## Contents

1 Installation Guide 3  
1.1 Where to get nmrglue 3  
1.2 Requirements 3  
1.3 Unix/OSX Installation 3  
1.4 Windows Installation 3  
1.5 Installing from source code 3  

2 Tutorial 5  
2.1 Introduction 5  
2.2 Reading NMR files 5  
2.3 Universal dictionaries 6  
2.4 Manipulating NMR data 7  
2.5 Writing NMR files 9  
2.6 The unit_conversion object 9  
2.7 Converting between file formats 10  
2.8 Low memory reading/writing of files 10  
2.9 Processing data 11  
2.10 Using matplotlib to create figures 12  
2.11 Additional resources 14  

3 Reference Guide 15  
3.1 fileio modules 15  
3.2 process modules 105  
3.3 analysis modules 199  
3.4 util modules 241  

4 Examples 245  
4.1 Application Examples 245  
4.2 Coadd Examples 257  
4.3 Convert Examples 259  
4.4 Fitting Examples 265  
4.5 Integration Examples 270  
4.6 Interactive Examples 274  
4.7 Plotting Examples 275  
4.8 Processing Examples 286  
4.9 Separate Examples 292  
4.10 Simulation Examples 294
Release 0.7-dev

Date November 29, 2017
1.1 Where to get nmrglue

Install files for all platforms are available for download. The .tar.gz file should be used on Linux and OS X and a binary, .exe file for Windows.

1.2 Requirements

nmrglue requires NumPy and SciPy to be installed. The matplotlib and IPython packages are highly recommended. A easy way of obtaining and installing these packages is to use a Python distribution which provides these packages, such as EPD. Detailed information on installing a Scipy stack is available.

1.3 Unix/OSX Installation

After installing the above dependencies download and extract the source distribution and run:

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

1.4 Windows Installation

Download the binary installer and run it.

1.5 Installing from source code

nmrglue can also be installed from source code. See the Source Code section of the Development Guide for details on this process.
2.1 Introduction

nmrglue is a python module for reading, writing, and interacting with the spectral data stored in a number of common NMR data formats. This tutorial provides an overview of some of the features of nmrglue. A basic understanding of python is assumed which can be obtained by reading some of the python documentation. The examples in this tutorial can be run interactively from the python shell but the use of an enhanced python shell which provides non-blocking control of GUI threads, for example ipython, is recommended when trying the examples which use matplotlib. The sample data using in this tutorial is available is you wish to follow along with the same files.

2.2 Reading NMR files

nmrglue can read and write to a number of common NMR file formats. To see how simple this can be let’s read a 2D NMRPipe file.

```python
>>> import nmrglue as ng
>>> dic, data = ng.pipe.read("test.fid")
```

Here we have imported the nmrglue module and opened the NMRPipe file test.fid. nmrglue contains a number of modules for reading and writing NMR files and all of these modules have a read function which opens a file or directory containing NMR data, reads in any necessary information, and loads the spectral data into memory. The read function returns a 2-tuple containing a python dictionary with file and spectral parameters and a numpy array object containing the numeric spectral data. Currently the following file formats are supported by nmrglue with the associated module:

<table>
<thead>
<tr>
<th>Module</th>
<th>File Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>bruker</td>
<td>Bruker</td>
</tr>
<tr>
<td>pipe</td>
<td>NMRPipe</td>
</tr>
<tr>
<td>sparky</td>
<td>Sparky</td>
</tr>
<tr>
<td>varian</td>
<td>Varian/Agilent</td>
</tr>
</tbody>
</table>

Examining the data object in more detail:
We can see that this is a two dimensional data set with 1500 complex points in the direct dimension and 332 points in the indirect dimension. nmrglue takes care of converting the raw data in the file into an array of appropriate type, dimensionality, and quadrature. For complex data the last axis, typically the direct dimension, is convert to a complex data type. The other axes are not converted. In some cases not all of the information needed to represent the spectral data as a well formed numpy array is not stored in the file or the values determined automatically are incorrect. In many of these cases this information can be specified directly in the function call.

For example the read function in the varian module sometimes cannot determine the shape or fid ordering of 3D files correctly. These parameters can be explicitly provided in the function call with the shape and torder keywords. See nmrglue.varian for details.

### 2.3 Universal dictionaries

In addition to the spectral data the read function also determines various spectral parameters that were stored in the file and stores them in a python dictionary:

```python
>>> dic["FDF2SW"]
50000.0
>>> dic["FDF1LABEL"]
'15N'
```

Here we see NMRPipe files stores the spectal width of the direct dimension (50000.0 Hz) and the name of the indirect dimension (15N) as well as a number of additional parameter. Some file formats describe well the spectral data, listing a large number of parameters, other only a few. In addition, the different format express the parameters in different units and under different names. For user who are familiar with the specific file format or working with only a single file type this is not a problem, the dictionary allows direct access to these parameters. If a more uniform listing of spectral parameter is desired the guess_udic function can be used to create a ‘universal’ dictionary.

```python
>>> udic = ng.pipe.guess_udic(dic,data)
>>> udic.keys()
['ndim', 0, 1]
```

This ‘universal’ dictionary of spectral parameter contains only the most fundamental parameters, the dimensionality of the data and a dictionary of parameters for each axis numbered according to the data array ordering (the direct dimension is the highest numbered dimension). The axis dictionaries contain the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>Carrier frequency in Hz.</td>
</tr>
<tr>
<td>complex</td>
<td>True for complex data, False for magnitude data.</td>
</tr>
<tr>
<td>encoding</td>
<td>How the data is encoded, ‘states’, ‘tppi’, etc.</td>
</tr>
<tr>
<td>freq</td>
<td>True for frequency domain data, False for time domain.</td>
</tr>
<tr>
<td>label</td>
<td>String describing the axis name.</td>
</tr>
<tr>
<td>obs</td>
<td>Observation frequency in MHz.</td>
</tr>
<tr>
<td>size</td>
<td>Dimension size (R</td>
</tr>
<tr>
<td>sw</td>
<td>Spectral width in Hz.</td>
</tr>
<tr>
<td>time</td>
<td>True for time domain data, False for frequency domain.</td>
</tr>
</tbody>
</table>
For our 2D NMRPipe file these parameter for the indirect dimension are:

```python
>>> for k,v in udic[0].iteritems(): print k,v
... encoding states
car 6077.75985718
sw 5555.55615234
label 15N
complex True
time True
freq False
obs 50.6479988098
size 332
```

One note on the size key, it was designed to always match the shape of the data:

```python
>>> [udic[n]['size'] for n in range(udic['ndim'])]
[332, 1500]
```

Not all NMR files formats contain all the information necessary to determine uniquely all of the universal dictionary parameters. In these cases the dictionary will be filled with generic values (999.99, “X”, “Y”, etc) and should be updated by the user with the correct values. In converting to a ‘universal’ dictionary we have sacrificed additional information about the data which was contained in the original file in order to provide a common description of NMR data. Despite the universal dictionaries limited information, together with the data array it is sufficient for most NMR tasks. We will see later that the universal dictionary allows for conversions between file formats.

### 2.4 Manipulating NMR data

Let us return again to the data array. By providing direct access to the spectral data as a numpy array we can examine and manipulate this data using a number of simple methods as well as a number of functions. Since the `read` function moves the data into memory all this data manipulation is done without effecting the original data file.

We can use slices to examine single values in the array:

```python
>>> print data[0,0]
(42.6003+139.717j)
```

Or an whole vector:

```python
>>> print data[0]
[ 42.60026550+139.71652222j 360.07470703+223.2023468j 245.21197510+202.19010925j ..., -5.77970505 +11.27639675j -25.34334183 +0.71600127j 4.61173439 -9.05398846j]
```

And along the indirect dimension:

```python
>>> print data[:,0]
[ 4.26002655e+01+139.71652222e+00j 1.69816299e+02+9.70676041e+01j ..., -5.77970505 +11.27639675j -25.34334183 +0.71600127j 4.61173439 -9.05398846j]
```

We can do more advanced slicing:
```python
tq docs>

>>> print data[2:5,0:10]
[[ 99.46063232+271.79595947j 336.36364746+246.67727661j
  ... 233.28765869+188.69224548j 280.29260254+227.20960999j]]

>>> print data[0,::-1]
[[  4.61173439 -9.05398846j -25.34334183 +0.71600127j
  ... 245.21197510+202.19010925j
  360.07470703+223.2023468j 42.60026550+139.71652222j]]

If we just want the real or imaginary channel:

```python
tq docs>

```python
>>> print data[0,0:2].real
[ 42.6002655 360.07470703]

>>> print data[0,0:2].imag
[ 139.71652222 223.2023468]
```

We find characteristics of the data:

```python
tq docs>

```python
>>> data.min()
(-161.38414+71.787979j)

>>> data.max()
(360.07471+223.20235j)

>>> data.mean()
(0.041979135291164656+0.086375666729417669j)

>>> data.std()
23.997132358800357

>>> data.sum()
(20905.609+43015.082j)
```

Reshape or transpose the data:

```python
tq docs>

```python
>>> data.shape
(332, 1500)

>>> data.reshape(664,750).shape
(664, 750)

>>> data.transpose().shape
(1500, 332)
```

Finally we can set the value of `data` as desired. For example setting a single point:

```python
tq docs>

```python
>>> data[0,0] = (100.+100.j)

```python
tq docs>

```python
>>> data[0,0]
(100+100j)
```

Or a region:

```python
tq docs>

```python
>>> data[1]
array([ 0.+0.j, 0.+0.j, 0.+0.j, ..., 0.+0.j, 0.+0.j, 0.+0.j], dtype=complex64)

>>> data[9].imag
array([ 1., 1., 1., ..., 1., 1., 1.], dtype=float32)
```

The `numpy documentation` has additional information on the `array` object. In addition by combining `nmrglue` with `numpy` and/or `scipy` more complex data manipulation and calculation can be performed. Later we will show how these modules are used to create a full suite of processing functions.
2.5 Writing NMR files

Now that we have modified the original NMR data we can write our modification to a file. nmrglue again makes this simple:

```python
>>> ng.pipe.write("new_data.fid",dic,data)
```

Reading in both the original data and this new data we can see that they are different:

```python
>>> new_dic,new_data = ng.pipe.read("new_data.fid")
>>> ng.misc.isdatasimilar(orig_data,new_data)
False
>>> orig_data[0,0]
(42.600266+139.71652j)
>>> new_data[0,0]
(100+100j)
```

The parameter dictionary has not changed:

```python
>>> ng.misc.isdicsimilar(orig_dic,new_dic)
True
```

By default nmrglue will not overwrite existing data with the `write` function:

```python
>>> ng.pipe.write("new_data.fid",dic,data)
Traceback (most recent call last):
... IOError: File exists, recall with overwrite=True
```

But this check can be by-passed with the `overwrite` parameter:

```python
>>> ng.pipe.write("new_data.fid",dic,data,overwrite=True)
```

2.6 The unit_conversion object

Earlier we used the array index values for slicing the numpy array. For reference your data in more common NMR units nmrglue provides the `unit_conversion` object. Use the `make_uc` function to create a `unit_conversion` object:

```python
>>> dic,data = ng.pipe.read("test.ft2")
>>> uc0 = ng.pipe.make_uc(dic,data,dim=0)
>>> uc1 = ng.pipe.make_uc(dic,data,dim=1)
```

We now have unit conversion objects for both axes in the 2D spectrum. We can use these objects to determine the nearest point for a given unit:

```python
>>> uc0("100.0 ppm")
1397
>>> uc1(5000,"Hz")
2205
```

Or an exact value:
We can also convert from points to various units:

```
>>> uc0.ppm(1200)
110.57355437408664
>>> uc1.hz(100)
30692.301979064941
>>> uc0.unit(768, "percent")
37.518319491939423
```

These objects can also be used for slicing, for example to find the trace closes to 120 ppm:

```
>>> data[uc0("120ppm")]
array([- 534.28442383, - 3447.58349609, - 5216.93701172, ..., - 8258.26171875,
       - 8828.359375, -1102.84863281], dtype=float32)
```

## 2.7 Converting between file formats

nmrglue can also be used to convert between file formats using the `convert` module. For example to convert a 2D NMRPipe file to a Sparky file:

```
>>> dic, data = ng.pipe.read("test.ft2")
>>> C = ng.convert.converter()
>>> sparky_dic, sparky_data = C.to_sparky()
>>> ng.sparky.write("sparky_file.ucsf", sparky_dic, sparky_data)
```

Here we opened the NMRPipe file `test.ft2`, created a new `converter` object and loaded it with the NMRPipe data. The `converter` is then used to generate the Sparky parameter dictionary and a data array appropriate for Sparky data which is written to `sparky_file.ucsf`. All type conversions, and sign manipulation of the data array is performed internally by the `converter` object. In addition new dictionaries are created from an internal universal dictionary for the desired output. Additional examples showing how to use nmrglue to convert between NMR file formats can be found in the `Convert Examples`.

## 2.8 Low memory reading/writing of files

Up to this point we have read NMR data from files using the `read` function. This function reads the spectral data from a NMR file into the computer's memory. For small data sets this is fine, modern computer have sufficient RAM to store complete 1D and 2D NMR data sets and a few copies of the data while processing. For 3D and larger dimensionality data set this is often not desired. Reading in an entire 3D data set is not required when only a small portion must be examined for viewing or processing. With this in mind nmrglue provides methods to read only a portions of NMR data from files when it is required. This is accomplished by creating a new object which look very similar to numpy array but does not load data into memory. Rather when a particular slice is requested the `read` object opens the necessary file(s), reads in the data and returns to the user a numpy array with the data. In addition these objects have tranpose and swapaxes method and can be iterated over just as numpy arrays but without using large amounts of memory. The only limitation of these objects is that they do not support assignment, so a slice must be taken before changing the value of data. The fileio sub-modules all have some form of `read_lowmem` function which return these low-memory objects. For example reading the 2D sparky file we created earlier:
>>> dic, data = ng.sparky.read_lowmem("sparky_file.ucsf")
>>> type(data)
<class 'nmrglue.fileio.sparky.sparky_2d'>
>>> data.shape
(2048, 4096)

Slicing returns a numpy array:

```python
>>> data[0,1]
array(1601.8291015625, dtype=float32)
>>> data[0]
array([-2287.25195312, 1601.82910156, 475.85516357, ..., -4680.2265625 ,
         -72.70507812, -1402.25256348], dtype=float32)
```

The data can be transposed as a numpy array:

```python
>>> tdata = data.transpose()
>>> type(tdata)
<class 'nmrglue.fileio.sparky.sparky_2d'>
>>> tdata.shape
(4096, 2048)
>>> tdata[1,0]
array(1601.8291015625, dtype=float32)
```

These low memory usage objects can be written to disk or used in to load a conversion object just as if they were normal numpy arrays.

Similar when large data sets are to be written to disk, it often does not make sense to write the entire data set at once. For this the write_lowmem functions in the fileIO submodules provide methods for trace-by-trace or similar writing.

### 2.9 Processing data

With NMR spectral data being stored as a numpy array a number of linear algebra and signal processing functions can be applied to the data. The functions in the numpy and scipy modules offer a number of processing functions users might find useful. nmrglue provides a number of common NMR functions in the nmrglue.proc_base module, baseline related functions in nmrglue.proc_bl, and linear prediction functions in the nmrglue.proc_lp module. For example we perform some simple processing on our 2D NMRPipe file (output supressed):

```python
>>> dic, data = ng.pipe.read("test.fid")
>>> ng.proc_base.ft(data)
>>> ng.proc_base.mir_left(data)
>>> ng.proc_base.neg_left(data)
>>> ng.proc_bl.sol_sine(data)
```

These functions process only the data, they do not update the spectral parameter associated with the data. Because these values are key when examining NMR data we want functions which take into account these parameter while processing. nmrglue provides the nmrglue.pipe_proc module for processing NMRPipe data while updating the spectral properties simulanatously. Additional modules for processing other file format are being developed. Using pipe_proc is similar to using NMRPipe itself. For example to process the sample 2D NMRPipe file:

```python
>>> dic, data = ng.pipe.read("test.fid")
>>> dic, data = ng.pipe_proc.sp(dic, data, off=0.35, end=0.98, pow=1, c=1.0)
>>> dic, data = ng.pipe_proc.zf(dic, data, auto=True)
>>> dic, data = ng.pipe_proc.ps(dic, data, p0=-29.0, p1=0.0)
```
This processed file can then be written out

```python
dic, data = ng.pipe.proc.di(dic, data)
dic, data = ng.pipe.proc.tp(dic, data)
dic, data = ng.pipe.proc.sp(dic, data, off=0.35, end=0.9, pow=1, c=0.5)
dic, data = ng.pipe.proc.zf(dic, data, size=2048)
dic, data = ng.pipe.proc.ft(dic, data, auto=True)
dic, data = ng.pipe.proc.ps(dic, data, p0=0.0, p1=0.0)
dic, data = ng.pipe.proc.di(dic, data)
dic, data = ng.pipe.proc.tp(dic, data)
```

In the example above the entire data set was processed in memory. All the processing functions were applied to a set of data stored in the computers RAM after which the entire 2D data set was written to disk. For 1D and 2D data sets this is fine, but as mentioned earlier many 3D and larger data sets cannot be processed in this manner. For a 3D file what is desired is that each 2D XY plane be read, processed and saved. Then the ZX planes are read from this new file, the Z plane processed and these planes saved into the final file. In nmrglue this can be accomplished for NMRPipe files using the `iter3D object`. Currently no other file format allows such processing but development of these is planned. An example of processing a 3D NMRPipe file using a `iter3D object` can be found in `process example: process_pipe_3d`.

Additional examples showing how to use nmrglue to process NMR data can be found in the `Processing Examples`

### 2.10 Using matplotlib to create figures

A number of python plotting libraries exist which can be used in conjunction with nmrglue to produce publication quality figures. `matplotlib` is one of the more popular libraries and has the ability to output to a number of hardcopy formats as well as offering a robust interactive environment. When using `matplotlib` interactively use of `ipython` or a similar shell is recommended although the standard python shell can be used. For example to create a simple plot of a 1D spectrum (if the ipython shell is used for this example use the `-pylab` switch):

```python
>>> import pylab
>>> dic, data = ng.pipe.read("test.ft")
>>> pylab.plot(data)
[<matplotlib.lines.Line2D object at 0x8754fd0>]
>>> pylab.savefig("plot_1d.png")
```

Here we have loaded the `pylab` module from `matplotlib` and used it to plot the 1D frequency domain data of a model protein. The resulting figure is saved as `plot_1d.png`. 

```python
>>> import pylab
>>> dic, data = ng.pipe.read("test.ft")
>>> pylab.plot(data)
[<matplotlib.lines.Line2D object at 0x8754fd0>]
>>> pylab.savefig("plot_1d.png")
```
A contour plot of 2D data can be created in a similar manner:

```python
>>> pylab.cla()
>>> dic, data = ng.pipe.read("test.ft2")
>>> cl = [30000*1.2**x for x in range(20)]
>>> pylab.contour(data, cl)
<matplotlib.contour.ContourSet instance at 0x151e2f80>
>>> pylab.show()
```

The `show()` method raises an interactive window for examining the plot:

Matplotlib can be used to create more complicated figures with annotations, ppm axes and more. The *Plotting Examples* and *Interactive Examples* showcase some of this functionality. For additional information see the matplotlib webpage.
2.11 Additional resources

Detailed information about each module in nmrglue as well as the functions provided by that module can be found in the nmrglue Reference Guide or by using Python build in help system:

```python
>>> help(ng.pipe.read)
```

A number of Examples using nmrglue to interact with NMR data are available. Finally documentation for the following packages might be useful to users of nmrglue:

- numpy
- scipy
- matplotlib
- h5py
3.1 fileio modules

3.1.1 nmrglue.agilent

All the functions and classes in the nmrglue.varian are also imported to nmrglue.agilent and can be called as such. See the nmrglue.varian for documentation.

3.1.2 nmrglue.bruker

Functions for reading and writing Bruker binary (set/fid) files, Bruker JCAMP-DX parameter (acqus) files, and Bruker pulse program (pulseprogram) files.

This module is imported as nmrglue.bruker and can be called as such.

