRdata H.close()

or H = HDF5File(filename,"r") H.load() B = H.data # B is a FTICRdata H.close()

and to write H = HDF5File(filename,"w") H.set_data(B) # where B is a FTICRdata; do not use H.data = B H.save() H.close()

axes_update(*group='resol1'*, *axis=2*, *infos=None*)
routine called when you want to modify the information on a given axis group is the group name, default is resol1 axis is the dimension we want to adjust infos is a dictionnary with al fields we want to adjust

checkversion()
check file version and exit if incompatible

close()
Closes HDF5File

createCArray(*where*, *name*, *data_type*, *shape*, *chunk=None*)
Create a CArray in the given hf_file

createGroup(*where*, *name*)
Create a group in the given hf_file

createTable(*where*, *name*, *description*)
Create a Table in the given hf_file at the given position with the right description

create_HDF5_info()
Creates a HDF5 file, takes info as parameter

create_HDF5_nparray()
Creates a HDF5 file, takes nparray as parameters

create_from_template(*data*, *group='resol1'*)
Take params from the empty FTICR data and put all the informations in the HDF5File creates an empty data, and attach it to data.buffer data is created in group, with default value 'resol1'

create_generic(*owner='NMRTEC'*)
A table is created with all generic informations about the file : owner, method, HDF5 Release,CreationDate, Last modification

create_tables()
Creates the different tables needed in a HDF5File according to the info parameters given If you don't pass any info dictionnary, it will take parameters from the self.header

determine_chunkshape(*sizeF1=None*, *sizeF2=None*)
Determine a good chunkshape according to the size of each axis

fill_table(*table*, *infos*)
Fill in the given table. Axis is the dimension we are processing

get_data(*group='resol1'*, *mode='onfile'*)
loads and returns the FTICRdata attached to the self file same parameters as load()

get_file_infos()
Read the generic_table and return the informations

get_info()
Retrieve info from self.nparray

load(*group='resol1'*, *mode='onfile'*)
loads the data into memory, set self.data as a FTICRData

group defines which group is loaded (default is resol1) mode defines how it is loaded in memory,

"onfile" (default) the data is kept on file and loaded only on demand. the capability of modifying the data is determined by the way the file was opened the data cannot be modified unless the file was opened with access='w' or 'rw'

"memory" the data is copied to a memroy buffer and can be freely modified warning - may saturate the computer memory, there is no control

if you want to load data into memory after having opened in 'onfile" mode, then do the following : h.load(mode="onfile") b = d.data.buffer[...] # data are now copied into a new memory buffer b using ellipsis syntax d.data.buffer = b # and b is used as the data buffer.

position_array(*group='resol1'*)
Fill in the HDF5 file with the given buffer, HDF5 file is created with the given numpy array and the corresponding tables

save(*ser_file*, *group='resol1'*)
save the ser_file to the HDF5 file

save_fticrd()
save the FTICRData to the H5F file

set_compression(*On=False*)
sets HDF5 file compression to zlib if On is True; to none otherwise

set_data(*data*, *group='resol1'*)
Take the ser_file and the params and put all the informations in the HDF5File

set_data_from_fticrd(*buff*, *group='resol1'*)
sets the FTICRdata attached to the (to be written) file

table_update(*group='resol1'*, *axis=2*, *key='highmass'*, *value=4000.0*)
Microchangement in the wanted table

File.HDF5File.**determine_chunkshape**(*size1*, *size2*)
returns optimum size for chuncks for a dataset of file size1, size2 and update cachesize for accomodating dataset

File.HDF5File.**nparray_to_fticrd**(*name*, *nparray*)

File.HDF5File.**up0p6_to_0p7**(*fname*, *debug=1*)
docstring for up0p6_to_0p7 Function that deals with changing HDF5 files created with file_version 0.6 to be read with 0.7 It modifies

File.HDF5File.**up0p7_to_0p8**(*fname*, *debug=1*)
docstring for up0p7_to_0p8 Function that deals with changing HDF5 files created with file_version 0.7 to be read with 0.8

File.HDF5File.**update**(*fname*, *debug=1*)
update so that the file is up to date

3.2.4 Solarix

Solarix.py

Utility to Handle Solarix files

Created by mac on 2013-05-24. Copyright (c) 2013 __NMRTEC__. All rights reserved.

File.Solarix.**Import_1D**(*folder*, *outfile=''*)
Entry point to import 1D spectra It returns a FTICRData It writes a HDF5 file if an outfile is mentionned

File.Solarix.**Import_2D**(*folder*, *outfile=''*, *F1specwidth=None*)
Entry point to import 2D spectra It returns a FTICRData It writes a HDF5 file if an outfile is mentionned

File.Solarix.**Ser2D_to_FTICRFile**(*sizeF1*, *sizeF2*, *filename='ser'*, *outfile='H5f.h5'*, *chunks=None*)
Charge any ser file directly in H5f file

File.Solarix.**get_param**(*param*, *names*, *values*)
From params, this function returns the value of the given param

File.Solarix.**locate_acquisition**(*folder*)
From the given folder this function return the absolute path to the apexAcquisition.method file It should always be in a subfolder

File.Solarix.**read_2D**(*sizeF1*, *sizeF2*, *filename='ser'*)
Reads in a Apex 2D fid

sizeF1 is the number of fid sizeF2 is the number of data-points in the fid uses array

File.Solarix.**read_3D**(*sizeF1*, *sizeF2*, *sizeF3*, *filename='ser'*)
Ebauche de fonction

Reads in a Apex 3D fid

uses array

File.Solarix.**read_param**(*filename*)
Open the given file and retrieve all parameters from apexAcquisition.method NC is written when no value for value is found

structure : <param name = "AMS_ActiveExclusion"><value>0</value></param>

read_param returns values in a dictionnary

File.Solarix.**read_scan**(*filename*)
Function that returns the number of scan that have been recorded It is used to see wether the number of recorded points correspond to the L_20 parameter