User Information

User Functions

These are functions which are targeted for users of nmrglue.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read(dir, bin_file, acqus_files, ...)</code></td>
<td>Read Bruker files from a directory.</td>
</tr>
<tr>
<td><code>write(dir, dic, data[, bin_file, ...])</code></td>
<td>Write Bruker files to disk.</td>
</tr>
<tr>
<td><code>read_pdata([dir, bin_files, procs_files, ...])</code></td>
<td>Read processed Bruker files from a directory.</td>
</tr>
<tr>
<td><code>remove_digital_filter(dic, data[, truncate, ...])</code></td>
<td>Remove the digital filter from Bruker data.</td>
</tr>
<tr>
<td><code>read_lowmem([dir, bin_file, acqus_files, ...])</code></td>
<td>Read Bruker files from a directory using minimal amounts of memory.</td>
</tr>
<tr>
<td><code>write_lowmem(dir, dic, data[, bin_file, ...])</code></td>
<td>Write Bruker files using minimal amounts of memory (trace by trace).</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read_binary(filename[, shape, cplex, big])</code></td>
<td>Read Bruker binary data from file and return dic, data pair.</td>
</tr>
<tr>
<td><code>write_binary(filename, dic, data[, ...])</code></td>
<td>Write Bruker binary data to file.</td>
</tr>
<tr>
<td><code>read_pdata_binary(filename[, shape, ...])</code></td>
<td>Read a processed Bruker binary file and return dic, data pair.</td>
</tr>
<tr>
<td><code>scale_pdata(dic, data)</code></td>
<td>Scale Bruker processed data using parameters from the procs file.</td>
</tr>
<tr>
<td><code>read_binary_lowmem(filename[, shape, cplex, big])</code></td>
<td>Read Bruker binary data from file using minimal memory.</td>
</tr>
<tr>
<td><code>write_binary_lowmem(filename, dic, data[, ...])</code></td>
<td>Write Bruker binary data to file using minimal memory (trace by trace).</td>
</tr>
<tr>
<td><code>read_jcamp(filename)</code></td>
<td>Read a Bruker JCAMP-DX file into a dictionary.</td>
</tr>
<tr>
<td><code>write_jcamp(dic, filename[, overwrite])</code></td>
<td>Write a Bruker JCAMP-DX file from a dictionary.</td>
</tr>
<tr>
<td><code>read_pprog(filename)</code></td>
<td>Read a Bruker pulse program (pulseprogram) file.</td>
</tr>
<tr>
<td><code>write_pprog(filename, dic[, overwrite])</code></td>
<td>Write a minimal Bruker pulse program to file.</td>
</tr>
<tr>
<td><code>guess_udic(dic, data[, strip_fake])</code></td>
<td>Guess parameters of universal dictionary from dic, data pair.</td>
</tr>
<tr>
<td><code>create_dic(udic)</code></td>
<td>Create a Bruker parameter dictionary from a universal dictionary.</td>
</tr>
</tbody>
</table>

nmrglue.fileio.bruker.read

nmrglue.fileio.bruker.read( `dir='.', bin_file=None, acqus_files=None, pprog_file=None, shape=None, cplex=None, big=None, read_pulseprogram=True, read_acqus=True, procs_files=None, read_procs=True)`

Read Bruker files from a directory.

Parameters dir : str

- Directory to read from.

bin_file : str, optional

- Filename of binary file in directory. None uses standard files.

acqus_files : list, optional

- List of filename(s) of acqus parameter files in directory. None uses standard files.

pprog_file : str, optional

- Filename of pulse program in directory. None uses standard files.

shape : tuple, optional

- Shape of resulting data. None will guess the shape from the spectral parameters.

cplex : bool, optional

- True is direct dimension is complex, False otherwise. None will guess quadrature from spectral parameters.

big : bool or None, optional

- Endiness of binary file. True of big-endian, False for little-endian, None to determine endiness from acqus file(s).

read_pulseprogram : bool, optional

- True to read pulse program, False prevents reading.

read_acqus : bool, optional

Chapter 3. Reference Guide
True to read acqus files(s), False prevents reading.

**procs_files** : list, optional

List of filename(s) of procs parameter files in directory. None uses standard files.

**read_procs** : bool, optional

True to read procs files(s), False prevents reading.

**Returns**

**dic** : dict

Dictionary of Bruker parameters.

**data** : ndarray

Array of NMR data.

**See also:**

*read_pdata* Read Bruker processed files.

*read_lowmem* Low memory reading of Bruker files.

*write* Write Bruker files.

**nmrglue.fileio.bruker.write**

**nmrglue.fileio.bruker.write**(dir, dic, data, bin_file=None, acqus_files=None, pprog_file=None, overwrite=False, big=None, write_prog=True, write_acqus=True)

Write Bruker files to disk.

**Parameters**

**dir** : str

Directory to write files to.

**dic** : dict

Dictionary of Bruker parameters.

**data** : array_like

Array of NMR data.

**bin_file** : str, optional

Filename of binary file in directory. None uses standard files.

**acqus_files** : list, optional

List of filename(s) of acqus parameter files in directory. None uses standard files.

**pprog_file** : str, optional

Filename of pulse program in directory. None uses standard files.

**overwrite** : bool, optional

Set True to overwrite files, False will raise a Warning if files exist.

**big** : bool or None, optional

Endiness of binary file. True of big-endian, False for little-endian, None to determine endiness from Bruker dictionary.

**write_pprog** : bool, optional

True to write the pulse program file, False prevents writing.
nmrglue.fileio.bruker.read_pdata

nmrglue.fileio.bruker.read_pdata (dir='.', bin_files=None, procs_files=None, read_procs=True, acqus_files=None, read_acqus=True, scale_data=True, shape=None, submatrix_shape=None, all_components=False, big=None)

Read processed Bruker files from a directory.

In Topspin and other programs this data is typically scaled by dividing by $2^{-\text{NC\_proc}}$ where \text{NC\_proc} is defined in the procs file. This scaling can be accomplished by setting the scale_data parameter to True.

**Parameters**

- **dir**: str
  - Directory to read from.

- **bin_files**: list of str, optional
  - List of filename of binary file in directory. None uses standard files.

- **procs_files**: list, optional
  - List of filename(s) of procs parameter files in directory. None uses standard files.

- **read_procs**: bool, optional
  - True to read procs files(s), False prevents reading.

- **acqus_files**: list, optional
  - List of filename(s) of acqus parameter files in directory. None uses standard files.

- **read_acqus**: bool, optional
  - True to read acqus files(s), False prevents reading.

- **scale_data**: bool, optional
  - True, the default, to apply scaling defined in the procs file. The data should almost always be scaled. False, returns the data as it appears in the file.

- **shape**: tuple, optional
  - Shape of resulting data. None will guess the shape from the parameters in the procs file(s).

- **submatrix_shape**: tuple, optional
  - Shape of submatrix for 2D+ data. None will guess the shape from the metadata in the procs file(s).

- **all_components**: bool
  - True to return a list of all components, False returns just the all real component (1r, 2rr, 3rrr, etc).

- **big**: bool or None, optional
Endiness of binary file. True of big-endian, False for little-endian, None to determine endiness from procs file(s).

**Returns**

- **dic**: dict
  - Dictionary of Bruker parameters.
- **data**: ndarray or list
  - Array of NMR data. If all_compoents is True this is a list of array with each quadrature component.

**Notes**

There is currently no support for writing Bruker processed files or reading processed files using minimal memory.

### nmrglue.fileio.bruker.remove_digital_filter

**nmrglue.fileio.bruker.remove_digital_filter** *(dic, data, truncate=True, post_proc=False)*

Remove the digital filter from Bruker data.

**Parameters**

- **dic**: dict
  - Dictionary of Bruker parameters.
- **data**: ndarray
  - Array of NMR data to remove digital filter from.
- **truncate**: bool, optional
  - True to truncate the phase shift prior to removing the digital filter. This typically produces a better looking spectrum but may remove useful data. False uses a non-truncated phase.
- **post_proc**: bool, optional
  - True if the digital filter is to be removed post processing, i.e after fourier transformation. The corrected FID will not be returned, only a corrected spectrum in the frequency dimension will be returned.

**Returns**

- **ndata**: ndarray
  - Array of NMR data with digital filter removed

**See also:**

- **rm_dig_filter** Remove digital filter by specifying parameters.

### nmrglue.fileio.bruker.read_lowmem

**nmrglue.fileio.bruker.read_lowmem** *(dir='.', bin_file=None, acqus_files=None, pprog_file=None, shape=None, cplex=None, big=None, read_pulseprogram=True, read_acqus=True, procs_files=None, read_procs=True)*

Read Bruker files from a directory using minimal amounts of memory.

**See** **read()** **for Parameters.**
Returns `dic` : dict

Dictionary of Bruker parameters.

`data` : array_like

Low memory object which can access NMR data on demand.

See also:

**read** Read Bruker files.

**write_lowmem** Write Bruker files using minimal amounts of memory.

**nmrglue.fileio.bruker.write_lowmem**

```
nmrglue.fileio.bruker.write_lowmem(dir, dic, data, bin_file=None, acqus_files=None, pprog_file=None, overwrite=False, big=None, write_prog=True, write_acqus=True)
```

Write Bruker files using minimal amounts of memory (trace by trace).

See `write()` for Parameters.

See also:

**write** Write Bruker files.

**read_lowmem** Read Bruker files using minimal amounts of memory.

**nmrglue.fileio.bruker.read_binary**

```
mnrglue.fileio.bruker.read_binary(filename, shape=1, cplex=True, big=True)
```

Read Bruker binary data from file and return `dic,data` pair.

If data cannot be reshaped as described a 1D representation of the data will be returned after printing a warning message.

**Parameters**

- `filename` : str
  
  Filename of Bruker binary file.

- `shape` : tuple
  
  Tuple describing shape of resulting data.

- `cplex` : bool
  
  Flag indicating if direct dimension is complex.

- `big` : bool
  
  Endianness of binary file, True for big-endian, False for little-endian.

**Returns**

- `dic` : dict
  
  Dictionary containing “FILE_SIZE” key and value.

- `data` : ndarray
  
  Array of raw NMR data.

See also:
**read_binary_lowmem** Read Bruker binary file using minimal memory.

**nmrglue.fileio.bruker.write_binary**

```
write_binary(filename, dic, data, overwrite=False, big=True)
```

Write Bruker binary data to file.

**Parameters**

- **filename** : str
  Filename to write to.

- **dic** : dict
  Dictionary of Bruker parameters.

- **data** : ndarray
  Array of NMR data.

- **overwrite** : bool
  True to overwrite files, False will raise a Warning if file exists.

- **big** : bool
  Endianness to write binary data with True of big-endian, False for little-endian.

**See also:**

**write_binary_lowmem** Write Bruker binary data using minimal memory.

**nmrglue.fileio.bruker.read_pdata_binary**

```
read_pdata_binary(filename, shape=None, submatrix_shape=None, big=True)
```

Read a processed Bruker binary file and return dic, data pair.

If data cannot be reshaped as described a 1D representation of the data will be returned after printing a warning message.

**Parameters**

- **filename** : str
  Filename of Bruker binary file.

- **shape** : tuple
  Shape of resulting data. None will return 1D data.

- **submatrix_shape** : tuple
  Tuple describing shape of resulting data. None will return 1D data.

- **big** : bool
  Endianness of binary file, True for big-endian, False for little-endian.

**Returns**

- **dic** : dict
  Dictionary containing “FILE_SIZE” key and value.

- **data** : ndarray
  Array of raw NMR data.
**nmrglue.fileio.bruker.scale_pdata**

```
nmrglue.fileio.bruker.scale_pdata(dic, data)
```

Scale Bruker processed data using parameters from the procs file.

**Parameters**

- `dic` : dict
  - Dictionary of Bruker parameters.
- `data` : ndarray
  - Array of NMR data.

**Returns**

- `sdata` : array
  - Scaled data.

**nmrglue.fileio.bruker.read_binary_lowmem**

```
mrglue.fileio.bruker.read_binary_lowmem(filename, shape=1, cplex=True, big=True)
```

Read Bruker binary data from file using minimal memory.

Raises ValueError if shape does not agree with file size. See `read_binary()` for Parameters.

**Returns**

- `dic` : dict
  - Dictionary containing “FILE_SIZE” key and value.
- `data` : array_like
  - Low memory object which can access NMR data on demand.

See also:

- `read_binary` Read Bruker binary file.

**nmrglue.fileio.bruker.write_binary_lowmem**

```
mrglue.fileio.bruker.write_binary_lowmem(filename, dic, data, overwrite=False, big=True)
```

Write Bruker binary data to file using minimal memory (trace by trace).

See `write_binary()` for Parameters.

See also:

- `write_binary` Write Bruker binary data to file.

**nmrglue.fileio.bruker.read_jcamp**

```
mrglue.fileio.bruker.read_jcamp(filename)
```

Read a Bruker JCAMP-DX file into a dictionary.

Creates two special dictionary keys _coreheader and _comments Bruker parameter “$FOO” are extracted into strings, floats or lists and assigned to dic[“FOO”]

**Parameters**

- `filename` : str
  - Filename of Bruker JCAMP-DX file.

**Returns**

- `dic` : dict
Dictionary of parameters in file.

See also:

**write_jcamp** Write a Bruker JCAMP-DX file.

**Notes**

This is not a fully functional JCAMP-DX reader, it is only intended to read Bruker acqus (and similar) files.

```python
nmrglue.fileio.bruker.write_jcamp
```

**nmrglue.fileio.bruker.write_jcamp**

**nmrglue.fileio.bruker.write_jcamp**(*dic*, *filename*, *overwrite=False*)

Write a Bruker JCAMP-DX file from a dictionary.

Written file will differ slightly from Bruker’s JCAMP-DX files in that all multi-value parameters will be written on multiple lines. Bruker is inconsistent on what is written to a single line and what is not. In addition line breaks may be slightly different but will always be within JCAMP-DX specification. Finally long floating point values may lose precision when writing.

For example:

```text
##$QS= (0..7)83 83 83 83 83 83 83 22
```

will be written as

```text
##$QS= (0..7) 83 83 83 83 83 83 83 22
```

**Parameters**

**dic** : dict

Dictionary of parameters to write

**filename** : str

Filename of JCAMP-DX file to write

**overwrite** : bool, optional

True to overwrite an existing file, False will raise a Warning if the file already exists.

See also:

**read_jcamp** Read a Bruker JCAMP-DX file.

```python
nmrglue.fileio.bruker.read_pprog
```

**nmrglue.fileio.bruker.read_pprog**

**nmrglue.fileio.bruker.read_pprog**(*filename*)

Read a Bruker pulse program (pulseprogram) file.

Resulting dictionary contains the following keys:

<table>
<thead>
<tr>
<th>key</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>dictionary of variables assigned in pulseprogram</td>
</tr>
<tr>
<td>incr</td>
<td>list of lists containing increment times</td>
</tr>
<tr>
<td>loop</td>
<td>list of loop multipliers</td>
</tr>
<tr>
<td>phase</td>
<td>list of lists containing phase elements</td>
</tr>
<tr>
<td>ph_extra</td>
<td>list of lists containing comments at the end of phase lines</td>
</tr>
</tbody>
</table>
The incr, phase and ph_extra lists match up with loop list. For example incr[0], phase[0] and ph_extra[0] are all increment and phase commands with comments which occur during loop 0 which has loop[0] steps.

**Parameters**

- **filename**: str
  
  Filename of pulseprogram file to read from.

**Returns**

- **dic**: dict
  
  A dictionary with keys described above.

**See also:**

- *write_pprog* Write a Bruker pulse program to file.

### nmrglue.fileio.bruker.write_pprog

```

nmrglue.fileio.bruker.write_pprog

nmrglue.fileio.bruker.write_pprog(filename, dic, overwrite=False)
```

Write a minimal Bruker pulse program to file.

**DO NOT TRY TO RUN THE RESULTING PULSE PROGRAM**

This pulse program should return the same dictionary when read using read_pprog, nothing else. The pulse program will be nonsense.

**Parameters**

- **filename**: str
  
  Filename of file to write pulse program to.

- **dic**: dict
  
  Dictionary of pulse program parameters.

- **overwrite**: bool, optional
  
  True to overwrite an existing file, False will raise a Warning if the file already exists.

**See also:**

- *read_pprog* Read a Bruker pulse program.

### nmrglue.fileio.bruker.guess_udic

```

nmrglue.fileio.bruker.guess_udic

nmrglue.fileio.bruker.guess_udic(dic, data, strip_fake=False)
```

Guess parameters of universal dictionary from dic, data pair.

**Parameters**

- **dic**: dict
  
  Dictionary of Bruker parameters.

- **data**: ndarray
  
  Array of NMR data.

- **strip_fake**: bool
  
  If data is proceed (i.e. read using *bruker.read_pdata*) and the Bruker processing parameters STSI and/or STSR are set, the returned sweep width and carrier frequencies is changed to values that are incorrect but instead can are intended to trick the normal unit_conversion object into producing the correct result.

**Returns**

- **udic**: dict
  
  Dictionary of parameters that can be saved.
Universal dictionary of spectral parameters.

**nmrglue.fileio.bruker.create_dic**

```

nmrglue.fileio.bruker.create_dic(udic)
Create a Bruker parameter dictionary from a universal dictionary.
```

**Parameters**

`udic : dict`
Universal dictionary of spectral parameters.

**Returns**

`dic : dict`
Dictionary of Bruker parameters.

---

**Developer Information**

from __future__ import print_function, division

```
__developer_info__ = """Bruker file format information

Bruker binary files (ser/fid) store data as an array of int32s whose endiness is determined by the parameter BYTORDA (1 = big endian, 0 = little endian). Typically the direct dimension is digitally filtered. The exact method of removing this filter is unknown but an approximation is available.

Bruker JCAMP-DX files (acqus, etc) are text file which are described by the JCAMP-DX standard. Bruker parameters are prefixed with a ‘$’. 

Bruker pulseprogram files are text files described in various Bruker manuals. Of special important are lines which describe external variable assignments (surrounded by “’s), loops (begin with lo), phases (contain ip of dp) or
```

**Developer Functions**

These functions are typically not used directly by users. Developers who want fine control over Bruker files may be interested in these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create_data(data)</code></td>
<td>Create a bruker data array (recast into a complex128 or int32)</td>
</tr>
<tr>
<td><code>add_axis_to_udic(udic, dic, udim, strip_fake)</code></td>
<td>Add axis parameters to a udic.</td>
</tr>
<tr>
<td><code>create_acqus_dic(adic[, direct])</code></td>
<td>Create a Bruker acqus dictionary from an Universal axis dictionary.</td>
</tr>
<tr>
<td><code>guess_shape(dic)</code></td>
<td>Determine data shape and complexity from Bruker dictionary.</td>
</tr>
<tr>
<td><code>guess_shape_and_submatrix_shape(dic)</code></td>
<td>Guess the data shape and the shape of the processed data submatrix.</td>
</tr>
<tr>
<td><code>get_data(f, big)</code></td>
<td>Get binary data from file object with given endiness.</td>
</tr>
<tr>
<td><code>put_data(f, data[, big])</code></td>
<td>Put data to file object with given endiness.</td>
</tr>
<tr>
<td><code>complexify_data(data)</code></td>
<td>Complexify data packed real, imag.</td>
</tr>
<tr>
<td><code>uncomplexify_data(data_in)</code></td>
<td>Uncomplexify data (pack real,imag) into a int32 array.</td>
</tr>
<tr>
<td><code>reorder_submatrix(data, shape, submatrix_shape)</code></td>
<td>Reorder processed binary Bruker data.</td>
</tr>
<tr>
<td><code>rm_dig_filter(data, decim, dspfvs[, grpdly, ...])</code></td>
<td>Remove the digital filter from Bruker data.</td>
</tr>
<tr>
<td><code>parse_jcamp_line(line, l)</code></td>
<td>Parse a single JCAMP-DX line.</td>
</tr>
<tr>
<td><code>parse_jcamp_value(text)</code></td>
<td>Parse value text from Bruker JCAMP-DX file returning the value.</td>
</tr>
</tbody>
</table>

---

3.1. fileio modules
Table 3.2 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>write_jcamp_pair(f, key, value)</code></td>
<td>Write out a line of a JCAMP file.</td>
</tr>
<tr>
<td><code>read_acqus_file([dir, acqus_files])</code></td>
<td>Read Bruker acquisition files from a directory.</td>
</tr>
<tr>
<td><code>read_procs_file([dir, procs_files])</code></td>
<td>Read Bruker processing files from a directory.</td>
</tr>
</tbody>
</table>

### nmrglue.fileio.bruker.create_data

**nmrglue.fileio.bruker.create_data(data)**

Create a bruker data array (recast into a complex128 or int32)

### nmrglue.fileio.bruker.add_axis_to_udic

**nmrglue.fileio.bruker.add_axis_to_udic(udic, dic, udim, strip_fake)**

Add axis parameters to a udic.

Parameters:

- **udic**: dict
  - Universal dictionary to update, modified in place.
- **dic**: dict
  - Bruker dictionary used to determine axes parameters.
- **udim**: int
  - Universal dictionary dimension to update.
- **strip_fake**: bool
  - See `bruker.guess_udic`

### nmrglue.fileio.bruker.create_acqus_dic

**nmrglue.fileio.bruker.create_acqus_dic(adic, direct=False)**

Create a Bruker acqus dictionary from an Universal axis dictionary. Set direct=True for direct dimension.

### nmrglue.fileio.bruker.guess_shape

**nmrglue.fileio.bruker.guess_shape(dic)**

Determine data shape and complexity from Bruker dictionary.

Returns:

- **shape**: tuple
  - Shape of data in Bruker binary file (R+I for all dimensions).
- **cplex**: bool
  - True for complex data in last (direct) dimension, False otherwise.