File.Solarix.**write_ser**(*bufferdata*, *filename='ser'*)
Write a ser file from FTICRData

3.2.5 Thermo

Utility to Handle Thermofisher files

Marc-André from first draft by Lionel

File.Thermo.**Import_1D**(*filename*)
Entry point to import 1D spectra It returns an Orbitrap data

class File.Thermo.**Thermo_Tests**(*methodName='runTest'*)
A FAIRE

File.Thermo.**read_data**(*F*, *typ='float'*)
given F, an opened file, reads the values and read_param returns values in a dictionnary

File.Thermo.**read_param**(*F*)
given F, an opend file , retrieve all parameters found in file header

read_param returns values in a plain dictionnary

File.Thermo.**read_thermo**(*filename*)
reads a thermofisher orbitrap file

3.3 FTICR

FTMS.py

class FTICR.**FTICRAxis**(*size=1024*, *specwidth=6283.185307179586*, *itype=0*, *units='point'*, *ref_mass=344.0974*, *ref_freq=419620.0*, *highmass=10000.0*, *left_point=0.0*)

hold information for one FT-ICR axis used internally

Hz_axis()

return axis containing Hz values, can be used for display

deltamz(*mz_value*)

computes the theorical resolution in m/z at m/z location

extract(*(start*, *end)*)

redefines the axis parameters so that the new axe is extracted for the points [start:end]

freq_axis()

return axis containing Hz values, can be used for display

htoi(*value*)

returns point value (i) from Hz value (h)

itoh(*value*)

returns Hz value (h) from point value (i)

itomz(*value*)

return m/z (mz) from point value (i)

lowmass

highest mass of interest - defined by the Nyquist frequency limit

mass_axis()

return axis containing m/z values, can be used for display

mz_axis()

return axis containing m/z values, can be used for display

mztoi(*value*)

return point value (i) from m/z (mz)

report()

high level reporting

class FTICR.**FTICRData**(*dim=1*, *shape=None*, *mode='memory'*, *buffer=None*, *name=None*, *debug=0*)

subclass of NPKData, meant for handling FT-ICR data allows 1D and 2D data-sets

display(*scale=1.0*, *absmax=0.0*, *show=False*, *label=None*, *new_fig=True*, *axis=None*, *mode3D=False*, *zoom=None*, *xlabel='_def_'*, *ylabel='_def_'*, *figure=None*)

display the FTICR data using NPKDATA display method check parameters in NPKDATA

- copied here - (might be out of date)

scale allows to increase the vertical scale of display absmax overwrite the value for the largest point, which will not be computed

display is scaled so that the largest point is first computed (and stored in absmax), and then the value at absmax/scale is set full screen

show will call plot.show() at the end, allowing every declared display to be shown on-screen useless in ipython

label add a label text to plot xlabel, ylabel : axes label (default is self.units - use None to remove) axis used as axis if present, axis length should match experiment length

in 2D, should be a pair (xaxis,yaxis)

new_fig will create a new window if set to True (default) (active only is figure==None) mode3D use malb 3D display instead of matplotlib contour for 2D display zoom is a tuple defining the zomm window (left,right) or ((F1_limits),(F2_limits)) figure if not None, will be used directly to display instead of using its own

can actually be called without harm, even if no graphic is available, it will just do nothing.

highmass
copy highmass to all the axes

ref_freq
copy ref_freq to all the axes

ref_mass
copy ref_mass to all the axes

save_msh5 (*name*, *set_compression=False*)
save data to a HDF5 file

experimental !

specwidth
copy specwidth to all the axes

trimz (*axis=0*)
extract the data so as to keep only lowmass-highmass range axis determines which axis to trim, axis=0 (default) indicates all axes

units
copy units to all the axes

FTICR.**fticr_mass_axis** (*length*, *spectral_width*, *ref_mass*, *ref_freq*)
returns an array which will calibrate a FT-ICR experiment length : number of points in the axis spectral_width : of the ICR measure ref_mass : value of the m/z reference ref_freq =: frequence at which is is observed.

3.4 Orbitrap

Orbitrap.py

Created by Marc-André and Lionel on 10 april 2014 Copyright (c) 2014 IGBMC. All rights reserved.

class Orbitrap.**OrbiAxis** (*size=1024*, *specwidth=10000000.0*, *itype=0*, *units='point'*, *ref_mass=715.3122*, *ref_freq=1887533.975611561*, *highmass=10000.0*, *left_point=0.0*)
hold information for one Orbitrap axis used internally

Hz_axis ()
return axis containing Hz values, can be used for display

deltamz (*mz_value*)
computes the theorical resolution in m/z at m/z location

extract (*(start*, *end)*)
redefines the axis parameters so that the new axe is extracted for the points [start:end]

freq_axis ()
return axis containing Hz values, can be used for display

htoi (*value*)
returns point value (i) from Hz value (h)

itoh(*value*)
returns Hz value (h) from point value (i)

itomz(*value*)
return m/z (mz) from point value (i)

lowmass
highest mass of interest - defined by the Nyquist frequency limit

mztoi(*value*)
return point value (i) from m/z (mz)

report()
high level reporting

class Orbitrap.**OrbiData**(*dim=1*, *shape=None*, *mode='memory'*, *buffer=None*, *name=None*, *debug=0*)
subclass of NPKData, meant for handling Orbitrap data doc to be written ...

display(*scale=1.0*, *absmax=0.0*, *show=False*, *label=None*, *new_fig=True*, *axis=None*, *mode3D=False*, *zoom=None*, *xlabel='_def_'*, *ylabel='_def_'*, *figure=None*)
display the Orbitrap data using NPKDATA display method check parameters in NPKDATA