### nmrglue.fileio.bruker.guess_shape_and_submatrix_shape

**nmrglue.fileio.bruker.guess_shape_and_submatrix_shape(dic)**

Guess the data shape and the shape of the processed data submatrix.
nmrglue.fileio.bruker.get_data

nmrglue.fileio.bruker.get_data(f, big)
Get binary data from file object with given endiness.

nmrglue.fileio.bruker.put_data

nmrglue.fileio.bruker.put_data(f, data, big=True)
Put data to file object with given endiness.

nmrglue.fileio.bruker.complexify_data

nmrglue.fileio.bruker.complexify_data(data)
Complexify data packed real, imag.

nmrglue.fileio.bruker.uncomplexify_data

nmrglue.fileio.bruker.uncomplexify_data(data_in)
Uncomplexify data (pack real, imag) into a int32 array.

nmrglue.fileio.bruker.reorder_submatrix

nmrglue.fileio.bruker.reorder_submatrix(data, shape, submatrix_shape)
Reorder processed binary Bruker data.

Parameters
data : array
shape : tuple
Shape of final data.
submatrix_shape : tuple
Shape of submatrix.

Returns
rdata : array
Array in which data has been reordered and correctly shaped.

nmrglue.fileio.bruker.rm_dig_filter

nmrglue.fileio.bruker.rm_dig_filter(data, decim, dspfvs, grpdl=0, truncate_grpdl=True, post_proc=False)
Remove the digital filter from Bruker data.

Parameters
data : ndarray
Array of NMR data to remove digital filter from.
decim : int
Decimation rate (Bruker DECIM parameter).
dspfvs : int
Firmware version (Bruker DSPFVS parameter).

**grpdly** : float, optional

Group delay. (Bruker GRPDLY parameter). When non-zero decim and dspfvs are ignored.

**truncate_grpdly** : bool, optional

True to truncate the value of grpdly provided or determined from the decim and dspfvs parameters before removing the digital filter. This typically produces a better looking spectrum but may remove useful data. False uses a non-truncated grpdly value.

**post_proc** : bool, optional

True if the digital filter is to be removed post processing, i.e after fourier transformation. The corrected time domain data will not be returned, only the corrected spectrum in the frequency dimension will be returned.

**Returns**  ndata : ndarray

Array of NMR data with digital filter removed.

**See also:**

*remove_digital_filter* Remove digital filter using Bruker dictionary.

**nmrglue.fileio.bruker.parse_jcamp_line**

**nmrglue.fileio.bruker.parse_jcamp_line(line, f)**

Parse a single JCAMP-DX line.

Extract the Bruker parameter name and value from a line from a JCAMP-DX file. This may entail reading additional lines from the fileobj f if the parameter value extends over multiple lines.

**nmrglue.fileio.bruker.parse_jcamp_value**

**nmrglue.fileio.bruker.parse_jcamp_value(text)**

Parse value text from Bruker JCAMP-DX file returning the value.

**nmrglue.fileio.bruker.write_jcamp_pair**

**nmrglue.fileio.bruker.write_jcamp_pair(f, key, value)**

Write out a line of a JCAMP file.

A line might actually be more than one line of text for arrays.

**nmrglue.fileio.bruker.read_acqus_file**

**nmrglue.fileio.bruker.read_acqus_file(dir='.', acqus_files=None)**

Read Bruker acquisition files from a directory.

**Parameters**  dir : str

Directory to read from.

acqus_files : list, optional
List of filename(s) of acqus parameter files in directory. None uses standard files.

Returns dic : dict

Dictionary of Bruker parameters.

**nmrglue.fileio.bruker.read_procs_file**

Read Bruker processing files from a directory.

**Parameters**

- **dir** : str
  Directory to read from.
- **procs_files** : list, optional
  List of filename(s) of procs parameter files in directory. None uses standard files.

Returns dic : dict

Dictionary of Bruker parameters.

**Developer Classes**

<table>
<thead>
<tr>
<th>bruker_nd(filename, fshape, cplex, big[, order])</th>
<th>Emulate a ndarray objects without loading data into memory for low memory reading of Bruker fid/ser files.</th>
</tr>
</thead>
</table>

**nmrglue.fileio.bruker.bruker_nd**

Emulate a ndarray objects without loading data into memory for low memory reading of Bruker fid/ser files.

- slicing operations return ndarray objects.
- can iterate over with expected results.
- transpose and swapaxes methods create a new objects with correct axes ordering.
- has ndim, shape, and dtype attributes.

**Parameters**

- **filename** : str
  Filename of Bruker binary file.
- **fshape** : tuple
  Shape of NMR data.
- **cplex** : bool
  Flag indicating if direct dimension is complex.
- **big** : bool
  Endianess of data. True for big-endian, False for little-endian.
- **order** : tuple
  Ordering of axis against file.
__init__ (filename, fshape, cplex, big[, order=None])
Create and set up object.

Methods

__init__ (filename, fshape, cplex, big[, order]) Create and set up object.
swapaxes(axis1, axis2) Return object with axis1 and axis2 interchanged.
transpose(*axes) Return object with axes transposed.

3.1.3 nmrglue.convert

Functions to convert between NMR file formats

This module is imported as nmrglue.convert and can be called as such.

User Information

User Classes

This class is targeted for users of nmrglue.

class nmrglue.fileio.convert.converter
    Object which allows conversion between NMR file formats, including low memory data objects.

Conversion between NMR file formats with this class involves three steps. First a new converter object must be created. Then the converter must be loaded with data using a `from_` method. Finally, the dictionary and data representation of a NMR data in the desired format is extracted using a `to_` method. This can then be written to disk.

Example conversion:

```python
dic, data = ng.varian.read("varian_dir")
C = ng.convert.converter()
C.from_varian(vdic, vdata)
pdic, pdata = C.to_pipe()
ng.pipe.write("test.fid", pdic, pdata)
```

Spectral parameters can be provided directly by passing a Universal dictionary to any of the `from_` methods. If not provided the spectral parameters are guessed from the file format's dictionary of parameters.

`from_bruker (dic, data, udic=None, remove_digital_filter=False)`
Load converter with Bruker data.

**Parameters**

dic : dict
    Dictionary of Bruker parameters.

data : array_like
    NMR data.

udic : dict, optional
    Universal dictionary, if not provided will be guesses from dic.

remove_digital_filter : bool, optional
True to remove the Bruker digital filter. Do not use this option with low memory data or when the \texttt{udic} parameter is specified. False leave the digital filter in place.

\texttt{from\_pipe}(\texttt{dic, data, udic=None})

Load converter with NMRPipe data.

\textbf{Parameters} \texttt{dic : dict}

Dictionary of NMRPipe parameters.

\textbf{data : array\_like}

NMR data.

\textbf{udic : dict, optional}

Universal dictionary, if not provided will be guesses from \texttt{dic}.

\texttt{from\_rnmrtk}(\texttt{dic, data, udic=None, agilent\_compatible=False})

Load converter with RNMRTK data.

\textbf{Parameters} \texttt{dic : dict}

Dictionary of RNMRTK parameters.

\textbf{data : array\_like}

NMR data.

\textbf{udic : dict, optional}

Universal dictionary, if not provided will be guesses from \texttt{dic}.

\textbf{agilent\_compatible : bool, optional}

True when RNMRTK data is being compared to Agilent/Varian data.

\texttt{from\_sparky}(\texttt{dic, data, udic=None})

Load converter with Sparky data.

\textbf{Parameters} \texttt{dic : dict}

Dictionary of Sparky parameters.

\textbf{data : array\_like}

NMR data.

\textbf{udic : dict, optional}

Universal dictionary, if not provided will be guesses from \texttt{dic}.

\texttt{from\_universal}(\texttt{dic, data})

Load converter with Universal data.

\textbf{Parameters} \texttt{dic : dict}

Dictionary of universal parameters.

\textbf{data : array\_like}

NMR data.

\texttt{from\_varian}(\texttt{dic, data, udic=None})

Load converter with Agilent/Varian data.

\textbf{Parameters} \texttt{dic : dict}

Dictionary of Agilent/Varian parameters.
data : array_like
    NMR data.
udic : dict, optional
    Universal dictionary, if not provided will be guesses from dic.

to_bruker ()
    Return Bruker format data.

    Returns dic : dict
        Dictionary of Bruker parameters.
        data : array_like
            NMR data in Bruker format.

to_pipe (datetimeobj=datetime.datetime(2017, 11, 29, 18, 21, 13, 32638))
    Return NMRPipe format data.

    Parameters datetime : datetime object, optional
        Datetime object to include in the NMRPipe parameters. The current date and time is
        used by default.

    Returns dic : dict
        Dictionary of NMRPipe parameters.
        data : array_like
            NMR data in NMRPipe format.

to_rnmrtk (agilent_compatible=False, dim_order=None)
    Return RNMRTK format data.

    Parameters agilent_compatible : bool, optional
        True when RNMRTK data is being compared to Agilent/Varian data.
    dim_order : list, optional
        List mapping axis numbers in the universal dictionary to the to the order in which they
        will appear in the RNMRTK dictionary. If None, the default, [0, 1, 2, ...] will be used.

    Returns dic : dict
        Dictionary of RNMRTK parameters.
        data : array_like
            NMR data in RNMRTK format.

to_sparky (datetimeobj=datetime.datetime(2017, 11, 29, 18, 21, 13, 32667), user='user')
    Return Sparky format data.

    Parameters datetime : datetime object, optional
        Datetime object to include in the Sparky parameters. The current date and time is used
        by default.
    user : str, optional
        Username to include in the Sparky parameters. ‘user’ is the default.

    Returns dic : dict
Dictionary of Sparky parameters.

**data**: array_like
NMR data in Sparky format.

**to_universal()**
Return Universal format data.

**Returns**
  **dic**: dict
  Dictionary of Universal parameters.
  **data**: array_like
  NMR data in format as provided.

**to_varian()**
Return Agilent/Varian format data.

**Returns**
  **dic**: dict
  Dictionary of Agilent/Varian parameters.
  **data**: array_like
  NMR data in Agilent/Varian format.

**Developer Information**

**Developer Classes**

These classes are typically not used directly by users. Developers who want fine control over file conversion will be interested in these classes

```
class nmrglue.fileio.convert.udata_nd(edata, iproc, oproc, odtype, order=None)
```
Wrap other fileiobase.data_nd derived objects with input/output conversion when slices are requested.

- slicing operations return ndarray objects.
- can iterate over with expected results.
- transpose and swapaxes methods create a new objects with correct axes ordering.
- has ndim, shape, and dtype attributes.

**Parameters**
  **edata**: fileiobase.data_nd derived object
  Data object to wrap.
  **iproc**: dict
  Dictionary of processing required by input format.
  **oproc**: dict
  Dictionary of processing required by output format.
  **odtype**: dtype
  Output dtype.
  **order**: tuple
  Axis ordering relative to input data.
Notes

The iproc and oproc dictionary can contain the following keys and values.

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt_id_sign</td>
<td>True/False</td>
<td>True alternates signs along indirect dims.</td>
</tr>
<tr>
<td>realfactor</td>
<td>float</td>
<td>Real channel scaling factor.</td>
</tr>
<tr>
<td>imagfactor</td>
<td>float</td>
<td>Imaginary channel scaling factor.</td>
</tr>
</tbody>
</table>

3.1.4 nmrglue.fileiobase

fileiobase provides general purpose NMR file IO functions and classes used by multiple nmrglue.fileio modules. This module is imported as nmrglue.fileiobase and can be called as such.

Developer Information

Developer Functions

These functions are typically not used directly by users. They are called by high level functions.

```python
create_blank_udic(ndim) Create a blank universal dictionary for a spectrum of dimension ndim.
uc_from_udic(udic[, dim]) Create a unit conversion object from a Universal dictionary.
uc_from_freqscale(scale, obs[, unit]) Create a unit conversion object from a spectrum frequency scale axis.
open_towrite(filename[, overwrite, mode]) Open filename for writing and return file object
ndfrom_iter(shape, slices)
ndto_iter(shape, slices)
ndtofrom_iter(shape, slices)
size_and_ndtofrom_iter(shape, slices)
index2trace_flat(shape, index) Calculate trace number from shape and index of all indirect dimensions
trace2index_flat(shape, ntrace) Calculate the index of a trace assuming a flat structure
index2trace_opp(shape, index) Calculate trace number from shape and index of all indirect dimensions assuming a phase ordering opposite the time increments.
trace2index_opp(shape, ntrace) Calculate the index of a trace assuming opposite phase/time increment
index2trace_reg(shape, index) Calculate trace number from shape and index of all indirect dimensions assuming the same phase and time ordering.
trace2index_reg(shape, ntrace) Calculate the index of a trace assuming the same phase/time increment
```

nmrglue.fileio.fileiobase.create_blank_udic

nmrglue.fileio.fileiobase.create_blank_udic(ndim)
Create a blank universal dictionary for a spectrum of dimension ndim.
nmrglue.fileio.fileiobase.uc_from_udic

nmrglue.fileio.fileiobase.uc_from_udic(udic, dim=-1)
Create a unit conversion object from a Universal dictionary.

Parameters
udic: dic
Universal dictionary of spectral parameters.
dim: int, optional
Dimension number to create unit conversion object for. Default is for last dimension.

Returns
uc: unit conversion object.
Unit conversion object for given dimension.

nmrglue.fileio.fileiobase.uc_from_freqscale

nmrglue.fileio.fileiobase.uc_from_freqscale(scale, obs, unit=’ppm’)
Create a unit conversion object from a spectrum frequency scale axis.

Parameters
scale: array like
array of spectrum axis
obs: float
Observation frequency in MHz.
unit: {‘ppm’, ‘hz’, ‘khz’}:
The unit of the scale axis.

Returns
uc: unit conversion object.
Unit conversion object for given axis.

nmrglue.fileio.fileiobase.open_towrite

nmrglue.fileio.fileiobase.open_towrite(filename, overwrite=False, mode=’wb’)
Open filename for writing and return file object
Function checks if file exists (and raises IOError if overwrite=False) and creates necessary directories as needed.

nmrglue.fileio.fileiobase.ndfrom_iter

nmrglue.fileio.fileiobase.ndfrom_iter(shape, slices)

nmrglue.fileio.fileiobase.ndto_iter

nmrglue.fileio.fileiobase.ndto_iter(shape, slices)

nmrglue.fileio.fileiobase.ndtofrom_iter

nmrglue.fileio.fileiobase.ndtofrom_iter(shape, slices)
nmrglue.fileio.fileiobase.size_and_ndtofrom_iter

\texttt{nmrglue.fileio.fileiobase.size\_and\_ndtofrom\_iter(\textit{shape}, \textit{slices})}

nmrglue.fileio.fileiobase.index2trace_flat

\texttt{nmrglue.fileio.fileiobase.index2trace\_flat(\textit{shape}, \textit{index})}
\hspace{1em}Calculate trace number from shape and index of all indirect dimensions assuming a flat structure

nmrglue.fileio.fileiobase.trace2index_flat

\texttt{nmrglue.fileio.fileiobase.trace2index\_flat(\textit{shape}, \textit{ntrace})}
\hspace{1em}Calculate the index of a trace assuming a flat structure

nmrglue.fileio.fileiobase.index2trace_opp

\texttt{nmrglue.fileio.fileiobase.index2trace\_opp(\textit{shape}, \textit{index})}
\hspace{1em}Calculate trace number from shape and index of all indirect dimensions assuming a phase ordering opposite the time increments.

nmrglue.fileio.fileiobase.trace2index_opp

\texttt{nmrglue.fileio.fileiobase.trace2index\_opp(\textit{shape}, \textit{ntrace})}
\hspace{1em}Calculate the index of a trace assuming opposite phase/time increment ordering

nmrglue.fileio.fileiobase.index2trace_reg

\texttt{nmrglue.fileio.fileiobase.index2trace\_reg(\textit{shape}, \textit{index})}
\hspace{1em}Calculate trace number from shape and index of all indirect dimensions assuming the same phase and time ordering.

nmrglue.fileio.fileiobase.trace2index_reg

\texttt{nmrglue.fileio.fileiobase.trace2index\_reg(\textit{shape}, \textit{ntrace})}
\hspace{1em}Calculate the index of a trace assuming the same phase/time increment ordering

\textbf{Developer Classes}

These classes are typically not used directly by users. Developers may be interested in these classes.

\begin{tabular}{|l|l|}
\hline
\textit{unit\_conversion}(size, cplx, sw, obs, car) & Provides methods to convert between common NMR units \\
\textit{data\_nd}(order) & Base class for building objects which emulate ndarray objects without loading data into memory. \\
\hline
\end{tabular}
nmrglue.fileio.fileiobase.unit_conversion

class nmrglue.fileio.fileiobase.unit_conversion(size, cplx, sw, obs, car)
    Provides methods to convert between common NMR units

    Parameters size : int
        Number of points in dimension (RI).
    cplx : bool
        True if dimension is complex, False is real.
    sw : float
        Spectral width in Hz.
    obs : float
        Observation frequency in MHz.
    car : float
        Carrier frequency in Hz.

    __init__(size, cplx, sw, obs, car)
        create and set up a unit_conversion object

    Methods

    __init__(size, cplx, sw, obs, car) create and set up a unit_conversion object
    f(val[, unit]) Convert string or value/unit pair to float
    hz(val) Convert to Hz
    hz_limits() Return tuple of left and right edges in Hz
    hz_scale() Return array of Hz values
    i(val[, unit]) Convert string or value/unit pair to integer
    ms(val) Convert to milliseconds (ms)
    ms_limits() Return tuple of left and right edges in milliseconds
    ms_scale() Return array of seconds values
    percent(val) Convert to percent
    percent_limits() Return tuple of left and right edges in percent
    percent_scale() Return array of percent values
    ppm(val) Convert to ppm
    ppm_limits() Return tuple of left and right edges in ppm
    ppm_scale() Return array of ppm values
    sec(val) Convert to seconds
    sec_limits() Return tuple of left and right edges in seconds
    sec_scale() Return array of seconds values
    seconds(val) Convert to seconds
    unit(val, unit) Convert val points to unit
    us(val) Convert to microseconds (us)
    us_limits() Return tuple of left and right edges in milliseconds
    us_scale() Return array of seconds values
nmrglue.fileio.fileiobase.data_nd

class nmrglue.fileio.fileiobase.data_nd(order)

Base class for building objects which emulate ndarray objects without loading data into memory. These objects have the following properties:

- slicing operations return ndarray objects
- can iterate over with expected results
- transpose and swapaxes functions create a new data_nd object with the new axes ordering
- has ndim, shape, and dtype attributes.

Notes

Classes which are use this class as a base should define the following methods:

__init__ which must set up the object and defines at minimum:

- self._shape [tuple] Shape of the data on disk, the shape when order = (0, 1, 2, ..)
- self._order [tuple] Ordering of the axes
- self._dtype [dtype] Dtype of the emulated ndarray
- self._setdimandshape__ should be called if self.dim and self.shape are not set up __init__.

__getitem__ which takes a well formatted tuple of slices and returns a ndarray object with the selected data

__fcopy__(self, order) which created a copy of the object with the new order

__init__ (order)

Methods

<table>
<thead>
<tr>
<th><strong>init</strong> (order)</th>
</tr>
</thead>
<tbody>
<tr>
<td>swapaxes(axis1, axis2)</td>
</tr>
<tr>
<td>transpose(*axes)</td>
</tr>
</tbody>
</table>

3.1.5 nmrglue.jcampdx

Functions for reading 1D JCAMP-DX files.

This module is imported as nmrglue.jcampdx and can be called as such.

User Information

User Functions

<table>
<thead>
<tr>
<th>read(filename)</th>
<th>Read JCAMP-DX file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continued on next page</td>
<td></td>
</tr>
</tbody>
</table>
### nmrglue.fileio.jcampdx.read

`nmrglue.fileio.jcampdx.read(filename)`

Read JCAMP-DX file

**Parameters**
- `filename` : str
  File to read from.

**Returns**
- `dic` : dict
  Dictionary of parameters.
- `data` : ndarray
  Array of NMR data, or a list NMR data arrays in order [real, imaginary]

### nmrglue.fileio.jcampdx.guess_udic

`nmrglue.fileio.jcampdx.guess_udic(dic, data)`

Guess parameters of universal dictionary from dic, data pair.

**Parameters**
- `dic` : dict
  Dictionary of JCAMP-DX parameters.
- `data` : ndarray
  Array of NMR data.

**Returns**
- `udic` : dict
  Universal dictionary of spectral parameters.

### Developer Information

**JCAMP-DX file format information**

The format reference publications are available at: [http://www.jcamp-dx.org/protocols.html]

Notes:
- Writing NMR data in JCAMP-DX format is not currently supported
- Multi-dimensional JCAMP-files are not currently supported
  (see [http://www.jcamp-dx.org/ndnmr-index.html#2dnmr%20testfiles])

### 3.1.6 nmrglue.nmrml

Functions for reading nmrML files.

This module is imported as nmrglue.nmrml and can be called as such.
**User Information**

**User Functions**

```python
read(filename[, data_dtype]) Read a nmrML file.
```

**nmrglue.fileio.nmrml.read**

nmrglue.fileio.nmrml.read(filename, data_dtype=None)
Read a nmrML file.

**Parameters**
- `filename` : str
  - Name of nmrML file to read.
- `data_dtype` : str, optional
  - NumPy data type of the data. None, the default, will determine this data type from the information in the file. Occasionally this information is incorrect and this argument can be used to explicitly supply this information.