•copied here - (might be out of date)

scale allows to increase the vertical scale of display absmax overwrite the value for the largest point, which will not be computed

display is scaled so that the largest point is first computed (and stored in absmax), and then the value at absmax/scale is set full screen

show will call plot.show() at the end, allowing every declared display to be shown on-screen useless in ipython

label add a label text to plot xlabel, ylabel : axes label (default is self.units - use None to remove) axis used as axis if present, axis length should match experiment length

in 2D, should be a pair (xaxis,yaxis)

new_fig will create a new window if set to True (default) (active only is figure==None) mode3D use malb 3D display instead of matplotlib contour for 2D display zoom is a tuple defining the zomm window (left,right) or ((F1_limits),(F2_limits)) figure if not None, will be used directly to display instead of using its own

can actually be called without harm, even if no graphic is available, it will just do nothing.

highmass
copy highmass to all the axes

ref_freq
copy ref_freq to all the axes

ref_mass
copy ref_mass to all the axes

save_msh5(*name*)
save data to a HDF5 file

experimental !

specwidth
copy specwidth to all the axes

trimz(*axis=0*)
extract the data so as to keep only lowmass-highmass range axis determines which axis to trim, axis=0 (default) indicates all axes

units
copy units to all the axes

3.5 visu2D

Created by Marc Andre Delsuc & Lionel Chiron on 2011-05-19. Copyright (c) 2011 IGBMC. All rights reserved. ### Program for visualizing FTICR2D data. Launch the PyQt4/PySide visualizer for FTICR2D.

visu2D.**debugs_activate**(**args*)
Debugging class methods. Classes debugged are -interface -display -convert -gtools -zooming -move window -interact

visu2D.**main**(*argv=None*)
creates and runs

3.5.1 inside Visu2D

Initialize and handle

interface

Created by Lionel Chiron 02/10/2013 Copyright (c) 2013 __NMRTEC__. All rights reserved.

class Visu.interface.**INTERFACE**
Creation of the graphic window Methods :

- init_interf
- makelayout
- makecanvas
- clearlayout
- run

clearlayout(*layout*)
Clear the layout of the centralwidget

init_interf()
Initialization of the interface

makelayout()
make the layout in the centralwidget

pr(*var*, *message=''*)
print

interface_actions

class Visu.interface_actions.**INTERACT**(*zoom*, *mwind*, *data*, *display*, *interf*, *paramz*, *gtools*, *zoom3d*, *stools*, *convert*, *save*)
Handle all the interface interactions.

afffile()
Show dataset in C window and addresses of used files vis.resmin : resolution for window D

backhome()
going to the original view

backzoo()
going back in the zooms

button(*nb*, *action*, *name_icon=None*, *icon_size=None*)
General definition for the buttons nb : number for the button action : method associated to the action. name_icon : name of the icon in the directory Visu/iconsUi/, must be in png format. icon_size : size of the icon for fitting to the button size.

coord_profile_x(*xval*)
Makes coordinates profile for x profile. Called by coord_profile. Returns the extreme coordinates in m/z or point format.

coord_profile_y(*yval*)
Makes coordinates profile for y profile Called by coord_profile Returns the extreme coordinates in m/z or point format

drag_connect()
Drag the main image.

forwzoo()
going forward in the zooms

interfGraph()
Defines the buttons, lineEdits and actions associated.

lineEdit(*nb*, *action*)
General lineEdit nb : number of the lineEdit action : associated action.

lineEdit_and_button(*name_lineEdit*, *name_button*, *action*)
Defines a lineEdit and an associated button. Uses self.lineEdit() and self.button()

list_res()
resolutions list made from self.data

manual_profile()
Manual profile

manual_scale()
Manual scaling.

manual_zoom()
Manual zoom Coordinates are entered manually with format llx, lly, urx, ury.

permute_states(*test*, *etat0*, *etat1*)
function for swapping between two states.. have to define first value of state before.

pr(*var*, *message=''*)
print

prepare_coord_profile(*values*, *ct=None*)
Makes coordinates profile Called by take_lineEdit_xy_format

savefigure(*kind*)
Function to save figure from window C.

savefigurepdf()
Save the main view in pdf

savefigurepng()
Save the main view in png

scale_control()
Show scale in the interface.

select_curs()
Select the arrow cursor

select_drag()
Select the drag function (hand)

select_manual_profile()
Select the function manual_profile

swap_from_mz()
Passes from m/z to point

swap_from_proint()
Passes from point to m/z

swap_pt_mz()
Function for passing from "point mode" to "m/z mode" and inversely. Change only in C window

take_lineEdit(*ledit*)
Takes values from lineEdit for zoom and profile. Passes the coordinates in "point mode". Called by manual_profile.

take_lineEdit_xy_format(*lineEdit_value*)
Takes the profile with format "y200" for horizontal line y=200 ct is the coordinates type

zoom3D()
From zoom coordinates, calculates the zoom area. If the area is small enough, makes the 3D reprensentation in m/z coordinates.

Pyside_PyQt4

Load

class Visu.Load.**LOAD**(*configfile=None*, *msh5file=None*)
Class to load the resolutions from the msh5file directly or addressed in the visu2D.mscf.

loadres()
Loads the different resolutions from Hdf5 files and put them in a list (self.d) of FTICRdata objects. self.d[0] is the highest resolution. Loadres() counts also the number of resolutions in the msh5 file.

pr(*var*, *message=''*)
print

class Visu.Load.**Visu_Parameters**(*configfile=None*)
this class is a container for visualization parameters

load(*cp*)
Loads in self the information from the configfile.

pr(*var*, *message=''*)
print

report()
Show all the parameters in self for the class Visu_parameters.

Visualization

paramzoom

class Visu.paramzoom.**PARAM_ZOOM**(*data*)
Central object for zoom management.

report()
Report values for zoom, greyzoom, zoomready, movezoo etc..

zoom_diag_vector()
Vector from diagonal for drag etc..

display

class Visu.display.**DISPLAY**(*QtMplCv*, *data*, *interface*, *paramz*)
Fill Canvas with Matplotlib figures. Handle connect disconnect.

aff_resolution()
Shows the resolution in the interface.

affd(*d1*, *d2*, *layout1*, *layout2=None*, *zoom=True*, *message=None*)
Routine to show the Mpl in qt zoom in main window and global window d1 and d2 are the data to be plotted.

affi(*canvas*, *d*)
Routine to print the 2d datas in Qt embedded environment It uses the coordinates of the zoom for a given resolution.

affichd(*canvas*, *d*, *zoom=True*)
Makes the display with NPKv2.

afflistco()
Prints the element of listco containing (zoom, resolution, scale) at position self.paramz.listview_index.

change_resolution(*layout1=None*, *layout2=None*)
Changes the resolution. self.currentd is selected according to vis.resolu.

connect(*event*, *action*, *window='C'*)
Connect

disconnect(*object_action*, *window='C'*)
Disconnect

distrib(*f*, *arg*)
Applying f to pairs of arg.

list_res()
Make list with the resolutions

local_abs_max()
maximum from local view

message(*message*, *posx=None*, *posy=None*, *colorlab=(1.0, 0.7, 0.7)*)
function to label the peaks

multzoom_coord(*alpha*, *beta*)
change the coordinates of the window with different factors (alpha, beta) according to the direction.

pr(*var*, *message=''*)
print

register_coordinates()
Keeps in a list "self.paramz.listview" the zooms coordinates and the associated resolutions.

res2dd()
Function to load the resolution from the current vis.resolu name

right_order_coord(*llx*, *lly*, *urx*, *ury*)
Restablishes right order for coordinates.

select_best_resolution()
In function of the size of the zoom chose the best resolution.

set_canvasC()
Makes the canvas C

set_canvasD()
Makes the canvas D

setcursor(*name*, *window='C'*)
Set the type of cursor used.

zoom_area()
Calculates area from zoom coordinates.

canvas_event

class Visu.canvas_event.**CANVAS_EVENT**(*display*, *interf*, *data*, *paramz*, *gtools*, *convert*, *stools*, *mwind*, *zoom*)
Handle Events in the Canvas.

aff_param()
Show resolution and print coordinates in lineEdit.

detect_corner(*event*)
detect position for stretching window and put the good mouse's shape

interact_with_canvasC()
Connects event to canvas C. Press, release, corner detection.

make_coord_manual()
Mouse coordinates are automatically written in the interface for manual interaction.

on_motion(*event*)
Activate on mouse motion.

on_press(*event*)
When pressed, triggers the rectangle drawing Waits for the mouse button's release to make the zoom. Calls self.on_press_D_event and self.on_press_C_event

on_press_C_event(*event*, *xpress*, *ypress*)
Makes the drawing of the zoom window in the layout C. If a click is produced again in the window, it makes the zoom.

on_press_D_event(*event*, *xpress*, *ypress*)
Permits to interact with the zoom in window D.

on_release(*event*)
On release keep the rectangle and makes the zoom on_released is used to make lines . self.paramz.listview, list of all the zooms.

on_release_C_event(*dx*, *dy*)
Release C event

on_release_D_event()
Release D event

pr(*var*, *message=''*)
print

recupxy(*event*)
Take the event coordinates and transform from "m/z" to "point" if necessary.

release_refrechC(*name_profile=None*)
When mouse is released, refreshes layout C.

canvas

class Visu.canvas.**Qt4MplCanvas**(*parent*, *paramz*)
Class for integrating Matplotlib in Qt

contextMenuEvent(*event*)
Context menu

pr(*var*, *message=''*)
print

convert

class Visu.convert.**CONVERT**(*display*, *data*, *paramz*)
Conversion between m/z and points.

itomz_all(*d*, *llx*, *lly*, *urx*, *ury*)
transforming from "point" coordinates to "mz" coordinates.

maxres(*llx*, *lly*, *urx*, *ury*)
If mode point, converts to coordinates with maximal resolution

mztoi(*d*, *coorr*, *ax*)
mz to point for each coordinate

mztoi_all(*d*, *llx*, *lly*, *urx*, *ury*)
transforming from "m/z" coordinate to "point" coordinates of coorr (zoom window)

pass_to_curr_mode(*llx*, *lly*, *urx*, *ury*)
If in mz/mode pass coordinates in "m/z mode", if in point mode pass the coordinates in "point mode".

pass_to_pt(*llx*, *lly*, *urx*, *ury*)
transforming from point coordinate to m/z coordinates with mode point or m/z conditon

pr(*var*, *message=''*)
print

set(*xa*, *ya*, *xb*, *yb*)
Setter to put the coordinate in the right order and avoid having values outside.

to_npk()
return npk formated zoom

Matplotlib_generictools

profile_popup

class Visu.profile_popup.**Dialog**(*data*, *save*, *toolbar*)
Dialog box for saving CSV and PDF files.

open_file_dialog(*kind_saved*)
Opens a file dialog Permits to save CSV and PDF.

pr(*var*, *message=''*)
print

class Visu.profile_popup.**NavigToolbar**(*canvas*, *parent*, *data*, *save*, *fig*, *axes*, *name_profile*, *namefile*)
Customized navigation toolbar with CSV and PDF added functions.