**Returns**
- `dic` : dict
  - Dictionary of spectra parameters.
- `data` : ndarray, 1D
  - Array of spectral data.

**Developer Information**

**Developer Functions**

These functions are typically not used directly by users.

```python
_get_nmrml_data(fid_dict, data_dtype) Return a NumPy array for spectral data in a nmrML data file.
```

**nmrglue.fileio.nmrml._get_nmrml_data**

nmrglue.fileio.nmrml._get_nmrml_data(fid_dict, data_dtype)
Return a NumPy array for spectral data in a nmrML data file.

**3.1.7 nmrglue.pipe**

Functions for reading and writing NMRPipe files and table (.tab) files

This modules is imported as nmrglue.pipe and can be called as such.

**User Information**

**User Functions**

---

Chapter 3. Reference Guide
nmrglue Documentation, Release 0.7-dev

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nmrglue.fileio.pipe.read</strong></td>
<td>Read a NMRPipe file.</td>
</tr>
<tr>
<td><strong>nmrglue.fileio.pipe.write</strong></td>
<td>Write a NMRPipe file to disk.</td>
</tr>
<tr>
<td><strong>read</strong> (filename)</td>
<td>Read a NMRPipe file.</td>
</tr>
<tr>
<td><strong>write</strong> (filename, dic, data[, overwrite])</td>
<td>Write a NMRPipe file to disk.</td>
</tr>
<tr>
<td><strong>read_lowmem</strong> (filename)</td>
<td>Read a NMRPipe file with minimal memory usage.</td>
</tr>
<tr>
<td><strong>write_lowmem</strong> (filename, dic, data[, overwrite])</td>
<td>Write a NMRPipe file to disk using minimal memory (trace by trace).</td>
</tr>
<tr>
<td><strong>read_table</strong> (filename)</td>
<td>Read a NMRPipe database table (.tab) file.</td>
</tr>
<tr>
<td><strong>write_table</strong> (filename, pcomments, pformats, rec)</td>
<td>Write a NMRPipe database table (.tab) file.</td>
</tr>
<tr>
<td><strong>make_uc</strong> (dic, data[, dim])</td>
<td>Create a unit conversion object.</td>
</tr>
<tr>
<td><strong>guess_udic</strong> (dic, data)</td>
<td>Guess parameters of universal dictionary from dic, data pair.</td>
</tr>
<tr>
<td><strong>create_dic</strong> (udic[, datetimeobj])</td>
<td>Crate a NMRPipe parameter dictionary from universal dictionary</td>
</tr>
<tr>
<td><strong>datetime2dic</strong> (dt, dic)</td>
<td>Add datetime object to a NMRPipe dictionary</td>
</tr>
<tr>
<td><strong>dic2datetime</strong> (dic)</td>
<td>Create a datetime object from a NMRPipe dictionary</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.pipe.read**

**nmrglue.fileio.pipe.read** (filename)

Read a NMRPipe file.

For standard multi-file 3D/4D NMRPipe data sets, filename should be a filemask (for example “/ft/test%03d.ft3”) with a “%” formatter. If only one file of a 3D/4D data set is provided only that 2D slice of the data is read (for example “/ft/test001.ft3” results in a 2D data set being read).

NMRPipe data streams stored as files (one file 3D/4D data sets made using xyz2pipe) can be read by providing the file name of the stream. The entire data set is read into memory.

**Parameters**

- **filename** : str
  - Filename or filemask of NMRPipe file(s) to read.

**Returns**

- **dic** : dict
  - Dictionary of NMRPipe parameters.
  - **data** : ndarray
    - Array of NMR data.

**See also**:

- **read_lowmem** NMRPipe file reading with minimal memory usage.
- **write** Write a NMRPipe data to file(s).

**nmrglue.fileio.pipe.write**

**nmrglue.fileio.pipe.write** (filename, dic, data, overwrite=False)

Write a NMRPipe file to disk.

**Parameters**

- **filename** : str
  - Filename of NMRPipe to write to. See Notes.
- **dic** : dict
  - Dictionary of NMRPipe parameters.
data : array_like
    Array of NMR data.

overwrite : bool, optional.
    Set True to overwrite files, False will raise a Warning if file exists.

See also:

write_lowmem Write NMRPipe files using minimal amounts of memory.
read Read NMRPipe files.

Notes

For 3D data if filename has no ‘%’ formatter then the data is written as a 3D NMRPipe data stream. When the ‘%’ formatter is provided the data is written out as a standard NMRPipe multi-file 3D.

For 4D data, filename can have one, two or no ‘%’ formatters resulting in a single index file (test%03d.ft), two index file(test%02d%03d.ft), or one file data stream (test.ft4).

dic["FDPIPEFLAG"] is not changed or checked when writing, please check that this value is 0.0 for standard non-data stream files, and 1.0 for data stream files or an file may be written with an incorrect header.

Set overwrite to True to overwrite files that exist.

nmrglue.fileio.pipe.read_lowmem

nmrglue.fileio.pipe.read_lowmem(filename)
Read a NMRPipe file with minimal memory usage.

See read() for Parameters and information.

Returns dic : dict
    Dictionary of NMRPipe parameters.

data : array_like
    Low memory object which can access NMR data on demand.

See also:

read Read NMRPipe files.
write_lowmem Write NMRPipe files using minimal amounts of memory.

nmrglue.fileio.pipe.write_lowmem

nmrglue.fileio.pipe.write_lowmem(filename, dic, data, overwrite=False)
Write a NMRPipe file to disk using minimal memory (trace by trace).

Parameters filename : str
    Filename of NMRPipe to write to. See write() for details.

dic : dict
    Dictionary of NMRPipe parameters.
data : array_like
    Array of NMR data.
overwrite : bool, optional.
    Set True to overwrite files, False will raise a Warning if file exists.

See also:

write Write a NMRPipe file to disk.
read_lowmem Read a NMRPipe file using minimal memory.

nmrglue.fileio.pipe.read_table

nmrglue.fileio.pipe.read_table(filename)
    Read a NMRPipe database table (.tab) file.

    Parameters filename : str
        Filename of NMRPipe table file to read.

    Returns pcomments : list
        List of NMRPipe comment lines
    pformat : list :
        List of NMRPipe table column format strings.
    rec : recarray
        Records array with named fields.

See also:

write_table Write a NMRPipe table file.

nmrglue.fileio.pipe.write_table

nmrglue.fileio.pipe.write_table(filename, pcomments, pformats, rec, overwrite=False)
    Write a NMRPipe database table (.tab) file.

    Parameters filename : str
        Filename of file to write to.
    pcomments : list :
        List of NMRPipe comment lines.
    pformats : :
        List of NMRPipe table column formats strings.
    rec : recarray
        Records array of table.
    overwrite : bool, optional :
        True to overwrite file if it exists, False will raise a Warning if the file exists.
See also:

**read_table**  
Read a NMRPipe table file.

### nmrglue.fileio.pipe.make_uc

**nmrglue.fileio.pipe.make_uc**(dic, data, dim=-1)

Create a unit conversion object

Parameters:
- **dic**: dict  
  Dictionary of NMRPipe parameters.
- **data**: ndarray  
  Array of NMR data.
- **dim**: int, optional  
  Dimension number to create unit conversion object for. Default is for last (direct) dimension.

Returns:
- **uc**: unit conversion object  
  Unit conversion object for given dimension.

### nmrglue.fileio.pipe.guess_udic

**nmrglue.fileio.pipe.guess_udic**(dic, data)

Guess parameters of universal dictionary from dic, data pair.

Parameters:
- **dic**: dict  
  Dictionary of NMRPipe parameters.
- **data**: ndarray  
  Array of NMR data.

Returns:
- **udic**: dict  
  Universal dictionary of spectral parameters.

### nmrglue.fileio.pipe.create_dic

**nmrglue.fileio.pipe.create_dic**(udic, datetimeobj=datetime.datetime(2017, 11, 29, 18, 21, 13, 29984))

Create a NMRPipe parameter dictionary from universal dictionary

This function does not update the dictionary keys that are unknown such as MIN/MAX, apodization and processing parameters, and sizes in none-current domain. Also rounding of parameter is different than NMRPipe.

Parameters:
- **udic**: dict  
  Universal dictionary of spectral parameters.
- **datetimeobj**: datetime object, optional  
  Datetime to record in NMRPipe dictionary

Returns:
- **dic**: dict
Dictionary NMRPipe parameters.

nmrglue.fileio.pipe.datetime2dic

nmrglue.fileio.pipe.datetime2dic(dt, dic)
Add datetime object to a NMRPipe dictionary

nmrglue.fileio.pipe.dic2datetime

nmrglue.fileio.pipe.dic2datetime(dic)
Create a datetime object from a NMRPipe dictionary

User Classes

iter3D(filemask[, in_lead, out_lead]) Object which allows for graceful iteration over 3D NMRPipe files.

nmrglue.fileio.pipe.iter3D

class nmrglue.fileio.pipe.iter3D (filemask, in_lead='x', out_lead='DEFAULT')
Object which allows for graceful iteration over 3D NMRPipe files.

iter3D.iter() returns a (dic,plane) tuple which can be written using the x.writeplane function.

When processing 3D files with iter3D object(s) the following dictionary parameters may not have the same values as NMRPipe processing scripts return:

• FDSLICECOUNT
• FDMAX,FDDISMAX,FDMIN,FDDISPMIN when FDSCALEFLAG == 0

Example:

```python
#3D data processing
xiter = iter3D("data/test%03d.fid","x","x")
for dic,YXplane in xiter:
    # process X and Y axis
    xiter.write("ft/test%03d.ft2",YXplane,dic)
ziter = iter3D("ft/test%03d.ft2","z","z")
for dic,XZplane in ziter:
    # process Z axis
    ziter.write("ft/test%03d.ft3",XZplane,dic)
```

__init__ (filemask, in_lead='x', out_lead='DEFAULT')
Create a iter3D object

Parameters

- **filemask**: str
  String file with single formatter (%) which represents which indicates which NMRPipe file to read.
- **in_lead**: ('x', 'y', 'z'), optional
  Axis name of last (1st) axis in outputted 2D
**out_lead**: (‘x’, ‘y’, ‘z’, ‘DEFAULT’), optional

Axis name of axis to be written, typically this is the same as in_load, which is the used if ‘DEFAULT’ is given.

### Notes

<table>
<thead>
<tr>
<th>In-lead</th>
<th>Iterated Planes</th>
</tr>
</thead>
<tbody>
<tr>
<td>“x”</td>
<td>(‘y’, ‘x’)</td>
</tr>
<tr>
<td>“y”</td>
<td>(‘x’, ‘y’)</td>
</tr>
<tr>
<td>“z”</td>
<td>(‘x’, ‘z’)</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Create a iter3D object</td>
</tr>
<tr>
<td>next()</td>
<td>Return the next dic, plane or raise StopIteration</td>
</tr>
<tr>
<td>reinitialize()</td>
<td>Restart iterator at first dic,plane.</td>
</tr>
<tr>
<td>write(filemask, plane, dic)</td>
<td>Write out current plane.</td>
</tr>
</tbody>
</table>

### Developer Information

from __future__ import print_function, division

### Developer Functions

These functions are typically not used directly by users. Developers who want fine control over NMRPipe files may be interested in these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_data(data)</td>
<td>Create a NMRPipe data array (recast into float32 or complex64)</td>
</tr>
<tr>
<td>add_axis_to_dic(dic, adic, n)</td>
<td>Add an axis dictionary (adic) to a NMRPipe dictionary (dic) as axis n.</td>
</tr>
<tr>
<td>create_empty_dic()</td>
<td>Creates a NMRPipe dictionary with default values</td>
</tr>
<tr>
<td>read_1D(filename)</td>
<td>Read a 1D NMRPipe file.</td>
</tr>
<tr>
<td>read_2D(filename)</td>
<td>Read a 2D NMRPipe file or NMRPipe data stream.</td>
</tr>
<tr>
<td>read_lowmem_2D(filename)</td>
<td>Read a 2D NMRPipe file or NMRPipe data stream using minimal memory.</td>
</tr>
<tr>
<td>read_stream(filename)</td>
<td>Read a NMRPipe data stream (one file 3D or 4D files).</td>
</tr>
<tr>
<td>read_lowmem_stream(filename)</td>
<td>Read a NMRPipe data stream using minimal memory.</td>
</tr>
<tr>
<td>read_3D(filename)</td>
<td>Read a 3D NMRPipe file.</td>
</tr>
<tr>
<td>read_lowmem_3D(filename)</td>
<td>Read a 3D NMRPipe file using minimal memory.</td>
</tr>
<tr>
<td>read_4D(filename)</td>
<td>Read a 4D NMRPipe file.</td>
</tr>
<tr>
<td>read_lowmem_4D(filename)</td>
<td>Read a NMRPipe file using minimal memory.</td>
</tr>
<tr>
<td>write_single(filename, dic, data[, overwrite])</td>
<td>Write data to a single NMRPipe file from memory.</td>
</tr>
<tr>
<td>write_3D(filename, dic, data[, overwrite])</td>
<td>Write a standard multi-file 3D NMRPipe file</td>
</tr>
<tr>
<td>write_4D(filename, dic, data[, overwrite])</td>
<td>Write a one or two index 4D NMRPipe file.</td>
</tr>
<tr>
<td>write_lowmem_2D(filename, dic, data[, overwrite])</td>
<td>Write a 2D NMRPipe file using minimal memory (trace by trace)</td>
</tr>
</tbody>
</table>
Table 3.15 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>write_lowmem_3D</code></td>
<td>Write a standard multi-file 3D NMRPipe file using minimal memory.</td>
</tr>
<tr>
<td><code>write_lowmem_3Ds</code></td>
<td>Write 3D NMRPipe data stream file using minimal memory (trace by trace)</td>
</tr>
<tr>
<td><code>write_lowmem_4D</code></td>
<td>Write a multi-file (single or double index) 4D NMRPipe file using minimal memory.</td>
</tr>
<tr>
<td><code>write_lowmem_4Ds</code></td>
<td>Write 4D NMRPipe data stream file using minimal memory (trace by trace)</td>
</tr>
<tr>
<td><code>put_fdata</code></td>
<td>Put NMR data, fdata, to a NMRPipe file described by file object fh.</td>
</tr>
<tr>
<td><code>put_trace</code></td>
<td>Put a trace (real or complex) to NMRPipe file described by file object fh.</td>
</tr>
<tr>
<td><code>put_data</code></td>
<td>Put fdata and data to 2D NMRPipe.</td>
</tr>
<tr>
<td><code>write_slice_3D</code></td>
<td>Write a slice of a 3D data array to file.</td>
</tr>
<tr>
<td><code>pack_complex</code></td>
<td>Pack interleaved real, imag array into complex array.</td>
</tr>
<tr>
<td><code>transpose_3D</code></td>
<td>Transpose pipe_3d object and dictionary</td>
</tr>
<tr>
<td><code>find_shape</code></td>
<td>Find the shape (tuple) of data in a NMRPipe file from parameters.</td>
</tr>
<tr>
<td><code>reshape_data</code></td>
<td>Reshape data or return 1D data after warning.</td>
</tr>
<tr>
<td><code>unshape_data</code></td>
<td>Return 1D version of data.</td>
</tr>
<tr>
<td><code>unappend_data</code></td>
<td>Return complex data with last axis (-1) unappended.</td>
</tr>
<tr>
<td><code>append_data</code></td>
<td>Return data with last axis (-1) appended.</td>
</tr>
<tr>
<td><code>fdata2dic</code></td>
<td>Convert a fdata array to fdata dictionary.</td>
</tr>
<tr>
<td><code>dic2fdata</code></td>
<td>Converts a NMRPipe dictionary into an array.</td>
</tr>
<tr>
<td><code>get_fdata</code></td>
<td>Get an array of length 512-bytes holding NMRPipe header.</td>
</tr>
<tr>
<td><code>get_data</code></td>
<td>Get array of data</td>
</tr>
<tr>
<td><code>get_fdata_data</code></td>
<td>Get fdata and data array, return (fdata, data)</td>
</tr>
<tr>
<td><code>get_trace</code></td>
<td>Get a single trace from a NMRPipe file</td>
</tr>
</tbody>
</table>

### nmrglue.fileio.pipe.create_data

_create_data(data)_

Create a NMRPipe data array (recast into float32 or complex64)

### nmrglue.fileio.pipe.add_axis_to_dic

_add_axis_to_dic(dic, adic, n)_

Add an axis dictionary (adic) to a NMRPipe dictionary (dic) as axis n.

### nmrglue.fileio.pipe.create_empty_dic

_create_empty_dic()_

Creates a NMRPipe dictionary with default values

### nmrglue.fileio.pipe.read_1D

_read_1D(filename)_

Read a 1D NMRPipe file.
See `read()` for documentation.

**nmrglue.fileio.pipe.read_2D**

`nmrglue.fileio.pipe.read_2D(filename)`  
Read a 2D NMRPipe file or NMRPipe data stream.  
See `read()` for documentation.

**nmrglue.fileio.pipe.read_lowmem_2D**

`nmrglue.fileio.pipe.read_lowmem_2D(filename)`  
Read a 2D NMRPipe file or NMRPipe data stream using minimal memory.  
See `read_lowmem()` for documentation.

**nmrglue.fileio.pipe.read_stream**

`nmrglue.fileio.pipe.read_stream(filename)`  
Read a NMRPipe data stream (one file 3D or 4D files).  
See `read()` for documentation.

**nmrglue.fileio.pipe.read_lowmem_stream**

`nmrglue.fileio.pipe.read_lowmem_stream(filename)`  
Read a NMRPipe data stream using minimal memory.  
See `read_lowmem()` for documentation.

**nmrglue.fileio.pipe.read_3D**

`nmrglue.fileio.pipe.read_3D(filemask)`  
Read a 3D NMRPipe file.  
See `read()` for documentation.

**nmrglue.fileio.pipe.read_lowmem_3D**

`nmrglue.fileio.pipe.read_lowmem_3D(filemask)`  
Read a 3D NMRPipe file using minimal memory.  
See `read_lowmem()` for documentation.

**nmrglue.fileio.pipe.read_4D**

`nmrglue.fileio.pipe.read_4D(filemask)`  
Read a 3D NMRPipe file.  
See `read()` for documentation.
Notes

This function should not be used to read NMRPipe data streams stored in a single file (one file 3D/4D data sets made using xyz2pipe), read_2D() should be used.

nmrglue.fileio.pipe.read_lowmem_4D

nmrglue.fileio.pipe.read_lowmem_4D (filemask)
Read a NMRPipe file using minimal memory.
See read_lowmem() for documentation

Notes

This function should not be used to read NMRPipe data streams stored in a single file (one file 3D/4D data sets made using xyz2pipe), read_lowmem_2D() should be used.

nmrglue.fileio.pipe.write_single

nmrglue.fileio.pipe.write_single (filename, dic, data, overwrite=False)
Write data to a single NMRPipe file from memory.
Write 1D and 2D files completely as well as NMRPipe data streams. 2D planes of 3D and 4D files should be written with this function.
See write() for documentation.

nmrglue.fileio.pipe.write_3D

nmrglue.fileio.pipe.write_3D (filemask, dic, data, overwrite=False)
Write a standard multi-file 3D NMRPipe file
See write() for documentation.

nmrglue.fileio.pipe.write_4D

nmrglue.fileio.pipe.write_4D (filemask, dic, data, overwrite=False)
Write a one or two index 4D NMRPipe file.
See write() for documentation.

nmrglue.fileio.pipe.write_lowmem_2D

nmrglue.fileio.pipe.write_lowmem_2D (filename, dic, data, overwrite=False)
Write a 2D NMRPipe file using minimal memory (trace by trace)
See write_lowmem() for documentation.
nmrglue.fileio.pipe.write_lowmem_3D

nmrglue.fileio.pipe.write_lowmem_3D (filename, dic, data, overwrite=False)
Write a standard multi-file 3D NMRPipe file using minimal memory.
See write_lowmem() for documentation.

Notes
MIN/MAX parameters are not updated in the NMRPipe headers.

nmrglue.fileio.pipe.write_lowmem_3Ds

nmrglue.fileio.pipe.write_lowmem_3Ds (filename, dic, data, overwrite=False)
Write 3D NMRPipe data stream file using minimal memory (trace by trace)
See write_lowmem() for documentation.

nmrglue.fileio.pipe.write_lowmem_4D

nmrglue.fileio.pipe.write_lowmem_4D (filename, dic, data, overwrite=False)
Write a multi-file (single or double index) 4D NMRPipe file using minimal memory.
See write_lowmem() for documentation.

Notes
MIN/MAX parameters are not updated in the NMRPipe headers.

nmrglue.fileio.pipe.write_lowmem_4Ds

nmrglue.fileio.pipe.write_lowmem_4Ds (filename, dic, data, overwrite=False)
Write 4D NMRPipe data stream file using minimal memory (trace by trace)
See write_lowmem() for documentation.

nmrglue.fileio.pipe.put_fdata

nmrglue.fileio.pipe.put_fdata (fh, fdata)
Put NMR data, fdata, to a NMRPipe file described by file object fh.

nmrglue.fileio.pipe.put_trace

nmrglue.fileio.pipe.put_trace (fh, trace)
Put a trace (real or complex) to NMRPipe file described by file object fh.
nmrglue.fileio.pipe.put_data

nmrglue.fileio.pipe.put_data(filename, fdata, data, overwrite=False)
Put fdata and data to 2D NMRPipe.

nmrglue.fileio.pipe.write_slice_3D

nmrglue.fileio.pipe.write_slice_3D(filemask, dic, data, shape, slices)
Write a slice of a 3D data array to file.
Opens (or if necessary creates) a 2D NMRPipe file(s) to write data, where the total 3D file size is given by shape.