Button_template(*add_icon*, *action*)
General Button template. Used in custom_button_csv and custom_button_pdf

custom_button_csv()
Button for saving as CSV. Makes double columns CSV files using NPKv2 csv code.

custom_button_fullscale()
Button for rescalling at fullscale.

custom_button_pdf()
Button for saving PDFs.

pr(*var*, *message=''*)
print

class Visu.profile_popup.**PROFILE**(*data_profile*, *save*, *name_profile*, *namefile*, *ptlx=None*)
Popup window for the profiles. Creates a QMainWindow in which is made a customized toolbar. -Possible to save in PDF, CSV -Function for having full scale.

axes_mz(*data*)
Calculates axes in the case of x or y profile.

make_window()
Makes the profile window

plot_profile()
Plots the profile in the window

pr(*var*, *message=''*)
print

prepare_window()
Prepares the window with name, size, canvas

Visu.profile_popup.**profile_popup**(*data*)

zooming

class Visu.zooming.**ZOOM3D**
Zoom 3D

drawrect(*llx*, *lly*, *urx*, *ury*)
Draws rectangle for showing the area where the 3D is performed.

make_mz_xyz(*d*, *pt1min*, *pt1max*, *pt2min*, *pt2max*)
From the resolution d and frequency coordinates, returns the m/z coordinates x, y, z for making the irregular meshgrid.

makemesh(*d*, *pt1min*, *pt1max*, *pt2min*, *pt2max*)
from frequencies limits pt1min, pt1max, pt2min, pt2max, makes the 3D mesh X, Y, Z. Calls make_mz_xyz() then makemeshfromirreg().

makemeshfromirreg(*x*, *y*, *z*, *sizef1*, *sizef2*)
Makes the meshgrid from irregular mesh (x, y) in m/z coordinates. Called after make_mz_xyz()

plotregion3d(*d*, *pt1min*, *pt1max*, *pt2min*, *pt2max*, *visible=False*)
Makes the 3D plot from the meshgrid. Makes the json that is read by FTICR2D_3d.html

pr(*var*, *message=''*)
print

class Visu.zooming.**ZOOMING**(*display*, *interf*, *data*, *paramz*, *gtools*, *convert*, *stools*, *mwind*)
Zoom and draw a rectangle around the zooming area The coordinates are in "point" format at the root of the treatment so as to simplify all the procedure.

change_view(*change_layoutD=False*)
Changes views.

change_view_from_list()
Refreshes views from history.

change_zoom()
Makes zoom and resolution if necessary, condition = self.areazoom() >= AREAMIN Keeps both zoom and resolution in a list. vis.resolu is the name of the current resolution.

debug_trig()
Debug for canvas event

find_numbpix()
Numper of pixels in the zoom area.

on_scroll(*event*)
Use of scrolling function to control the level

plot_zooms()
Calculates the coordinates of the zoom and prints in both windows

pr(*var*, *message=''*)
print

press_zoom(*xpress*, *ypress*)
Zoom activated or not after pressing the left mouse button

press_zoomready_and_in()
Activated when zoom is ready and pressed is in the zoom window.

press_zoomready_and_out()
Activated when zoom is ready an pressed is outside the zoom window

stretchrect(*dx*, *dy*)
function to stretch the zoom by taking the corners

zoom_check_size()
If zoom too small reinitializes ie self.paramz.zoom_coord = [], disconnects the mouse and sets flag self.newrectcready to True

zoom_plot

class Visu.zoom_plot.**ZOOM_PLOT**(*display*, *interf*, *data*, *paramz*, *gtools*)
Makes the zoom rectangles for C and D window and the greyarea.

drawrect(*llx*, *lly*, *urx*, *ury*, *layout1=None*, *layout2=None*)
Draws rectangles with absolute coordinates llx, lly, urx, ury in C and D windows

drawrectC(*llx*, *lly*, *urx*, *ury*, *layout1=None*)
Draws the zoom rectangle in the C windows

drawrectD(*llx*, *lly*, *urx*, *ury*, *layout2=None*)
Draws the zoom rectangle in the D windows

greyzoom(*llx*, *lly*, *urx*, *ury*)
Grey area around the zoom

pr(*var*, *message=''*)
print

zoom_tools

label_2D

Illustrate simple contour plotting, contours on an image with a colorbar for the contours, and labelled contours.

See also contour_image.py.

class Visu.label_2D.**PEAKPICK**(*data*, *display*, *convert*, *paramz*)
Peakpicking

baryc_far()
Makes the points far from the self.barycenter. self.lx, self.ly defined from a small division fo the window dimensions.

barycenter_method()
Takes the labels far one from another and all the labels far from the peaks 1) Begins by making a self.barycenter and getting the labels at the opposite of the vector 'peak to self.barycenter' at the distance dist_min. 2) Makes a correction on the labels ovelapping the peaks 3) Makes a correction on the labels ovelapping each other.

correc_label_label(*correct*)
Correction between labels

correc_peak_label(*correct*)
Correction between peaks and labels

decrossing()
Avoidance of arrow crossing.

find_peaks(*thresh*, *zoom*, *maxpeaks*)
Find the peaks in the 2D dataset.

get_far(*method='barycenter'*)
Take the labels far one from another and from peaks Two methods: barycenter

make_labels(*method='barycenter'*)
Finds positions of the labels and plots. self.lx, self.ly defined from dimension of the window.

peaklabel(*i*)
function to label the peaks. Plots the labels if the labels are not calculated outside the limited range.

pr(*var*, *message=''*)
print

test_correct(*pt0*, *pt1*, *correct=False*, *kind=None*)
correct permits to chose between correction (True) and simple test (False).. kind be set to 'label_label' or 'peak_label'

Miscellaneous

Saving

class Visu.Saving.**SAVE**(*data*)
Class for savings the 2D, 3D, profiles etc.. Permits to save the object in the same location : a directory named with the date etc..

dir_save()
Makes a directory for the data if it doesn't exist The directory is named with the day, the month, the year and the hour.

pr(*var*, *message=''*)
print

prep_path_save(*namefile*)
prepares address path_save.