Parameters

filemask : str
String of NMRPipe file with single formatting operator (%).
dic : dict
Dictionary of NMRPipe parameters.
data : ndarray
3D array of NMR data.
shape : tuple
Tuple of 3 integers indicating the overall matrix shape.
slices : slices
Slice objects which specify the location of the to be written data.

See also:

iter3D Users should use this object, not this function.

Notes

This function memmaps 2D NMRPipe files for speed. It only writes dictionaries to file when created, leaving them unmodified if the file exists. Only error checking is that data is 3D.

nmrglue.fileio.pipe.pack_complex

nmrglue.fileio.pipe.pack_complex(data)
Pack interleaved real,imag array into complex array.

nmrglue.fileio.pipe.transpose_3D

nmrglue.fileio.pipe.transpose_3D(dic, data, axes=(2, 1, 0))
Transpose pipe_3D object and dictionary
**nmrglue.fileio.pipe.find_shape**

```python
nmrglue.fileio.pipe.find_shape(dic)
```
Find the shape (tuple) of data in a NMRPipe file from parameters.

- 1-tuple is returned for 1D data, 2-tuple for 2D and non-stream 3D/4D data, 3-tuple or 4-tuple for stream 3D/4D data.

The last dimension of the tuple is length of the data in the file, the actual length of the data matrix may be half of this if the data is complex.

**nmrglue.fileio.pipe.reshape_data**

```python
nmrglue.fileio.pipe.reshape_data(data, shape)
```
Reshape data or return 1D data after warning.

**nmrglue.fileio.pipe.unshape_data**

```python
nmrglue.fileio.pipe.unshape_data(data)
```
Return 1D version of data.

**nmrglue.fileio.pipe.unappend_data**

```python
nmrglue.fileio.pipe.unappend_data(data)
```
Return complex data with last axis (-1) unappended.

Data should have imaginary data vector appended to real data vector.

**nmrglue.fileio.pipe.append_data**

```python
nmrglue.fileio.pipe.append_data(data)
```
Return data with last axis (-1) appended.

Data should be complex.

**nmrglue.fileio.pipe.fdata2dic**

```python
nmrglue.fileio.pipe.fdata2dic(fdata)
```
Convert a fdata array to fdata dictionary.

Converts the raw 512x4-byte NMRPipe header into a python dictionary with keys as given in fdatap.h.

**nmrglue.fileio.pipe.dic2fdata**

```python
nmrglue.fileio.pipe.dic2fdata(dic)
```
Converts a NMRPipe dictionary into an array.
nmrglue.fileio.pipe.get_fdata

`nmrglue.fileio.pipe.get_fdata(filename)`
Get an array of length 512-bytes holding NMRPipe header.

nmrglue.fileio.pipe.get_data

`nmrglue.fileio.pipe.get_data(filename)`
Get array of data

nmrglue.fileio.pipe.get_fdata_data

`nmrglue.fileio.pipe.get_fdata_data(filename)`
Get fdata and data array, return (fdata, data)

nmrglue.fileio.pipe.get_trace

`nmrglue.fileio.pipe.get_trace(fhandle, ntrace, pts, bswap, cplex)`
Get a single trace from a NMRPipe file

**Parameters**

- **fhandle** : file object
  File object of open NMRPipe file.
- **ntrace** : int
  Trace numbers (starting from 0).
- **pts** : int
  Number of points in trace, R1I.
- **bswap** : bool
  True to perform byteswap on trace.
- **cplex** : bool
  True to unappend imaginary data.

**Developer Classes**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pipe_2d</strong> (filename[, order])</td>
<td>Emulate ndarray objects without loading data into memory for low memory reading of 2D NMRPipe files.</td>
</tr>
<tr>
<td><strong>pipe_3d</strong> (filemask[, order, fcheck])</td>
<td>Emulate ndarray objects without loading data into memory for low memory reading of 3D NMRPipe files (multiple file data sets).</td>
</tr>
<tr>
<td><strong>pipestream_3d</strong> (filename[, order])</td>
<td>Emulate ndarray objects without loading data into memory for low memory reading of 3D NMRPipe data stream files (one file data sets).</td>
</tr>
<tr>
<td><strong>pipe_4d</strong> (filemask[, order, fcheck])</td>
<td>Emulate ndarray objects without loading data into memory for low memory reading of single/two index 4D NMRPipe data files.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.16 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipestream_4d(filename[, order])</td>
<td>Emulate a ndarray objects without loading data into memory for low memory reading of 4D NMRPipe data streams (one file 4D data sets).</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.pipe.pipe_2d**

class nmrglue.fileio.pipe.pipe_2d(filename, order=(0, 1))

Emulate a ndarray objects without loading data into memory for low memory reading of 2D NMRPipe files.

- slicing operations return ndarray objects.
- can iterate over with expected results.
- transpose and swapaxes methods create a new objects with correct axes ordering.
- has ndim, shape, and dtype attributes.

**Parameters**

- **filename** : str
  Filename of 2D NMRPipe file.
- **order** : tuple
  Ordering of axes against file.

**Methods**

- __init__(filename, order=(0, 1))
  Create and set up object

- swapaxes(axis1, axis2)
  Return object with $axis1$ and $axis2$ interchanged.

- transpose(*axes)
  Return object with axes transposed.

**nmrglue.fileio.pipe.pipe_3d**

class nmrglue.fileio.pipe.pipe_3d(filemask, order=(0, 1, 2), fcheck=False)

Emulate a ndarray objects without loading data into memory for low memory reading of 3D NMRPipe files (multiple file data sets).

- slicing operations return ndarray objects.
- can iterate over with expected results.
- transpose and swapaxes methods create a new objects with correct axes ordering.
- has ndim, shape, and dtype attributes.

**Parameters**

- **filemask** : str
  Filename of 3D NMRPipe file. Should contain one formatter ‘%’ operator.
- **order** : tuple
  Ordering of axes against file.
- **fcheck** : bool, optional.
True to perform a basic check to see if all files expected for the data set exist. Raises a
IOError if files are missing. Default is False.

```python
__init__(filemask, order=(0, 1, 2), fcheck=False)
    Create and set up object, check that files exist if fcheck is True
```

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Create and set up object, check that files exist if fcheck is True</td>
</tr>
<tr>
<td>swapaxes</td>
<td>Return object with axis1 and axis2 interchanged.</td>
</tr>
<tr>
<td>transpose</td>
<td>Return object with axes transposed.</td>
</tr>
</tbody>
</table>

### nmrglue.fileio.pipe.pipestream_3d

**class** nmrglue.fileio.pipe.pipestream_3d

Emulate a ndarray objects without loading data into memory for low memory reading of 3D NMRPipe data stream files (one file data sets).

- slicing operations return ndarray objects.
- can iterate over with expected results.
- transpose and swapaxes methods create a new objects with correct axes ordering.
- has ndim, shape, and dtype attributes.

**Parameters**

- `filename` : str
  Filename of 3D NMRPipe stream file.
- `order` : tuple
  Ordering of axes against file.

```python
__init__(filename, order=(0, 1, 2))
    Create and set up object
```

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Create and set up object</td>
</tr>
<tr>
<td>swapaxes</td>
<td>Return object with axis1 and axis2 interchanged.</td>
</tr>
<tr>
<td>transpose</td>
<td>Return object with axes transposed.</td>
</tr>
</tbody>
</table>

### nmrglue.fileio.pipe.pipe_4d

**class** nmrglue.fileio.pipe.pipe_4d

Emulate a ndarray objects without loading data into memory for low memory reading of single/two index 4D NMRPipe data files.

- slicing operations return ndarray objects.
- can iterate over with expected results.

**Parameters**

- `filemask` : tuple
  Mask of files expected for data set.
- `order` : tuple
  Ordering of axes against file.
- `fcheck` : bool, default=False
  True to perform a basic check to see if all files expected for the data set exist. Raises a
  IOError if files are missing. Default is False.

```python
__init__(filemask, order=(0, 1, 2), fcheck=False)
    Create and set up object, check that files exist if fcheck is True
```

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>Create and set up object, check that files exist if fcheck is True</td>
</tr>
<tr>
<td>swapaxes</td>
<td>Return object with axis1 and axis2 interchanged.</td>
</tr>
<tr>
<td>transpose</td>
<td>Return object with axes transposed.</td>
</tr>
</tbody>
</table>
• transpose and swapaxes methods create a new objects with correct axes ordering.
• has ndim, shape, and dtype attributes.

Parameters

**filemask** : str
Filename of 4D NMRPipe file with one or two formatter (%) operators.

**order** : tuple
Ordering of axes against file.

**fcheck** : bool, optional.
True to perform a basic check to see if all files expected for the data set exist. Raises a IOError if files are missing. Default is False.

```python
__init__(filemask, order=(0, 1, 2, 3), fcheck=False)
Create and set up object, check that files exist if fcheck is True
```

Methods

```python
__init__(filemask[, order, fcheck])  # Create and set up object, check that files exist if fcheck is True
swapaxes(axis1, axis2)  # Return object with axis1 and axis2 interchanged.
transpose(*axes)  # Return object with axes transposed.
```

**nmrglue.fileio.pipe.pipestream_4d**

class **nmrglue.fileio.pipe.pipestream_4d** (filename, order=(0, 1, 2, 3))
Emulate a ndarray objects without loading data into memory for low memory reading of 4D NMRPipe data streams (one file 4D data sets).

• slicing operations return ndarray objects.
• can iterate over with expected results.
• transpose and swapaxes methods create a new objects with correct axes ordering.
• has ndim, shape, and dtype attributes.

Parameters

**filename** : str
Filename of 4D NMRPipe stream file.

**order** : tuple
Ordering of axes against file.

```python
__init__(filename, order=(0, 1, 2, 3))
Create and set up object
```

Methods

```python
__init__(filename[, order])  # Create and set up object
```

Continued on next page
Table 3.21 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>swapaxes(axis1, axis2)</code></td>
<td>Return object with <code>axis1</code> and <code>axis2</code> interchanged.</td>
</tr>
<tr>
<td><code>transpose(*axes)</code></td>
<td>Return object with axes transposed.</td>
</tr>
</tbody>
</table>

### 3.1.8 nmrglue.rnmrtk

Functions for reading and writing Rowland NMR Toolkit (RNMRTK) files

This modules is imported as `nmrglue.rnmrtk` and can be called as such.

#### User Information

#### User Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read(filename[, par_file])</code></td>
<td>Read RNMRTK files.</td>
</tr>
<tr>
<td><code>write(filename, dic, data[, par_file, overwrite])</code></td>
<td>Write RNMRTK files.</td>
</tr>
<tr>
<td><code>write_lowmem(filename, dic, data[, ...])</code></td>
<td>Write RNMRTK files using minimal amounts of memory (trace by trace).</td>
</tr>
<tr>
<td><code>read_lowmem(filename[, par_file])</code></td>
<td>Read RNMRTK files with minimal memory usage.</td>
</tr>
<tr>
<td><code>read_sec(filename, dtype, shape, cplex)</code></td>
<td>Read a RNMRTK parameter .par file.</td>
</tr>
<tr>
<td><code>write_sec(filename, data[, dtype, overwrite])</code></td>
<td>Write a RNMRTK .sec file.</td>
</tr>
<tr>
<td><code>read_par(filename)</code></td>
<td>Parse a RNMRTK parameter (.par) file.</td>
</tr>
<tr>
<td><code>write_par(par_file, dic, overwrite)</code></td>
<td>Write a RNMRTK parameter file (.par).</td>
</tr>
<tr>
<td><code>make_uc(dic, data[, dim])</code></td>
<td>Create a unit conversion object.</td>
</tr>
<tr>
<td><code>guess_udic(dic, data)</code></td>
<td>Guess parameters of a universal dictionary from a dic, data pair.</td>
</tr>
<tr>
<td><code>create_dic(udic[, dim_order])</code></td>
<td>Create a RNMRTK dictionary from a universal dictionary.</td>
</tr>
</tbody>
</table>

### nmrglue.fileio.rnmrtk.read

`nmrglue.fileio.rnmrtk.read(filename[, par_file=None])`

Read RNMRTK files.

**Parameters**

- `filename`: str
  - Filename of RNMRTK file to read (.sec).
- `par_file`: str or None, optional
  - Filename of RNMRTK parameter file. If None (default) a the last four characters of `file` are changed to .par.

**Returns**

- `dic`: dic
  - Dictionary of RNMRTK parameters.
- `data`: ndarray
  - Array of NMR data.

**See also:**

- `read_lowmem` Read RNMRTK files with minimal memory usage.
- `write` Write RNMRTK files.
Notes

The dictionary parameters are ordered opposite the data layout, that is to say the the FIRST parameter in each list corresponds to the LAST axis in the data array.

nmrglue.fileio.rnmrtk.write

nmrglue.fileio.rnmrtk.write(filename, dic, data, par_file=None, overwrite=False)

Write RNMRTK files.

Parameters filename : str
  Filename of RNMRTK file to write to (.sec).

dic : dict
  Dictionary of RNMRTK parameters.

data : ndarray
  Array of NMR data.

par_file : str or None, optional
  Filename of RNMRTK parameter file. If None (default) a the last four characters of file are changed to .par.

overwrite : bool, optional
  True to overwrite existing files. False will raises a Warning if the file exists.

See also:

write_lowmem Write RNMRTK files using minimal amounts of memory.

read Read RNMRTK files.

nmrglue.fileio.rnmrtk.write_lowmem

nmrglue.fileio.rnmrtk.write_lowmem(filename, dic, data, par_file=None, overwrite=False)

Write RNMRTK files using minimal amounts of memory (trace by trace).

Parameters filename : str
  Filename of RNMRTK file to write to (.sec).

dic : dict
  Dictionary of RNMRTK parameters.

data : array_like
  Array of NMR data.

par_file : str or None, optional
  Filename of RNMRTK parameter file. If None (default) a the last four characters of file are changed to .par.

overwrite : bool, optional
  True to overwrite existing files. False will raises a Warning if the file exists.
See also:

**write**  Write RNMRTK files using minimal amounts of memory.

**read_lowmem**  Read RNMRTK files using minimal amounts of memory.

### nmrglue.fileio.rnmrtk.read_lowmem

nmrglue.fileio.rnmrtk.read_lowmem(filename, par_file=None)

Read RNMRTK files with minimal memory usage

**Parameters**

- **filename**: str
  
  Filename of RNMRTK file to read (.sec).

- **par_file**: str or None, optional
  
  Filename of RNMRTK parameter file. If None (default) a the last four characters of file are changed to .par.

**Returns**

- **dic**: dic
  
  Dictionary of RNMRTK parameters.

- **data**: array_like
  
  Low memory object which can access NMR data on demand.

See also:

**read**  Read RNMRTK files.

**write**  Write RNMRTK files.

**Notes**

The dictionary parameters are ordered opposite the data layout, that is to say the the FIRST parameter in each list corresponds to the LAST axis in the data array.

### nmrglue.fileio.rnmrtk.read_sec

nmrglue.fileio.rnmrtk.read_sec(filename, dtype, shape, cplex)

Read a RNMRTK parameter .par file.

**Parameters**

- **filename**: str
  
  Filename of RNMRTK (.sec) file to read.

- **dtype**: dtype
  
  Type of data in file, typically ‘float32’.

- **shape**: tuple
  
  Shape of data.

- **cplex**: bool
  
  True if the last (fast) dimension is complex. False is real only.

**Returns**

- **data**: ndarray
  
  3.1. fileio modules 59
Array of NMR data.

**nmrglue.fileio.rnmrtk.write_sec**

nmrglue.fileio.rnmrtk.write_sec(filename, data, dtype=’f4’, overwrite=False)

Write a RNMRTK .sec file.

- **Parameters**
  - `filename` : str
    - Filename of RNMRTK file to write to (.sec).
  - `data` : array_like
    - Array of NMR data.
  - `dtype` : dtype
    - Data type to convert data to before writing to disk.
  - `overwrite` : bool, optional
    - True to overwrite existing files. False will raise a Warning if the file exists.

- **See also**:
  - write Write RNMRTK files.

**nmrglue.fileio.rnmrtk.read_par**

nmrglue.fileio.rnmrtk.read_par(filename)

Parse a RNMRTK parameter (.par) file.

- **Parameters**
  - `file` : str
    - Filename of RNMRTK parameter file (.par) to read

- **Returns**
  - `dic` : dict
    - Dictionary of RNMRTK parameters.

**nmrglue.fileio.rnmrtk.write_par**

nmrglue.fileio.rnmrtk.write_par(par_file, dic, overwrite)

Write a RNMRTK parameter file (.par).

- **Parameters**
  - `par_file` : str
    - Filename of RNMRTK parameter file (.par) to write.
  - `dic` : dict
    - Dictionary of NMR parameters.
  - `overwrite` : bool
    - Set True to overwrite existing files, False will raise a Warning if the file exists.
nmrglue.fileio.rnmrtk.make_uc

nmrglue.fileio.rnmrtk.make_uc(dic, data, dim=-1)
Creat a unit conversion object

Parameters  
dic : dict
    Dictionary of RNMRTK parameters.

data : ndarray
    Array of NMR data.

dim : int, optional
    Dimension number to create unit conversion object for. Default is for the last dimension.

Returns  
uc : unit conversion object.
    Unit conversion object for given dimension.

nmrglue.fileio.rnmrtk.guess_udic

nmrglue.fileio.rnmrtk.guess_udic(dic, data)
Guess parameters of a universal dictionary from a dic, data pair.

Parameters  
dic : dict
    Dictionary of RNMRTK parameters.

data : ndarray
    Array of NMR data.

Returns  
udic : dict
    Universal dictionary of spectral parameters.

nmrglue.fileio.rnmrtk.create_dic

nmrglue.fileio.rnmrtk.create_dic(udic, dim_order=None)
Create a RNMRTK dictionary from a universal dictionary.

Parameters  
udic : dict
    Universal dictionary of spectral parameters.

dim_order : list, optional
    List mapping axis numbers in the universal dictionary to the order in which they will appear in the RNMRTK dictionary. If None, the default [0, 1, 2, ...] is used.

Returns  
dic : dict
    Dictionary of RNMRTK parameters.
Developer Information

Developer Functions

These functions are typically not used directly by users. Developers who want fine control over Rowland NMR Toolkit files will be interested in these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create_data(data)</code></td>
<td>Create a RNMRTK data array (recast into float32 or complex64)</td>
</tr>
<tr>
<td><code>get_data(filename, dtype)</code></td>
<td>Get spectral data from a RNMRTK file.</td>
</tr>
<tr>
<td><code>get_trace(f, num_points, big)</code></td>
<td>Get a trace from an open RNMRTK file.</td>
</tr>
<tr>
<td><code>put_trace(f, trace)</code></td>
<td>Put a trace to an open RNMRTK file.</td>
</tr>
<tr>
<td><code>uninterleave_data(data)</code></td>
<td>Remove interleaving of real.</td>
</tr>
<tr>
<td><code>interleave_data(data)</code></td>
<td>Interleave real, imag data in data</td>
</tr>
<tr>
<td><code>parse_par_line(line, dic)</code></td>
<td>Parse a line from a RNMRTK parameter file (.par).</td>
</tr>
<tr>
<td><code>find_dic_dim(dic, dim)</code></td>
<td>Find dimension in dictionary which corresponds to array dimension.</td>
</tr>
<tr>
<td><code>find_array_dim(dic, ddim)</code></td>
<td>Find array dimension which corresponds to dictionary dimension.</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.rnmrtk.create_data**

Create a RNMRTK data array (recast into float32 or complex64)

**nmrglue.fileio.rnmrtk.get_data**

Get spectral data from a RNMRTK file.

**Parameters**

- **filename** : str
  - Filename of RNMRTK file (.sec) to get data from.
- **dtype** : dtype
  - Type of data in file, typically ‘float32’

**Returns**

- **rdata** : ndarray
  - Raw NMR data, unshaped and typically not complex.

**nmrglue.fileio.rnmrtk.get_trace**

Get a trace from an open RNMRTK file.

**Parameters**

- **f** : file object
  - Open file object to read from.
- **num_points** : int
  - Number of points in trace (R+I)
- **big** : bool
True for data that is big-endian, False for little-endian.

**Returns**

`trace : ndarray`

Raw trace of NMR data.

### `nmrglue.fileio.rnmrtk.put_trace`

**nmrglue.fileio.rnmrtk.put_trace**

**put_trace**

`f, trace`  
Put a trace to an open RNMRTK file.

**Parameters**

- **f** : file object  
  Open file object to read from.
- **trace** : ndarray  
  Raw trace of NMR data, may be complex64.

### `nmrglue.fileio.rnmrtk.uninterleave_data`

**nmrglue.fileio.rnmrtk.uninterleave_data**

**uninterleave_data**

`data`

Remove interleaving of real, imag data in last dimension of data.

### `nmrglue.fileio.rnmrtk.interleave_data`

**nmrglue.fileio.rnmrtk.interleave_data**

**interleave_data**

`data`

Interleave real, imag data in data.

### `nmrglue.fileio.rnmrtk.parse_par_line`

**nmrglue.fileio.rnmrtk.parse_par_line**

**parse_par_line**

`line, dic`

Parse a line from a RNMRTK parameter file (.par).

### `nmrglue.fileio.rnmrtk.find_dic_dim`

**nmrglue.fileio.rnmrtk.find_dic_dim**

**find_dic_dim**

`dic, dim`

Find dimension in dictionary which corresponds to array dimension.

**Parameters**

- **dic** : dict  
  Dictionary of RNMRTK parameters.
- **dim** : int, non-negative  
  Dimension of data array.

**Returns**

`ddim : int`

Dimension in dic which corresponds to array dimension, dim.
nmrglue.fileio.rnmrtk.find_array_dim

nmrglue.fileio.rnmrtk.find_array_dim(dic, ddim)
Find array dimension which corresponds to dictionary dimension.

Parameters
dic : dict
   Dictionary of RNMRTK parameters.
ddim : int, non-negative
   Dimension in dictionary.

Returns
dim : int
   Dimension in array which corresponds to dictionary dimension, ddim.

Developer Classes

rnmrtk_nd(filename, fshape, cplex, big[, order])
Emulate a ndarray objects without loading data into memory for low memory reading of RNMRTK files.

class nmrglue.fileio.rnmrtk.rnmrtk_nd(filename, fshape, cplex, big, order=None)
Emulate a ndarray objects without loading data into memory for low memory reading of RNMRTK files.

   • slicing operations return ndarray objects.
   • can iterate over with expected results.
   • transpose and swapaxes methods create a new objects with correct axes ordering.
   • has ndim, shape, and dtype attributes.

Parameters
filename : str
   Filename of RNMRTK file (.sec) to read.
fshape : tuple of ints
   Shape of data in file.
cplex : bool
   True if the last (fast) axis is complex.
big : bool
   True for big-endian data, False for little-endian.
order : tuple
   Ordering of axes against file. None for (0, 1, 2,...).

__init__(filename, fshape, cplex, big, order=None)
Create and set up
Methods
__init__(filename, fshape, cplex, big[, order]) Create and set up
swapaxes(axis1, axis2) Return object with axis1 and axis2 interchanged.
transpose(*axes) Return object with axes transposed.

3.1.9 nmrglue.simpson

Functions for reading and writing files created by the SIMPSON simulations program.
This modules is imported as nmrglue.simpson and can be called as such.

User Information

User Functions

read(filename[, ftype, ndim, NP, NI, spe]) Read a SIMPSON file.
read_text(filename) Read a SIMPSON text file.
read_xreim(filename) Read a 1D indexed SIMPSON file.
read_xyreim(filename) Read a 2D indexed SIMPSON file.
read_raw_bin_1d(filename[, spe]) Read a 1D raw binary SIMPSON file.
read_raw_bin_2d(filename, NP, NI[, spe]) Read a 2D raw binary SIMPSON file.
read_binary(filename) Read a binary SIMPSON file.

nmrglue.fileio.simpson.read

nmrglue.fileio.simpson.read(filename, ftype=None, ndim=None, NP=None, NI=None, spe=None) Read a SIMPSON file.

Read a NMR data file saved using in a number of formats produced using the SIMPSON simulation program.

Parameters

filename : str
Name of SIMPSON file to read data from.

A string indicating the type of SIMPSON file:

TEXT : SIMPSON text format, no fsave arguments.
BINARY : SIMPSON binary format, -binary fsave argument.
XREIM : Indexed 1D format, rows of frequency/time, real and imaginary parts of the data. Saved with -xreim argument.
XYREIM : Indexed 2D format, rows of frequency/time (indirect dimension then direct dimension), real and imaginary parts of the data. Saved with -xyreim argument.
RAWBIN : Raw binary format. Saved with -raw_bin argument. ndim and spe must also be provided. In addition if ndim is 2, NP and NI must be defined.
None : Automatically determine file type. If this fails the file type should be implicitly provided.

Other formats of files may be created by SIMPSON, but are not currently supported.
**Returns**

dic : dict

Dictionary of spectra parameters. For some file formats this may be empty.

data : ndarray

Complex array of spectral data.

**Other Parameters**

ndim : {None, 1, 2}, optional

Dimensionality of the data in the file, only used when ftype is “RAWBIN”.

NP : int, optional

Number of points {R|I} in the direct dimension. Only used when ftype is “RAWBIN” and ndim is 2.

NI : int, optional

Number of points in the indirect dimension. Only used when ftype is “RAWBIN” and ndim is 2.

spe : bool, optional

True when the data is in the frequency domain, False for time domain data. Only used when ftype is “RAWBIN”

```
nmrglue.fileio.simpson.read_text
```

Read a SIMPSON text file. See `read()`.

```
nmrglue.fileio.simpson.read_xreim
```

Read a 1D indexed SIMPSON file. See `read()`.

```
nmrglue.fileio.simpson.read_xyreim
```

Read a 2D indexed SIMPSON file. See `read()`.

```
nmrglue.fileio.simpson.read_raw_bin_1d
```

Read a 1D raw binary SIMPSON file. See `read()`.

```
nmrglue.fileio.simpson.read_raw_bin_2d
```

Read a 2D raw binary SIMPSON file. See `read()`.
nmrglue.fileio.simpson.read_binary

nmrglue.fileio.simpson.read_binary(filename)
  Read a binary SIMPSON file. See read().

Developer Information

Developer Functions

These functions are typically not used directly by users. Developers who want fine control over SIMPSON files will be interested in these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>guess_ftype(filename)</td>
<td>Determine a SIMPSON file type from the first few lines of the file.</td>
</tr>
<tr>
<td>unappend_data(data)</td>
<td>Return complex data with last axis unappended.</td>
</tr>
<tr>
<td>chars2bytes(chars)</td>
<td>Convert four characters from a data block into 3 ‘bytes’.</td>
</tr>
<tr>
<td>bytes2float(bytes)</td>
<td>Convert four bytes to a string.</td>
</tr>
</tbody>
</table>

nmrglue.fileio.simpson.guess_ftype

nmrglue.fileio.simpson.guess_ftype(filename)
  Determine a SIMPSON file type from the first few lines of the file.

nmrglue.fileio.simpson.unappend_data

nmrglue.fileio.simpson.unappend_data(data)
  Return complex data with last axis unappended.
  Data should have imaginary data vector appended to real data vector.

nmrglue.fileio.simpson.chars2bytes

nmrglue.fileio.simpson.chars2bytes(chars)
  Convert four characters from a data block into 3 ‘bytes’.

nmrglue.fileio.simpson.bytes2float

nmrglue.fileio.simpson.bytes2float(bytes)
  Convert four bytes to a string.

3.1.10 nmrglue.sparky

Functions for reading and writing Sparky (.ucsf) files.
This module is imported as nmrglue.sparky and can be called as such.
User Information

User Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read(filename)</code></td>
<td>Read a Sparky file.</td>
</tr>
<tr>
<td><code>write(filename, dic, data[, overwrite])</code></td>
<td>Write a Sparky file.</td>
</tr>
<tr>
<td><code>read_lowmem(filename)</code></td>
<td>Read a Sparky file using minimal memory.</td>
</tr>
<tr>
<td><code>write_lowmem(filename, dic, data[, overwrite])</code></td>
<td>Write a Sparky using minimum amounts of memory (tile by tile)</td>
</tr>
<tr>
<td><code>make_uc(dic, data[, dim])</code></td>
<td>Create a unit conversion object.</td>
</tr>
<tr>
<td><code>guess_udic(dic, data)</code></td>
<td>Guess parameter of universal dictionary from dic, data pair.</td>
</tr>
<tr>
<td><code>create_dic(udic[, datetimeobj, user])</code></td>
<td>Create a Sparky parameter dictionary from universal dictionary.</td>
</tr>
<tr>
<td><code>datetime2dic(datetimeobj, dic)</code></td>
<td>Add datetime object to dictionary</td>
</tr>
<tr>
<td><code>dic2datetime(dic)</code></td>
<td>Create a datetime object from a Sparky dictionary</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.sparky.read**

```
nmrglue.fileio.sparky.read(filename)
```

Read a Sparky file.

**Parameters**

- `filename : str`  
  Filename of Sparky file to read.

**Returns**

- `dic : dict`  
  Dictionary of Sparky parameters.

- `data : ndarray`  
  Array of NMR data.

**See also:**

- `read_lowmem` Sparky file reading with minimal memory usage.
- `write` Write a Sparky file.

**nmrglue.fileio.sparky.write**

```
nmrglue.fileio.sparky.write(filename, dic, data, overwrite=False)
```

Write a Sparky file.

**Parameters**

- `filename : str`  
  Filename of Sparky file to write to.

- `dic : dict`  
  Dictionary of Sparky parameters.

- `data : array_like`  
  Array of NMR data.

- `overwrite : bool, optional`  
  Overwrite existing file if True.

3.1. fileio modules
Set True to overwrite files, False will raise a Warning if the file exists.

See also:

- **write_lowmem** Write a Sparky file using minimal amounts of memory.
- **read** Read a Sparky file.

### nmrglue.fileio.sparky.read_lowmem

**nmrglue.fileio.sparky.read_lowmem**

Read a Sparky file using minimal memory.

**Parameters**

- **filename** : str
  
  Filename of Sparky file to read.

**Returns**

- **dic** : dict
  
  Dictionary of Sparky parameters.

- **data** : array_like
  
  Low memory object which can access NMR data on demand.

See also:

- **read** Read a Sparky file.
- **write_lowmem** Write a Sparky file using minimal memory.

### nmrglue.fileio.sparky.write_lowmem

**nmrglue.fileio.sparky.write_lowmem**

Write a Sparky using minimum amounts of memory (tile by tile)

**Parameters**

- **filename** : str
  
  Filename of Sparky file to write to.

- **dic** : dict
  
  Dictionary of Sparky parameters.

- **data** : array_like.
  
  Array of NMR data.

- **overwrite** : bool, optional
  
  Set True to overwrite files, False will raise a Warning if the file exists.

See also:

- **write** Write a Sparky file.
- **read_lowmem** Read a Sparky file using minimal amounts of memory.
nmrglue.fileio.sparky.make_uc

nmrglue.fileio.sparky.make_uc(dic, data, dim=-1)
Create a unit conversion object.

Parameters
dic : dict
       Dictionary of Sparky parameters.
data : ndarray
       Array of NMR data.
dim : int, optional
       Dimension number to create unit conversion object for. Default is for last dimension.

Returns
uc : unit conversion object.
       Unit conversion object for given dimension.

nmrglue.fileio.sparky.guess_udic

nmrglue.fileio.sparky.guess_udic(dic, data)
Guess parameter of universal dictionary from dic, data pair.

Parameters
dic : dict
       Dictionary of Sparky parameters.
data : ndarray
       Array of NMR data.

Returns
udic : dict
       Universal dictionary of spectral parameter.

nmrglue.fileio.sparky.create_dic

nmrglue.fileio.sparky.create_dic(udic, datetimeobj=datetime.datetime(2017, 11, 29, 18, 21, 13, 32278), user='user')
Create a Sparky parameter dictionary from universal dictionary.

Parameters
udic : dict
       Universal dictionary of spectral parameters.
datetimeobj : datetime object, optional
       Datetime to record in Sparky dictionary
user : str, optional
       Username to record in Sparky dictionary. Default is ‘user’

Returns
dic : dict
       Dictionary of Sparky parameters.
nmrglue Documentation, Release 0.7-dev

nmrglue.fileio.sparky.datetime2dic

nmrglue.fileio.sparky.datetime2dic(datetimeobj, dic)
Add datetime object to dictionary

nmrglue.fileio.sparky.dic2datetime

nmrglue.fileio.sparky.dic2datetime(dic)
Create a datetime object from a Sparky dictionary

Developer Information

from __future__ import print_function
__developer_info__ = """" Sparky file format information

Developer Functions

These functions are typically not used directly by users. Developers who want fine control over Sparky files may be interested in these functions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_data(data)</td>
<td>Create a Sparky data array (recast into float32 array)</td>
</tr>
<tr>
<td>create_axisdicadic, tlen, dlen</td>
<td>Make an Sparky axis dictionary from a universal axis dictionary.</td>
</tr>
<tr>
<td>calc_tshape(shape[, kbyte_max])</td>
<td>Calculate a tile shape from data shape.</td>
</tr>
<tr>
<td>read_2D(filename)</td>
<td>Read a 2D sparky file.</td>
</tr>
<tr>
<td>write_2D(filename, dic, data[, overwrite])</td>
<td>Write a 2D Sparky file.</td>
</tr>
<tr>
<td>read_3D(filename)</td>
<td>Read a 3D Sparky file.</td>
</tr>
<tr>
<td>write_3D(filename, dic, data[, overwrite])</td>
<td>Write a 3D Sparky file.</td>
</tr>
<tr>
<td>read_lowmem_2D(filename)</td>
<td>Read a 2D Sparky file using minimal memory.</td>
</tr>
<tr>
<td>read_lowmem_3D(filename)</td>
<td>Read a 3D sparky file using minimal memory.</td>
</tr>
<tr>
<td>get_tilen(f, n_tile, tw_tuple)</td>
<td>Read a tile from a Sparky file object.</td>
</tr>
<tr>
<td>get_tile(f, num_points)</td>
<td>Read the next tile from a Sparky file object.</td>
</tr>
<tr>
<td>put_tile(f, tile)</td>
<td>Put a tile to a Sparky file object.</td>
</tr>
<tr>
<td>get_data(f)</td>
<td>Read all data from sparky file object.</td>
</tr>
<tr>
<td>put_data(f, data)</td>
<td>Put data to a Sparky file object.</td>
</tr>
<tr>
<td>find_tilen_2d(data, ntile, tile_size)</td>
<td>Return a tile from a 2D NMR data set.</td>
</tr>
<tr>
<td>tile_data2d(data, tile_size)</td>
<td>Tile 2D data into a 1D array.</td>
</tr>
<tr>
<td>untile_data2d(data, tile_size, data_size)</td>
<td>Rearrange 2D Tiled/Sparky formatted data into standard format.</td>
</tr>
<tr>
<td>find_tilen_3d(data, ntile, tile_size)</td>
<td>Return a single tile from a 3D NMR data set.</td>
</tr>
<tr>
<td>tile_data3d(data, tile_size)</td>
<td>Tile 3D data into a 1D numpy array.</td>
</tr>
<tr>
<td>untile_data3d(data, tile_size, data_size)</td>
<td>Rearrange 3D tiled/Sparky formatted data into standard format.</td>
</tr>
<tr>
<td>get_fileheader(f)</td>
<td>Get fileheader from file and return a list.</td>
</tr>
<tr>
<td>put_fileheader(f, fl)</td>
<td>Write fileheader list to file (180-bytes).</td>
</tr>
<tr>
<td>fileheader2dic(header)</td>
<td>Convert a fileheader list into a Sparky parameter dictionary.</td>
</tr>
<tr>
<td>dic2fileheader(dic)</td>
<td>Convert a Sparky parameter dictionary into a fileheader list.</td>
</tr>
<tr>
<td>get_axisheader(f)</td>
<td>Get an axisheader from file and return a list.</td>
</tr>
<tr>
<td>put_axisheader(f, al)</td>
<td>Write an axisheader list to file (128-bytes written).</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.29 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axisheader2dic(header)</td>
<td>Convert an axisheader list into Sparky parameter axis dictionary.</td>
</tr>
<tr>
<td>dic2axisheader(dic)</td>
<td>Convert a Sparky parameter axis dictionary into an axisheader list.</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.sparky.create_data**

```python
nmrglue.fileio.sparky.create_data(data)
```
Create a Sparky data array (recast into float32 array)

**nmrglue.fileio.sparky.create_axisdic**

```python
nmrglue.fileio.sparky.create_axisdic(adic, tlen, dlen)
```
Make an Sparky axis dictionary from a universal axis dictionary.

**Parameters**
- `adic` : dict
  Axis dictionary from a universal dictionary.
- `tlen` : int
  Tile length of axis.
- `dlen` : int
  Data length of axis.

**Returns**
- `adic` : dict
  Sparky axis dictionary

**nmrglue.fileio.sparky.calc_tshape**

```python
nmrglue.fileio.sparky.calc_tshape(shape, kbyte_max=128)
```
Calculate a tile shape from data shape.

**Parameters**
- `shape` : tuple
  Shape of NMR data (data.shape).
- `kbyte_max` : float or int
  Maximum tile size in Kilobytes.

**Returns**
- `tshape` : tuple
  Shape of tile.

**nmrglue.fileio.sparky.read_2D**

```python
nmrglue.fileio.sparky.read_2D(filename)
```
Read a 2D sparky file. See `read()` for documentation.
nmrglue.fileio.sparky.write_2D

nmrglue.fileio.sparky.write_2D(filename, dic, data, overwrite=False)
Write a 2D Sparky file. See write() for documentation.

nmrglue.fileio.sparky.read_3D

nmrglue.fileio.sparky.read_3D(filename)
Read a 3D Sparky file. See read() for documentation.

nmrglue.fileio.sparky.write_3D

nmrglue.fileio.sparky.write_3D(filename, dic, data, overwrite=False)
Write a 3D Sparky file. See write() for documentation.

nmrglue.fileio.sparky.read_lowmem_2D

nmrglue.fileio.sparky.read_lowmem_2D(filename)
Read a 2D Sparky file using minimal memory. See read_lowmem().

nmrglue.fileio.sparky.read_lowmem_3D

nmrglue.fileio.sparky.read_lowmem_3D(filename)
Read a 3D sparky file using minimal memory. See read_lowmem().

nmrglue.fileio.sparky.get_tile

nmrglue.fileio.sparky.get_tile(f, n_tile, tw_tuple)
Read a tile from a Sparky file object.

Parameters

f : file object
Open file object pointing to a Sparky file.

n_tile : int
Tile number to read

tw_tuple : tuple of ints
Tile size

Returns

tile : ndarray
Tile of NMR data. Data is returned as a 1D array.

Notes

Current file position is lost. It can be stored before calling if the position is later needed.
nmrglue.fileio.sparky.get_tile

nmrglue.fileio.sparky.get_tile(f, num_points)
Read the next tile from a Sparky file object.

Parameters:
- **f**: file object
  - Open file object pointing to a Sparky file.
- **num_points**: int
  - Number of points in the tile.

Returns:
- **tile**: ndarray
  - Tile of NMR data. Data is returned as a 1D array.

nmrglue.fileio.sparky.put_tile

nmrglue.fileio.sparky.put_tile(f, tile)
Put a tile to a Sparky file object.

Parameters:
- **f**: file object
  - Open file object pointing to a Sparky file, to be written to.
- **tile**: ndarray
  - Tile of NMR data to be written.

nmrglue.fileio.sparky.get_data

nmrglue.fileio.sparky.get_data(f)
Read all data from sparky file object.

nmrglue.fileio.sparky.put_data

nmrglue.fileio.sparky.put_data(f, data)
Put data to a Sparky file object.

This function does not untile data. This should be done before calling this function.

nmrglue.fileio.sparky.find_tilen_2d

nmrglue.fileio.sparky.find_tilen_2d(data, ntile, tile_size)
Return a tile from a 2D NMR data set.

Parameters:
- **data**: 2D ndarray
  - NMR data, untiled/standard format.
- **ntile**: int
  - Tile number to extract.
- **(lentY, lentX)**: tuple of ints
  - Tile size (w1, w2).
Returns tile : 1D ndarray
Tile of NMR data, returned as 1D array.

Notes
Edge tiles are zero filled to the indicated tile size.

nmrglue.fileio.sparky.tile_data2d

nmrglue.fileio.sparky.tile_data2d (data, tile_size)
Tile 2D data into a 1D array.

Parameters data : 2D ndarray
NMR data, untiled/standard format.
(lentY, lentX) : tuple of ints
Tile size.

Returns tdata : 1D ndarray
Tiled/Sparky formatted NMR data, returned as 1D array.

nmrglue.fileio.sparky.untile_data2D

nmrglue.fileio.sparky.untile_data2D (data, tile_size, data_size)
Rearrange 2D Tiled/Sparky formatted data into standard format.

Parameters data : 1D ndarray
Tiled/Sparky formatted 2D NMR data.
(lentY, lentX) : tuple of ints
Size of tile.

Returns sdata : 2D ndarray
NMR data, untiled/standard format.

nmrglue.fileio.sparky.find_tile_3d

nmrglue.fileio.sparky.find_tile_3d (data, ntile, tile_size)
Return a single tile from a 3D NMR data set.

Parameters data : 3D ndarray
NMR data, untiled/standard format.
nite : int
Tile number to extract.
(lentZ, lentY, lentX) : tuple of ints
Tile size \((w_1, w_2, w_3)\).

**Returns**

**tile** : 1D ndarray

Tile of NMR data, returned as 1D array.

**Notes**

Edge tiles are zero filled to the indicated tile size.

### nmrglue.fileio.sparky.tile_data3d

**nmrglue.fileio.sparky.tile_data3d**

\(\text{nmrglue.fileio.sparky.tile_data3d}(data, \text{tile\_size})\)

Tile 3D data into a 1D numpy array

**Parameters**

**data** : 3D ndarray

NMR data, untiled/standard format.