3.6 Processing 2D

Processing.py

This program realises the processing of an FTICR data

Created by Marc-Andre on 2011-09-23. Copyright (c) 2011 IGBMC. All rights reserved.

class processing.**Proc_Parameters**(*configfile=None*)
this class is a container for processing parameters

load(*cp*)
load from cp config file - should have been opened with ConfigParser() first

report()
print a formatted report

class processing.**Test**(*methodName='runTest'*)
tests

test_NUS()
apply a complete NUS processing test

test_intelli()
testing 'intelligent' rounding

test_proc()
apply a complete processing test

test_zf()
testing zerofilling computation

processing.**apod**(*d*, *size*, *axis=0*)
apply sin 0.5 apodisation and change size

processing.**comp_sizes**(*d0*, *zflist=None*, *szmlist=None*, *largest=8589934592*, *sizemin=1024*, *vignette=True*)

return a list with data-sizes, computed either zflist : from zerofilling index eg : (1,0,-1) szmlist : from multiplicant pairs eg : (2,2)

largest determines the largest dataset allowed sizemini determines the minimum size when downzerofilling when vignette == True (default) a minimum size data (defined by sizemini) is appended to the list

processing.**do_proc_F1**(*dinp*, *doutp*)
scan all cols of dinp, apply proc() and store into doutp

processing.**do_proc_F1_flip_modu**(*dinp*, *doutp*, *parameter*)
as do_proc_F1, but applies flip and then complex modulus() at the end

processing.**do_proc_F1_modu**(*dinp*, *doutp*)
as do_proc_F1, but applies hypercomplex modulus() at the end

processing.**do_proc_F2**(*dinp*, *doutp*)
scan all rows of dinp, apply proc() and store into doutp

processing.**do_process2D**(*dinp*, *datatemp*, *doutp*, *parameter*)
apply the processing to an input 2D data set : dinp result is found in an output file : doutp

dinp and doutp should have been created before, size of doutp will determine the processing will use a temporay file if needed

processing.**downsample2D**(*data*, *outp*, *n1*, *n2*)
takes data (a 2D) and generate a smaller dataset downsampled by factor (n1,n2) on each axis then returned data-set is n1*n2 times smaller - simply takes the mean ** Not fully tested on non powers of 2 **

processing.**intelliround**(*x*)
returns a number rounded to the nearest 'round' (easy to FT) integer

processing.**interfproc = False**
Processing for performing urQRd and/or Fista on 2D FTICR datasets. previous version was named processing2-urqrd-superresol under Linux or MacOsX : mpirun -n nbproc python processing.py (configfile.mscf) under Windows : mpiexec -n nbproc python processing.py (configfile.mscf)

processing.**iterarg**(*dinp*, *rot*, *size*, *parameter*)
an iterator used by the processing to allow multiprocessing or MPI set-up

processing.**load_input**(*name*)
load input file and returns it, in read-only mode

processing.**main**(*argv=None*)
Does the whole on-file processing, syntax is processing.py [configuration_file.mscf] if no argument is given, the standard file : process.mscf is used.

processing.**pred_sizes**(*d0*, *szmult=(1, 1)*, *sizemin=1024*)

given an input data set, determines the optimum size s1,s2 to process it with a size multiplicant of szmult

szmult (szm1, szm2) where szm1 is multiplicant for s1 and szm2 for s2 szmx = 1 : no change / 2 : size doubling / 0.5 : size halving any strictly positive value is possible, 0.2 0.33 1.1 2 2.2 5 etc...

however, axes can never get smaller than sizemin returns (si1, si2, ...) as the dataset dimension

processing.**pred_sizes_zf**(*d0*, *zf=0*, *sizemin=1024*)
given an input data set, determines the optimum size s1,s2 to process it with a zerofilling of zf zf = +n is doubling

n times along each axis zf = -n is halving n times along each axis zf = 0 is no zerofiling however, axes can never get smaller than sizemin returns (si1, si2, ...) as the dataset dimension

processing.**print_time**(*t*, *st='Processing time'*)
prints processing time

3.7 Algorithms

3.7.1 urQRd

urQRd.py

Algorithm for denoising time series, named urQRd (standing for "uncoiled random QR denoising")

main function is urQRd(data, rank) data : the series to be denoised rank : the rank of the analysis

 Marc-Andr'e Delsuc <madelsuc@unistra.fr> Lionel Chiron <lionel.chiron@gmail.com>

Created by Lionel Chiron and Marc-Andr'e on 2013-10-13.

version 2.0 28/oct/2013

Algo.urQRd.**FastHankel_prod_mat_mat**(*gene_vect*, *matrix*)
Fast Hankel structured matrix matrix product based on FastHankel_prod_mat_vec

Algo.urQRd.**FastHankel_prod_mat_vec**(*gene_vect*, *prod_vect*)
Compute product of Hankel matrix (gene_vect) by vector prod_vect. H is not computed M is the length of the result

Algo.urQRd.**Fast_Hankel2dt**(*Q*, *QH*)
returning to data from Q and QstarH Based on FastHankel_prod_mat_vec.