\((\text{lentZ}, \text{lentY}, \text{lentX})\) : tuple of ints

Tile size \((w_1, w_2, w_3)\).

**Returns**

**tile** : 1D ndarray

Tiled/Sparky formatted NMR data, returned as 1D array.

### nmrglue.fileio.sparky.untile_data3D

**nmrglue.fileio.sparky.untile_data3D**

\(\text{nmrglue.fileio.sparky.untile_data3D}(data, \text{tile\_size}, \text{data\_size})\)

Rearrange 3D tiled/Sparky formatted data into standard format.

**Parameters**

**data** : 1D ndarray

Tiled/Sparky formatted 2D NMR data.

\((\text{lentZ}, \text{lentY}, \text{lentX})\) : tuple of ints

Size of tile

\((\text{lenZ}, \text{lenY}, \text{lenX})\) : tuple of ints

Size of NMR data.

**Returns**

**sdata** : 3D ndarray

NMR data, untiled/standard format.

### nmrglue.fileio.sparky.get_fileheader

**nmrglue.fileio.sparky.get_fileheader**

\(\text{nmrglue.fileio.sparky.get_fileheader}(f)\)

Get fileheader from file and return a list.

Reads the 180 byte file header of a Sparky file
nmrglue.fileio.sparky.put_fileheader

nmrglue.fileio.sparky.put_fileheader\((f, fl)\)
Write fileheader list to file (180-bytes).

nmrglue.fileio.sparky.fileheader2dic

nmrglue.fileio.sparky.fileheader2dic\((header)\)
Convert a fileheader list into a Sparky parameter dictionary.

nmrglue.fileio.sparky.dic2fileheader

nmrglue.fileio.sparky.dic2fileheader\((dic)\)
Convert a Sparky parameter dictionary into a fileheader list.

nmrglue.fileio.sparky.get_axisheader

nmrglue.fileio.sparky.get_axisheader\((f)\)
Get an axisheader from file and return a list.
Only the first 44 bytes are examined, the NMR_PROCESSED and other header parameters are ignored since the current version of Sparky does not use them.

nmrglue.fileio.sparky.put_axisheader

nmrglue.fileio.sparky.put_axisheader\((f, al)\)
Write an axisheader list to file (128-bytes written).

nmrglue.fileio.sparky.axisheader2dic

nmrglue.fileio.sparky.axisheader2dic\((header)\)
Convert an axisheader list into Sparky parameter axis dictionary.

nmrglue.fileio.sparky.dic2axisheader

nmrglue.fileio.sparky.dic2axisheader\((dic)\)
Convert a Sparky parameter axis dictionary into an axisheader list.

Developer Classes

<table>
<thead>
<tr>
<th>sparky_2d(filename[, order])</th>
<th>Emulates a ndarray object without loading data into memory for low memory reading of 2D Sparky files.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sparky_3d(filename[, order])</td>
<td>Emulates a ndarray object without loading data into memory for low memory read of 3D Sparky files.</td>
</tr>
</tbody>
</table>
nmrglue.fileio.sparky.sparky_2d

class nmrglue.fileio.sparky.sparky_2d(filename, order=None)
    Emulates a ndarray object without loading data into memory for low memory reading of 2D Sparky files.
    • slicing operations return ndarray objects.
    • can iterate over with expected results.
    • transpose and swapaxes methods create a new objects with correct axes ordering.
    • has ndim, shape, and dtype attributes.

Parameters

filename : str
    Filename of 2D Sparky file.

order : tuple, optional
    Order of axes against file. None is equivalent to (0, 1).

__init__(filename, order=None)
    Create and set up object

Methods

__init__(filename[, order]) Create and set up object
swapaxes(axis1, axis2) Return object with axis1 and axis2 interchanged.
transpose(*axes) Return object with axes transposed.

nmrglue.fileio.sparky.sparky_3d

class nmrglue.fileio.sparky.sparky_3d(filename, order=None)
    Emulates a ndarray object without loading data into memory for low memory read of 3D Sparky files.
    • slicing operations return ndarray objects.
    • can iterate over with expected results.
    • transpose and swapaxes methods create a new objects with correct axes ordering.
    • has ndim, shape, and dtype attributes.

Parameters

filename : str
    Filename of 3D Sparky file.

order : tuple
    Ordering of axes against file. None is equivalent to (0, 1, 2)

__init__(filename, order=None)
    Create and set up object

Methods
__init__(filename[, order])  Create and set up object
swapaxes(axis1, axis2)  Return object with axis1 and axis2 interchanged.
transpose(*axes)  Return object with axes transposed.

3.1.11 nmrglue.table

nmrglue table functions.
nmrglue uses numpy records array as stores of various data (peak tables, trajectories, etc). This module provides
functions to read and write records arrays from disk. Formatting of the numeric values is left to Python’s str function
and only the data type need be specified. In addition this module contains functions to convert nmrglue’s table format
NMRPipe’s table format.

This modules is imported as nmrglue.table and can be called as such.

User Information
User Functions

read(filename)  Read a nmrglue table file.
write(filename, comments, rec[, overwrite])  Write a nmrglue table to file.
insert_row(rec, N, row)  Insert a row into a records array before row number N.
append_row(rec, row)  Append a row to the end of a records array.
delete_row(rec, N)  Delete a row from a records array.
reorder_rows(rec, new_order)  Reorder or delete rows in a records array.
insert_column(rec, N, col[, name, format])  Insert a column into a records array.
append_column(rec, col[, name, format])  Append a column to the end of a records array.
delete_column(rec, N)  Delete a column from a records array.
reorder_columns(rec, new_order)  Reorder or delete columns in a records array.
pipe2glue(pcomments, pformat, rec)  Convert a NMRPipe table to a nmrglue table
glue2pipe(comments, rec)  Convert a nmrglue table to a NMRPipe table.
guess_pformat(col)  Guess a NMRPipe table column format string given a column.

nmrglue.fileio.table.read

nmrglue.fileio.table.read(filename)
Read a nmrglue table file.

Parameters  filename : str
            Filename of nmrglue table file to read.

Returns  comments : list
            List of comments (strings terminated with newline)

            rec : recarray
            Records array with named fields.
nmrglue.fileio.table.write

nmrglue.fileio.table.write(filename, comments, rec, overwrite=False)

Write a nmrglue table to file.

Parameters
    filename : str
        Filename of file to write table to.
    comments : list
        List of comments (strings terminated with newline).
    rec : recarray
        Records array to write to file.
    overwrite : bool, optional
        True to overwrite file if it exists. False will raise an Warning if the file exists.

nmrglue.fileio.table.insert_row

nmrglue.fileio.table.insert_row(rec, N, row)

Insert a row into a records array before row number N.

Parameters
    rec : recarray
        Records array.
    N : int
        Row number to insert new row before.
    row : array_like
        Array or similar object which will be converted into a new row.

Returns
    new_rec : recarray
        New records array with inserted row.

nmrglue.fileio.table.append_row

nmrglue.fileio.table.append_row(rec, row)

Append a row to the end of a records array.

Parameters
    rec : recarray
        Records array.
    row : array_like
        Array or similar object which will be converted into a new row.

Returns
    new_rec : recarray
        New records array with inserted row.
nmrglue.fileio.table.delete_row

nmrglue.fileio.table.delete_row(rec, N)
Delete a row from a records array.

**Parameters**
- `rec`: recarray
  Records array.
- `N`: int
  Row number to delete.

**Returns**
- `new_rec`: recarray
  New records array with row deleted.

**See also:**
- reorder_rows delete multiple rows in a single call.

nmrglue.fileio.table.reorder_rows

nmrglue.fileio.table.reorder_rows(rec, new_order)
Reorder or delete rows in a records array.

This function can also be used to delete multiple rows from a records array, only the rows in the new_order list are retained in the new records array.

**Parameters**
- `rec`: recarray
  Records array.
- `new_order`: list
  List of row indices and order in new records array. Only the rows in this list are retained in the new records array. Therefore this function can also be used to delete multiple rows from a records array.

**Returns**
- `new_rec`: recarray
  New records array with rows reordered.

nmrglue.fileio.table.insert_column

nmrglue.fileio.table.insert_column(rec, N, col, name=None, format=None)
Insert a column into a records array.

**Parameters**
- `rec`: recarray
  Records array.
- `col`: array_like
  Array or similar object which will be converted into the new column.
- `N`: int
  Column number to insert new column before.
- `name`: str, optional
  Name of the column. If None col.dtypes.name will be used.
**format**: dtype, optional

Data type to convert the new column into before appending. Required if col is not an ndarray.

**Returns**

**new_rec**: recarray

New records array with column inserted.

**nmrglue.fileio.table.append_column**

nmrglue.fileio.table.append_column(rec, col=None, name=None, format=None)

Append a column to the end of a records array.

**Parameters**

**rec**: recarray

Records array.

**col**: array_like

Array or similar object which will be converted into the new column.

**name**: str, optional

Name of the column. If None col.dtype.name will be used.

**format**: dtype, optional

Data type to convert the new column into before appending. Required if col is not an ndarray.

**Returns**

**new_rec**: recarray

New records array with column appended.

**nmrglue.fileio.table.delete_column**

nmrglue.fileio.table.delete_column(rec, N)

Delete a column from a records array.

**Parameters**

**rec**: recarray

Records array.

**N**: int

Column number to delete.

**Returns**

**new_rec**: recarray

New records array with column deleted.

**See also:**

reorder_columns Delete multiple columns from a records array.

**nmrglue.fileio.table.reorder_columns**

nmrglue.fileio.table.reorder_columns(rec, new_order)

Reorder or delete columns in a records array.
Parameters rec : recarray

   Records array.

new_order : list

   List of column indices and order in new records array. Only the columns in this
   list are retained in the new records array. Therefore this function can also be used to delete
   multiple columns from a records array.

Returns new_rec : recarray

   New records array with columns reordered.

`nmrglue.fileio.table.pipe2glue`

`nmrglue.fileio.table.pipe2glue`(pcomments, pformat, rec)

Convert a NMRPipe table to a nmrglue table

Parameters pcomments : list

   List of NMRPipe comment lines.

pformats : list

   List of NMRPipe table column formats strings.

rec : recarray

   Records array with named fields.

Returns comments : list

   List of comments

rec : recarray

   Records array with named fields.

`nmrglue.fileio.table.glue2pipe`

`nmrglue.fileio.table.glue2pipe`(comments, rec)

Convert a nmrglue table to a NMRPipe table.

Parameters comments : list

   List of comments

rec : recarray

   Records array with named fields.

Returns pcomments : list

   List of NMRPipe comment lines.

pformats : list

   List of NMRPipe table column formats strings. This list is guessed from the data types
   and precision in the reconrds array. This may not be the exact format desired, edit this
   to your liking.

rec : recarray
Records array with named fields.

**nmrglue.fileio.table.guess_pformat**

**nmrglue.fileio.table.guess_pformat(col)**

Guess a NMRPipe table column format string given a column.

**Parameters**

- **col**: ndarray
  - Array from a records array.

**Returns**

- **s**: str
  - String for formatting NMRPipe table.

### 3.1.12 nmrglue.tecmag

Functions for reading Tecmag .tnt data files.

This module is imported as nmrglue.tecmag and can be called as such.

#### User Information

**User Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read(filename)</td>
<td>Read a Tecmag .tnt data file.</td>
</tr>
<tr>
<td>guess_udic(dic, data)</td>
<td>Guess parameters of universal dictionary from dic, data pair.</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.tecmag.read**

**nmrglue.fileio.tecmag.read(filename)**

Read a Tecmag .tnt data file.

**Parameters**

- **filename**: str
  - Name of file to read from

**Returns**

- **dic**: dict
  - Dictionary of Tecmag parameters.
  - **data**: ndarray
    - Array of NMR data.

**nmrglue.fileio.tecmag.guess_udic**

**nmrglue.fileio.tecmag.guess_udic(dic, data)**

Guess parameters of universal dictionary from dic, data pair.

**Parameters**

- **dic**: dict
  - Dictionary of Tecmag parameters.
**nmrglue Documentation, Release 0.7-dev**

```text

**data** : ndarray

Array of NMR data.

**Returns udic** : dict

Universal dictionary of spectral parameters.

---

**Developer Information**

**Tecmag .tnt file format information**

The Tecmag .tnt file format is documented with C pseudo-code in the file “A1 - TNMR File Format.doc” distributed with the TNMR software.

This file is based on the pytnt module. Please inform upstream if you find a bug or to request additional features.

**3.1.13 nmrglue.varian**

Functions for reading and writing Agilent/Varian binary (fid) files and parameter (procpar) files.

This modules is imported as nmrglue.varian and can be called as such. These functions and classes can also be access from nmrglue.agilent.

**User Information**

**User Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read</code></td>
<td>Read Agilent/Varian files in a directory.</td>
</tr>
<tr>
<td><code>write</code></td>
<td>Write Agilent/Varian files to a directory.</td>
</tr>
<tr>
<td><code>write_lowmem</code></td>
<td>Write Agilent/Varian files to a directory using minimal amounts of memory.</td>
</tr>
<tr>
<td><code>read_lowmem</code></td>
<td>Read Agilent/Varian files in a directory using minimal amount of memory.</td>
</tr>
<tr>
<td><code>read_fid</code></td>
<td>Read a Agilent/Varian binary (fid) file.</td>
</tr>
<tr>
<td><code>write_fid</code></td>
<td>Write a Agilent/Varian binary (fid) file.</td>
</tr>
<tr>
<td><code>read_fid_lowmem</code></td>
<td>Read a Agilent/Varian binary (fid) file using minimal amounts of memory.</td>
</tr>
<tr>
<td><code>write_fid_lowmem</code></td>
<td>Write a Agilent/Varian binary (fid) file using minimal amounts of memory.</td>
</tr>
<tr>
<td><code>read_fid_ntraces</code></td>
<td>Read a Agilent/Varian binary (fid) file possibility having multiple traces per block.</td>
</tr>
<tr>
<td><code>read_procpar</code></td>
<td>Read a procpar file returning a dictionary of procpar parameters.</td>
</tr>
<tr>
<td><code>write_procpar</code></td>
<td>Write a Agilent/Varian procpar file from a dictionary</td>
</tr>
<tr>
<td><code>guess_udic</code></td>
<td>Guess parameter of a universal dictionary from dic, data pair.</td>
</tr>
<tr>
<td><code>create_dic</code></td>
<td>Create a Agilent/Varian parameter dictionary from a universal dictionary.</td>
</tr>
</tbody>
</table>
```

---

86 Chapter 3. Reference Guide
nmrglue.fileio.varian.read

nmrglue.fileio.varian.read(dir='.', fid_file='fid', procpar_file='procpar', read_blockhead=False, shape=None, torder=None, as_2d=False)

Read Agilent/Varian files in a directory.

Parameters
- **dir**: str, optional
  Directory holding Agilent/Varian data. Default is the current working directory.
- **fid_file**: str, optional
  Filename of binary (fid) file in directory.
- **procpar_file**: str, optional
  Filename of procpar file in directory.
- **read_blockhead**: bool, optional
  True to read blockheader(s) and return then in the Agilent/Varian parameter dictionary. False (default) does not perform this reading.
- **shape**: tuple of ints, optional
  Shape of data in binary file. None (default) will attempt to finds this automatically.
- **torder**: {None, ‘r’, ‘o’, ‘f’ or a Python function}, optional
  Description of the mapping of traces in the file to the NMR data matrix. None (the default) will attempt to find this automatically which is typically fine for most NMR experiments. See below for additional details.
- **as_2d**: bool, optional
  True to return data as a 2D array ignoring the shape and torder parameters.

Returns
- **dic**: dict
  Dictionary of Agilent/Varian parameters.
- **data**: ndarray
  Array of NMR data

See also:
- *read_lowmem* Read Agilent/Varian files using mimimal amounts of memory.
- *write* Write Agilent/Varian files.

Notes

The torder parameter describes how the traces on disk should be re-organized to form the NMR data matrix. In most cases this can be determined automatically by examining the order of phase parameters in the procpar file. This is done if torder is set to None. In some cases it must be explicitly provided. Three common cases are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Ordering of arrays</th>
<th>torder</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>d3, d2, phase2, phase</td>
<td>‘regular’ or ‘r’</td>
<td></td>
</tr>
<tr>
<td>opposite</td>
<td>d3, d2, phase, phase2</td>
<td>‘opposite’ or ‘o’</td>
<td></td>
</tr>
<tr>
<td>flat</td>
<td>As data exists in file</td>
<td>‘flat’ or ‘f’</td>
<td>Valid for 1D or 2D data</td>
</tr>
</tbody>
</table>

In addition a function which maps indirect dimension index tuples to/from trace numbers as stored on disk can be provided. For reading this function should take 2 arguments: shape, index_tuple and return an integer trace.
nmrglue Documentation, Release 0.7-dev

number. For writing this function should again take 2 arguments: shape,trace_number and return the indirect
dimension index tuple for the given trace.

nmrglue.fileio.varian.write

nmrglue.fileio.varian.write(dir, dic, data, fid_file='fid', procpar_file='procpar', torder=None,
repack=False, overwrite=False)

Write Agilent/Varian files to a directory.

Parameters dir : str
    Name of the directory to write to.

dic : dict
    Dictionary of Agilent/Varian parameters.

data : array_like
    Array of NMR data to write.

fid_file : str, optional
    Filename of binary (fid) file in directory to write to.

procpar_file : str, optional
    Filename of procpar file in directory to write to.

torder : {None, ‘r’, ‘o’, ‘f’ or a Python function} , optional
    Description of the mapping of traces in the file to the NMR data matrix. None (the
default) will attempt to find this automatically which is typically fine for most NMR
experiments. See read() for additional details.

repack : bool, optional
    True to repack file and block headers. False (default) leaves these as is.

overwrite : bool, optional
    Set to True to overwrite existing files, False will raise a Warning if files already exist.

See also:

write_lowmem Write Agilent/Varian files using minimal memory

read Read Agilent/Varian files.

nmrglue.fileio.varian.write_lowmem

nmrglue.fileio.varian.write_lowmem(dir, dic, data, fid_file='fid', procpar_file='procpar',
torder=None, repack=False, overwrite=False)

Write Agilent/Varian files to a directory using minimal amounts of memory.

Parameters dir : str
    Name of the directory to write to.

dic : dict
    Dictionary of Agilent/Varian parameters.

data : array_like
Array of NMR data to write.

**fid_file**: str, optional

Filename of binary (fid) file in directory to write to.

**procpar_file**: str, optional

Filename of procpar file in directory to write to.

**torder**: {None, ‘r’, ‘o’, ‘f’ or a Python function}, optional

Description of the mapping of traces in the file to the NMR data matrix. None (the default) will attempt to find this automatically which is typically fine for most NMR experiments. See `read()` for additional details.

**repack**: bool, optional

True to repack file and block headers. False (default) leaves these as is.

**overwrite**: bool, optional

Set to True to overwrite existing files, False will raise a Warning if files already exist.

See also:

- **write** Write Agilent/Varian files.
- **read_lowmem** Read Agilent/Varian using minimal amounts of memory.

**nmrglue.fileio.varian.read_lowmem**

Read Agilent/Varian files in a directory using minimal amount of memory.

**Parameters**

- **dir**: str, optional
  
  Directory holding Agilent/Varian data. Default is the current working directory.

- **fid_file**: str, optional
  
  Filename of binary (fid) file in directory.

- **procpar_file**: str, optional
  
  Filename of procpar file in directory.

- **read_blockhead**: bool, optional
  
  True to read blockheader(s) and return then in the Agilent/Varian parameter dictionary. False (default) does not perform this reading.

- **shape**: tuple of ints, optional
  
  Shape of data in binary file. None (default) will attempt to finds this automatically.

- **torder**: {None, ‘r’, ‘o’, ‘f’ or a Python function}, optional
  
  Description of the mapping of traces in the file to the NMR data matrix. None (the default) will attempt to find this automatically which is typically fine for most NMR experiments. See `read()` for additional details.

**Returns**

- **dic**: dict
  
  Dictionary of Agilent/Varian parameters.
data : array_like
Low memory object which can access NMR data on demand.

See also:

read  Read Agilent/Varian files.
write_lowmem  Write Agilent/Varian files using minimal memory

nmrglue.fileio.varian.read_fid

nmrglue.fileio.varian.read_fid(filename, shape=None, torder='flat', as_2d=False, read_blockhead=False)
Read a Agilent/Varian binary (fid) file.

Parameters

filename : str
Filename of Agilent/Varian binary file (fid) to read.

shape : tuple of ints, optional
Shape of the binary data. If not provided data is returned as a 2D array. Required if more than one trace per block (non-standard).

torder : {'f', 'n', 'o'}
Trace order. See read() for details.

as_2d : bool, optional
True to return the data as a 2D array, ignoring the shape and torder parameters.

read_blockhead : bool, optional
True to read the Agilent/Varian blockheaders(s) into the returned dictionary. False ignores them.

Returns
dic : dict
Dictionary of Agilent/Varian binary file parameters.

data : ndarray
Array of NMR data.