Algo.urQRd.**urQRd**(*data*, *k*, *orda=None*, *iterations=1*, *optk=False*)
urQRd algorithm. Name stands for uncoiled random QR denoising. From a data series return a denoised series denoised data : the series to be denoised - a (normally complex) numpy buffer k : the rank of the analysis orda : is the order of the analysis

internally, a Hankel matrix (M,N) is constructed, with M = orda and N = len(data)-orda+1 if None (default) orda = (len(data)+1)/2

iterations : the number of time the operation should be repeated

values are such that orda <= (len(data)+1)/2 k < orda N = len(data)-orda+1 Omega is (N x k) ########## BECAREFUL datasize must be different from a product of primes !!!!!!.. a processing with a datasize of 120022 for example will be 50 times longer than a procesing of a datasize of 120000. ##########

Algo.urQRd.**urQRdCore**(*data*, *Omega*)
Core of urQRd algorithm

Algo.urQRd.**vec_mean**(*M*, *L*)
Vector for calculating the mean from the sum on the antidiagonal. data = vec_sum*vec_mean

3.7.2 Linpredic

Adaptation of code from :

file CollombBurg.py author/translator Ernesto P. Adorio

UPDEPP (UP Clark) ernesto.adorio@gmail.com

Version 0.0.1 jun 11, 2010 # first release. References Burg's Method, Algorithm and Recursion, pp. 9-11

Created by Lionel on 2011-09-18. Removed the "for loops" so as to speed up using numpy capabilities. Copyright (c) 2010 IGBMC. All rights reserved.

class Algo.Linpredic.**LinpredTests**(*methodName='runTest'*)

•Testing linear prediction , Burg algorithm-

test_burg()

•testing burg algo -

Algo.Linpredic.**burg**(*m*, *x*)
Based on Collomb's C++ code, pp. 10-11 Burgs Method, algorithm and recursion

m - number of lags in autoregressive model. x - data vector to approximate.

Algo.Linpredic.**denoise**(*data*, *ar*)
returned a denoised version of "data", using "ar" polynomial first len(ar) points are untouched.

Algo.Linpredic.**predict**(*data*, *ar*, *length*)
returns a vector with additional points, predicted at the end of "data" up to total size "length", using "ar" polynomial

CHAPTER

FOUR

LICENSES

4.1 SPIKE License

This software is governed by the CeCILL license under French law and abiding by the rules of distribution of free software. You can use, modify and/ or redistribute the software under the terms of the CeCILL license as circulated by CEA, CNRS and INRIA at the following URL "http://www.cecill.info".

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors have only limited liability.

In this respect, the user's attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the software by the user in light of its specific status of free software, that may mean that it is complicated to manipulate, and that also therefore means that it is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the software's suitability as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions as regards security.

The fact that you are presently reading this means that you have had knowledge of the CeCILL license and that you accept its terms.

4.2 Secondary Licenses

4.2.1 urQRd License

Copyright (c) 2013 IGBMC. All rights reserved. Marc-Andr'e Delsuc <madelsuc@unistra.fr> Lionel Chiron <lionel.chiron@gmail.com>

This software is a computer program whose purpose is to compute urQRd denoising.

This software is governed by the CeCILL license under French law and abiding by the rules of distribution of free software. You can use, modify and/ or redistribute the software under the terms of the CeCILL license as circulated by CEA, CNRS and INRIA at the following URL "http://www.cecill.info".

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors have only limited liability.

In this respect, the user's attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the software by the user in light of its specific status of free software, that may mean that it is complicated to manipulate, and that also therefore means that it is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the software's suitability as regards

their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions as regards security.

The fact that you are presently reading this means that you have had knowledge of the CeCILL license and that you accept its terms.

4.2.2 Anaconda License

Anaconda END USER LICENSE AGREEMENT

Anaconda ("the Software Product") and accompanying documentation is licensed and not sold. The Software Product is protected by copyright laws and treaties, as well as laws and treaties related to other forms of intellectual property. Continuum Analytics Inc or its subsidiaries, affiliates, and suppliers (collectively "Continuum") own intellectual property rights in the Software Product. The Licensee's ("you" or "your") license to download, use, copy, or change the Software Product is subject to these rights and to all the terms and conditions of this End User License Agreement ("Agreement").

In addition to Continuum-licensed software, the Software product contains a collection of software packages from other sources ("Other Vendor Tools"). Continuum may also distribute updates to these packages on an "as is" basis and subject to their individual license agreements. These licenses are available either in the package itself or at http://docs.continuum.io/anaconda/licenses.html. Continuum reserves the right to change which Other Vendor Tools are provided in Anaconda.

4.2.3 Licenses of Anaconda packages

Name Version License
apptools 4.2.1 BSD
argcomplete 0.6.7 Apache Software License
astroid 1.1.1 LGPL
astropy 0.3.1 BSD
atom 0.3.7 BSD
basemap 1.0.7 PSF
beautiful-soup 4.3.1 PSF/MIT
binstar 0.5.2 BSD
biopython 1.63 BSD-like
bitarray 0.8.1 PSF
blaze 0.4.2 BSD
blist 1.3.6 BSD
blz 0.6.2 BSD
bokeh 0.4.4 New BSD
boto 2.28.0 MIT
bsdiff4 1.1.4 BSD
cairo 1.12.2 LGPL 2.1 and MPL 1.1
casuarius 1.1 LGPL
cdecimal 2.3 BSD
cffi 0.8.2 MIT
chaco 4.4.1 BSD
cheetah 2.4.4 MIT
chrpath 0.13 GPL
colorama 0.2.7 BSD

conda 3.4.3 BSD
conda-api 1.1.0 BSD
conda-build 1.3.3 BSD
configobj 5.0.5 BSD
coverage 3.7.1 BSD
cubes 0.10.2 MIT
curl 7.30.0 MIT/X derivate
cython 0.20.1 Apache 2.0
datashape 0.1.1 BSD
dateutil 2.1 BSD
decorator 3.4.0 BSD
distribute 0.6.45 PSF or ZPL
dnspython 1.10.0 as-is
docutils 0.11 Public-Domain, PSF, 2-clause BSD, GPL3
dynd-python 0.6.1 BSD
ecdsa 0.11 MIT
enable 4.3.0 BSD
enaml 0.9.1 BSD
envisage 4.4.0 BSD
faulthandler 2.3 BSD
feedparser 5.1.3 MIT
fiona 1.1.4 BSD
flake8 2.1.0 MIT
flask 0.10.1 BSD
freetype 2.4.10 FreeType License
future 0.12.0 MIT
futures 2.1.6 BSD
gdal 1.10.1 MIT
gdata 2.0.18 Apache 2.0
geos 3.3.3 LGPL
gevent 1.0 MIT
gevent-websocket 0.9.2 Apache
gevent_zeromq 0.2.5 New BSD
googlecl 0.9.12 Apache 2.0
greenlet 0.4.2 MIT
grin 1.2.1 BSD
gunicorn 18.0 MIT
h5py 2.3.0 New BSD
hdf5 1.8.9 BSD-style
html5lib 0.999 MIT
hyde 0.8.5 MIT
iopro 1.6.5 proprietary - Continuum Analytics, Inc.
ipython 2.0.0 BSD
itsdangerous 0.24 BSD License
jinja2 2.7.2 BSD
jpeg 8d Custom free software license
keyring 3.7 PSF

kiwisolver 0.1.2 BSD
launcher 0.1.2 proprietary - Continuum Analytics, Inc.
lcms 1.19 MIT
libdynd 0.6.1 BSD
libffi 3.0.13 MIT
libnetcdf 4.2.1.1 MIT
libpng 1.5.13 Open Source
libsodium 0.4.5 MIT
libtiff 4.0.2 as-is
libxml2 2.9.0 MIT
libxslt 1.1.28 MIT
llvm 3.3 Open Source
llvmpy 0.12.4 New BSD License
logilab-common 0.61.0 LGPL
lxml 3.3.4 BSD
markdown 2.4 BSD
markupsafe 0.18 BSD
mathjax 2.2 Apache
matplotlib 1.3.1 PSF-based
mayavi 4.3.1 BSD
mccabe 0.2.1 Expat
mdp 3.3 BSD
menuinst 1.0.3 BDF
mercurial 2.9.1 GPLv2
mingw 4.7 GPL
mock 1.0.1 BSD
mpi4py 1.3 BSD
mpich2 1.4.1p1 mpich license
multipledispatch 0.4.0 BSD
netcdf4 1.0.8 MIT
networkx 1.8.1 BSD
nltk 2.0.4 Apache 2.0
nose 1.3.3 LGPL
numba 0.13.1 numba license
numexpr 2.3.1 MIT
numpy 1.8.1 BSD
numpydoc 0.4 BSD
openpyxl 1.8.5 MIT/Expat
openssl 1.0.1g Apache-style
pandas 0.13.1 BSD
pandasql 0.4.2 BSD
paramiko 1.14.0 LGPL
pastedeploy 1.5.2 MIT
patchelf 0.6 GPL3
patsy 0.2.1 BSD License
pep8 1.5.6 MIT License
pil 1.1.7 PIL license

pillow 2.4.0 Standard PIL license
pip 1.5.5 MIT
pixman 0.26.2 MIT
ply 3.4 BSD
psutil 1.2.1 BSD
py 1.4.20 MIT
py2cairo 1.10.0 LGPL 2.1 and MPL 1.1
pyasn1 0.1.6 BSD
pyaudio 0.2.7 MIT
pycosat 0.6.1 MIT
pycparser 2.10 BSD
pycrypto 2.6.1 Public Domain
pycurl 7.19.3.1 LGPL and MIT/X
pyface 4.4.0 BSD
pyflakes 0.8.1 MIT
pygments 1.6 BSD
pykit 0.2.0 BSD
pylint 1.2.1 GPL
pymc 2.3.2 Academic Free License
pyodbc 3.0.7 MIT
pyparsing 2.0.1 MIT
pyqt 4.10.4 GPL
pyreadline 2.0 BSD
pysal 1.6.0 New BSD License
pysam 0.6 MIT
pytables 3.1.1 BSD
pytest 2.5.2 MIT
python 2.7.6 PSF
pytz 2014.2 MIT
pywin32 218.4 PSF
pyyaml 3.11 MIT
pyzmq 14.3.0 BSD License and GNU Library or Lesser General Public License
qt 4.8.5 LGPL
readline 6.2 GPL 3
redis 2.6.9 3-clause BSD
redis-py 2.9.1 MIT
reportlab 3.1.8 BSD
requests 2.2.1 ISC
rope 0.9.4 GPL
runipy 0.0.8 BSD
scikit-image 0.9.3 Modified BSD
scikit-learn 0.14.1 3-clause BSD
scipy 0.14.0 BSD
setuptools 3.6 PSF or ZPL
shapely 1.3.1 BSD
sip 4.15.5 GPL
six 1.6.1 MIT

sphinx 1.2.2 BSD
spyder 2.2.5 MIT
sqlalchemy 0.9.4 MIT
sqlite 3.8.4.1 Public Domain
sqlparse 0.1.11 BSD
ssh 1.8.0 LGPL
ssl_match_hostname 3.4.0.2 PSF
starcluster 0.93.3 LGPL
statsmodels 0.5.0 3-clause Modified BSD
sympy 0.7.5 New BSD
theano 0.6.0 BSD
tk 8.5.15 BSD-style
tornado 3.2.1 Apache
traits 4.4.0 BSD
traitsui 4.4.0 BSD
twisted 13.2.0 MIT
ujson 1.33 BSD
unixodbc 2.3.1 ???
util-linux 2.21 GPL
vtk 5.10.1 BSD
w3lib 1.5 BSD
werkzeug 0.9.4 BSD
whoosh 2.5.7 BSD
workerpool 0.9.2 MIT
xlrd 0.9.3 BSD
xlsxwriter 0.5.5 BSD
xlwings 0.1.0 BSD 3-clause
xlwt 0.7.5 BSD
yaml 0.1.4 MIT
yt 2.6.2 BSD
zeromq 4.0.4 LGPL
zlib 1.2.7 zlib
zope.interface 4.0.5 Zope Public License

CHAPTER

FIVE

INDICES AND TABLES

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

a

f

n

o

p

v