See also:

read_fid_lowmem  Read a Agilent/Varian binary file using minimal amounts of memory.
read  Read Agilent/Varian files from a directory.

nmrglue.fileio.varian.write_fid

nmrglue.fileio.varian.write_fid(filename, dic, data, torder='flat', repack=False, correct=True, overwrite=False)
Write a Agilent/Varian binary (fid) file.

Parameters

filename : str
Filename of Agilent/Varian binary file to write to.

dic : dict
Dictionary of Agilent/Varian file parameters.

**data**: ndarray

Array of NMR data.

**torder**: {'f', 'r', 'o'}

Trace ordering. See *read()* for details.

**repack**: bool, optional

True to repack file and block headers. False leave as is.

**correct**: bool, optional

True (the default) will correct mis-sized np and nblocks values in dic. False will write out the incorrect values to the file header and record the full data set, the resulting file will not be readable by nmrglue.

**overwrite**: bool, optional

Set True to overwrite an existing file, False will raise a Warning if the file exists.

See also:

- **write_fid_lowmem** Write a Agilent/Varian binary file using mimimal amounts of memory
- **write** Write Agilent/Varian files to a directory.

### nmrglue.fileio.varian.read_fid_lowmem

nmrglue.fileio.varian.*read_fid_lowmem*(*filename*, *shape=None*, *torder='flat'*, *as_2d=False*, *read_blockhead=False*)

Read a Agilent/Varian binary (fid) file using mimimal amounts of memory.

**Parameters**

- **filename**: str
  
  Filename of Agilent/Varian binary file (fid) to read.

- **shape**: tuple of ints, optional
  
  Shape of the binary data. If not provided data is returned as a 2D array. Required if more than one trace per block (non-standard).

- **torder**: {'f', 'n', 'o'}
  
  Trace order. See *read()* for details.

- **read_blockhead**: bool, optional
  
  Not used, retained for compatibility with *read_fid()*

**Returns**

- **dic**: dict
  
  Dictionary of Agilent/Varian binary file parameters.

- **data**: array_like
  
  Low memory object which can access NMR data on demand.

See also:

- **read_fid** Read a Agilent/Varian binary file.
- **read_lowmem** Read Agilent/Varian files from a directory using minimal
nmrglue Documentation, Release 0.7-dev

amounts

nmrglue.fileio.varian.write_fid_lowmem

nmrglue.fileio.varian.write_fid_lowmem(filename, dic, data, torder='f', repack=False, overwrite=False)

Write a Agilent/Varian binary (fid) file using minimal amounts of memory.

File is written trace by trace with each trace read from data before writing to reduce memory usage.

Parameters

filename : str
    Filename of Agilent/Varian binary file to write to.

dic : dict
    Dictionary of Agilent/Varian file parameters.

data : array_like
    Array of NMR data.

torder : {'f', 'r', 'o'}
    Trace ordering. See read() for details.

repack : bool, optional
    True to repack file and block headers. False leave as is.

correct : bool, optional
    True (the default) will correct mis-sized np and nblocks values in dic. False will write out the incorrect values to the file header and record the full data set, the resulting file will not be readable by nmrglue.

overwrite : bool, optional
    Set True to overwrite an existing file, False will raise a Warning if the file exists.

See also:

write_fid  Write a Agilent/Varian binary.
write  Write Agilent/Varian files to a directory.

nmrglue.fileio.varian.read_fid_ntraces

nmrglue.fileio.varian.read_fid_ntraces(filename, shape=None, torder='flat', as_2d=False, read_blockhead=False)

Read a Agilent/Varian binary (fid) file possibility having multiple traces per block.

Parameters

filename : str
    Filename of Agilent/Varian binary file (fid) to read.

shape : tuple of ints, optional
    Shape of the binary data. If not provided data is returned as a 2D array. Required if more than one trace per block (non-standard).

torder : {'f', 'n', 'o'}
    Trace order. See read() for details.
as_2d : bool, optional
    True to return the data as a 2D array, ignoring the shape and torder parameters.

read_blockhead : bool, optional
    True to read the Agilent/Varian blockheaders(s) into the returned dictionary. False ig-
   nores them.

Returns dic : dict
    Dictionary of Agilent/Varian binary file parameters.

data : array_like
    Low memory object which can access NMR data on demand.

See also:

read_fid Read a Agilent/Varian binary file with one trace per block.

read_fid_lowmem Read a Agilent/Varian binary file with one trace per block using minimal amounts of
memory.

nmrglue.fileio.varian.read_procpar

nmrglue.fileio.varian.read_procpar (filename)
    Read a procpar file returning a dictionary of procpar parameters.

nmrglue.fileio.varian.write_procpar

nmrglue.fileio.varian.write_procpar (filename, dic, overwrite=False)
    Write a Agilent/Varian procpar file from a dictionary

nmrglue.fileio.varian.guess_udic

nmrglue.fileio.varian.guess_udic (dic, data)
    Guess parameter of a universal dictionary from dic, data pair.

Parameters dic : dict
    Dictionary of Agilent/Varian parameters.

data : ndarray
    Array of NMR data.

Returns udic : dict
    Universal dictionary of spectral parameters.

nmrglue.fileio.varian.create_dic

nmrglue.fileio.varian.create_dic (udic)
    Create a Agilent/Varian parameter dictionary from a universal dictionary.

Parameters udic : dict
Universal dictionary of spectral parameters.

Returns \texttt{dic : dict}

Dictionary of Agilent/Varian parameters

Developer Information

\_\_developer\_\_doc\_\_ = """" Agilent/Varian file format information

Both the Agilent/Varian binary and parameter formats are documented in:

- VnmrJ User Programming - Chapter 5: Parameters and Data

These are available (as of 04/2011) online from

Developer Functions

These functions are typically not used directly by users. Developers who want fine control over Agilent/Varian files will be interested in these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{create data}(\texttt{data})</td>
<td>Create a Agilent/Varian data array (recast into complex64 array)</td>
</tr>
<tr>
<td>\texttt{create pdic param}(\texttt{name}, \texttt{values})</td>
<td>Create a fake procpar dictionary element of with given name and values.</td>
</tr>
<tr>
<td>\texttt{find torder}(\texttt{dic}, \texttt{shape})</td>
<td>Find the torder from the procpar dictionary.</td>
</tr>
<tr>
<td>\texttt{torder2i2t}(\texttt{torder})</td>
<td>Convert torder to an index2trace function</td>
</tr>
<tr>
<td>\texttt{torder2t2i}(\texttt{torder})</td>
<td>Convert torder to a trace2index functions</td>
</tr>
<tr>
<td>\texttt{reorder data}(\texttt{data}, \texttt{shape}, \texttt{torder})</td>
<td>Reorder raw data from file.</td>
</tr>
<tr>
<td>\texttt{order data}(\texttt{data}, \texttt{torder})</td>
<td>Order NMR data for writing to file.</td>
</tr>
<tr>
<td>\texttt{get nbblocks}(\texttt{f}, \texttt{nblocks}, \texttt{pts}, \texttt{nbheaders}, \texttt{dt}, ...)</td>
<td>Read multiple blocks from a Agilent/Varian binary file.</td>
</tr>
<tr>
<td>\texttt{get block}(\texttt{f}, \texttt{pts}, \texttt{nbheaders}, \texttt{dt}[,...])</td>
<td>Read a single block from Agilent/Varian binary file.</td>
</tr>
<tr>
<td>\texttt{get nbblocks ntraces}(\texttt{f}, \texttt{nblocks}, \texttt{ntraces}, ...)</td>
<td>Read multiple blocks from a Agilent/Varian binary file which may have multiple traces per block.</td>
</tr>
<tr>
<td>\texttt{get block ntraces}(\texttt{f}, \texttt{ntraces}, \texttt{pts}, \texttt{nbheaders}, \texttt{dt})</td>
<td>Read a single block from Agilent/Varian binary file which may have multiple traces per block.</td>
</tr>
<tr>
<td>\texttt{get trace}(\texttt{f}, \texttt{pts}, \texttt{dt})</td>
<td>Read trace of pts points of dtype dt from Agilent/Varian binary file.</td>
</tr>
<tr>
<td>\texttt{get fileheader}(\texttt{f})</td>
<td>Unpack file header parameters into a list.</td>
</tr>
<tr>
<td>\texttt{get blockheader}(\texttt{f})</td>
<td>Unpack block header parameters into a list.</td>
</tr>
<tr>
<td>\texttt{skip blockheader}(\texttt{f})</td>
<td>Read a block header but do not unpack.</td>
</tr>
<tr>
<td>\texttt{get hyperheader}(\texttt{file})</td>
<td>Unpack hypercomplex header parameters to a list.</td>
</tr>
<tr>
<td>\texttt{put block}(\texttt{f}, \texttt{trace}, \texttt{nbheaders}, \texttt{bh}[, \texttt{hh}])</td>
<td>Put blockheader(s) and the trace to file.</td>
</tr>
<tr>
<td>\texttt{put trace}(\texttt{f}, \texttt{trace})</td>
<td>Write a trace to file f.</td>
</tr>
<tr>
<td>\texttt{put fileheader}(\texttt{f}, \texttt{fh})</td>
<td>Write a fileheader list to file (32-bytes written).</td>
</tr>
<tr>
<td>\texttt{put blockheader}(\texttt{f}, \texttt{bh})</td>
<td>Write a blockheader list to file (28-bytes written)</td>
</tr>
<tr>
<td>\texttt{put hyperheader}(\texttt{f}, \texttt{hh})</td>
<td>Write hyperheader list to file (28-bytes written)</td>
</tr>
<tr>
<td>\texttt{hyperheader2dic}(\texttt{head})</td>
<td>Convert a hypercomplex block header into a Python dictionary.</td>
</tr>
<tr>
<td>\texttt{repack hyperheader}(\texttt{dic})</td>
<td>Repack a hyperheader dictionary bit flag parameters into status.</td>
</tr>
</tbody>
</table>
Table 3.36 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dic2hyperheader(dic)</code></td>
<td>Convert a Python dictionary into a hypercomplex block header list.</td>
</tr>
<tr>
<td><code>make_blockheader([filedic, index])</code></td>
<td>Make a generic blockheader dictionary with a given block index.</td>
</tr>
<tr>
<td><code>blockheader2dic(head)</code></td>
<td>Convert a block header list into a Python dictionary.</td>
</tr>
<tr>
<td><code>repack_blockheader(dic)</code></td>
<td>Repack blockheader dic bit flag parameters into status and mode.</td>
</tr>
<tr>
<td><code>dic2blockheader(dic)</code></td>
<td>Convert a python dictionary into block header list.</td>
</tr>
<tr>
<td><code>fileheader2dic(head)</code></td>
<td>Convert fileheader list into a Python dictionary</td>
</tr>
<tr>
<td><code>repack_fileheader(dic)</code></td>
<td>Repack blockheader dic bit flag parameters into status and mode.</td>
</tr>
<tr>
<td><code>dic2fileheader(dic)</code></td>
<td>Convert a Python dictionary into a fileheader list</td>
</tr>
<tr>
<td><code>find_shape(pdic)</code></td>
<td>Determine the shape of a Agilent/Varian file from the procpar dictionary.</td>
</tr>
<tr>
<td><code>find_cdtype(dic)</code></td>
<td>Find the complex dtype from a Agilent/Varian dictionary</td>
</tr>
<tr>
<td><code>find_dtype(dic)</code></td>
<td>Find the real dtype from a dictionary</td>
</tr>
<tr>
<td><code>uninterleave_data(data)</code></td>
<td>Unpack interleaved real, imag data</td>
</tr>
<tr>
<td><code>interleave_data(data_in)</code></td>
<td>Interleave real, imag data</td>
</tr>
<tr>
<td><code>get_parameter(f)</code></td>
<td>Reads a procpar parameter from a file object.</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.varian.create_data**

`nmrglue.fileio.varian.create_data(data)`

Create a Agilent/Varian data array (recast into complex64 array)

**nmrglue.fileio.varian.create_pdic_param**

`nmrglue.fileio.varian.create_pdic_param(name, values)`

Create a fake procpar dictionary element of with given name and values.

**nmrglue.fileio.varian.find_torder**

`nmrglue.fileio.varian.find_torder(dic, shape)`

Find the torder from the procpar dictionary.

If procpar dictionary is incomplete a UserWarning is issued and ‘r’ is returned.

**Parameters**

- `dic : dict`
  
  Dictionary of parameters in the procpar file.

- `shape : tuple of ints`
  
  Shape of NMR data.

**Returns**

- `torder : {‘r’, ‘f’, ‘o’}`
  
  File ording for using in `read()` or `write()`.

3.1. fileio modules 95
nmrglue.fileio.varian.torder2i2t

nmrglue.fileio.varian.torder2i2t(torder)
Convert torder to an index2trace function

nmrglue.fileio.varian.torder2t2i

nmrglue.fileio.varian.torder2t2i(torder)
Convert torder to a trace2index functions

nmrglue.fileio.varian.reorder_data

nmrglue.fileio.varian.reorder_data(data, shape, torder)
Reorder raw data from file.

Parameters

- **data**: 2D ndarray
  Raw data as ordered in binary file.
- **shape**: tuple of ints
  Shape of the NMR data.
- **torder**: {'f', 'r', 'o'} of Python function
  Trace ordering. See `read()` for details.

Returns

- **tdata**: ndarray
  Array of NMR data.

Notes

Minimal error checking is done to see if data and shape contain the same number of values. This should be done before calling this function.

nmrglue.fileio.varian.order_data

nmrglue.fileio.varian.order_data(data, torder)
Order NMR data for writing to file.

Parameters

- **data**: ndarray
  Array of NMR data.
- **torder**: {'f', 'r', 'o'}
  Trace ordering. See `read()` for details.

Returns

- **raw_data**: 2D ndarray
  2D array ordered for writing to Agilent/Varian binary file.
nmrglue.fileio.varian.get_nblocks

nmrglue.fileio.varian.get_nblocks(f, nbblocks, pts, nbheaders, dt, read_blockhead)
Read multiple blocks from a Agilent/Varian binary file.

Parameters:
- f : file object
  File object of open Agilent/Varian binary file.
- nbblock : int
  Number of blocks to read.
- pts : int
  Number of points per trace.
- nbheaders : int
  Number of block header in each block.
- dt : dtype
  Data type of data in binary file (real type).
- read_blockheaders : bool
  True to read the blockheader(s) and return them in the returned dictionary. False ignores blockheader, reading over them.

Returns:
- dic : dict, optional
  Dictionary of blockheaders, only returned if read_blockheaders is True.
- data : 2D ndarray of size (nbblocks, pts)
  Array of read blocks.

nmrglue.fileio.varian.get_block

nmrglue.fileio.varian.get_block(f, pts, nbheaders, dt, read_blockhead=False)
Read a single block from Agilent/Varian binary file.

Parameters:
- f : file object
  File object of open Agilent/Varian binary file.
- pts : int
  Number of points per trace.
- nbheaders : int
  Number of block header in each block.
- dt : dtype
  Data type of data in binary file (real type).
- read_blockheaders : bool, optional.
  True to read the blockheader(s) and return them in the returned dictionary. False (default) ignores blockheader, reading over them.

Returns:
- dic : dict, optional
  Dictionary of blockheaders, only returned if read_blockheaders is True.
data : ndarray of size
    Array of read block.

nmrglue.fileio.varian.get_nblocks_ntraces

nmrglue.fileio.varian.get_nblocks_ntraces(f, nblocks, ntraces, pts, nbheaders, dt, read_blockhead)
    Read multiple blocks from a Agilent/Varian binary file which may have multiple traces per block.

Parameters f : file object
    File object of open Agilent/Varian binary file.

nbloc k : int
    Number of blocks to read.

ntrace : int
    Number of traces per block

pts : int
    Number of points per trace.

nbheaders : int
    Number of block header in each block.

dt : dtype
    Data type of data in binary file (real type).

read_blockheaders : bool
    True to read the blockheader(s) and return them in the returned dictionary. False ignores blockheader, reading over them.

Returns dic : dict, optional
    Dictionary of blockheaders, only returned if read_blockheaders is True.

data : 2D ndarray of size (nbloc k * ntraces, pts)
    Array of read blocks.

nmrglue.fileio.varian.get_block_ntraces

nmrglue.fileio.varian.get_block_ntraces(f, ntraces, pts, nbheaders, dt, read_blockhead=False)
    Read a single block from Agilent/Varian binary file which may have multiple traces per block.

Parameters f : file object
    File object of open Agilent/Varian binary file.

ntrace : int
    Number of traces per block

pts : int
    Number of points per trace.
nbheaders : int
    Number of block header in each block.

dt : dtype
    Data type of data in binary file (real type).

read_blockheaders : bool
    True to read the blockheader(s) and return them in the returned dictionary. False ignores blockheader, reading over them.

Returns
dic : dict, optional
    Dictionary of blockheaders, only returned if read_blockheaders is True.

data : 2D ndarray of shape (ntraces, pts)
    Array of read blocks.

nmrglue.fileio.varian.get_trace

nmrglue.fileio.varian.get_trace(f, pts, dt)
    Read trace of pts points of dtype dt from Agilent/Varian binary file
    Endiness should be handled by dt.

nmrglue.fileio.varian.get_fileheader

nmrglue.fileio.varian.get_fileheader(f)
    Unpack file header parameters into a list.
    Reads the 32-byte file header from file and unpacks into a list. Endiness is corrected as needed.

Returned list contents:

<table>
<thead>
<tr>
<th>N</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nblocks</td>
<td>data blocks in file</td>
</tr>
<tr>
<td>1</td>
<td>ntraces</td>
<td>traces per block</td>
</tr>
<tr>
<td>2</td>
<td>np</td>
<td>elements per trace</td>
</tr>
<tr>
<td>3</td>
<td>ebytes</td>
<td>bytes per element</td>
</tr>
<tr>
<td>4</td>
<td>tbytes</td>
<td>bytes per trace</td>
</tr>
<tr>
<td>5</td>
<td>bbytes</td>
<td>bytes per block</td>
</tr>
<tr>
<td>6</td>
<td>vers_id</td>
<td>software version, file_id status bits</td>
</tr>
<tr>
<td>7</td>
<td>status</td>
<td>status of whole file</td>
</tr>
<tr>
<td>8</td>
<td>nbheaders</td>
<td>number of block headers per block (1)</td>
</tr>
</tbody>
</table>

nmrglue.fileio.varian.get_blockheader

nmrglue.fileio.varian.get_blockheader(f)
    Unpack block header parameters into a list.
    Reads the 28-byte block header from f and unpacks into a list. Endiness is corrected as needed.

Returned list contents:
<table>
<thead>
<tr>
<th>N</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>scale</td>
<td>scaling factor</td>
</tr>
<tr>
<td>1</td>
<td>status</td>
<td>status of data in block</td>
</tr>
<tr>
<td>2</td>
<td>index</td>
<td>block index</td>
</tr>
<tr>
<td>3</td>
<td>mode</td>
<td>block mode</td>
</tr>
<tr>
<td>4</td>
<td>ctcount</td>
<td>ct value of FID</td>
</tr>
<tr>
<td>5</td>
<td>lpval</td>
<td>left phase</td>
</tr>
<tr>
<td>6</td>
<td>rpval</td>
<td>right phase</td>
</tr>
<tr>
<td>7</td>
<td>lvl</td>
<td>level drift correction</td>
</tr>
<tr>
<td>8</td>
<td>tlt</td>
<td>tilt drift correction</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.varian.skip_blockheader**

**nmrglue.fileio.varian.skip_blockheader**(*f*)

Read a block header but do not unpack.

This is a replacement for get_blockheader. It skips *f* ahead 28 bytes.

**nmrglue.fileio.varian.get_hyperheader**

**nmrglue.fileio.varian.get_hyperheader**(file)

Unpack hypercomplex header parameters to a list.

Reads the 28-bytes block header from file and unpacks into a list. Endiness is corrected as needed.

Returned list contents:

<table>
<thead>
<tr>
<th>N</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s_spar1</td>
<td>Not Used</td>
</tr>
<tr>
<td>1</td>
<td>status</td>
<td>status of block</td>
</tr>
<tr>
<td>2</td>
<td>s_spar2</td>
<td>Not Used</td>
</tr>
<tr>
<td>3</td>
<td>s_spar3</td>
<td>Not Used</td>
</tr>
<tr>
<td>4</td>
<td>l_spar1</td>
<td>Not Used</td>
</tr>
<tr>
<td>5</td>
<td>lpval1</td>
<td>2D left phase</td>
</tr>
<tr>
<td>6</td>
<td>rpval1</td>
<td>2D right phase</td>
</tr>
<tr>
<td>7</td>
<td>f_spar1</td>
<td>Not Used</td>
</tr>
<tr>
<td>8</td>
<td>f_spar2</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

**nmrglue.fileio.varian.put_block**

**nmrglue.fileio.varian.put_block**(f, trace, nbheaders, bh=False)

Put blockheader(s) and the trace to file.

**Parameters**

- f: file object
  
  Open file object to write to.

- trace: ndarray
  
  Trace to write to current block.

- nbheaders: int
  
  Number of block headers.

- bh: list
  
  List of block headers to write.
Blockheader list.

\texttt{hh} : list, optional

Hyperheader list. Required when \texttt{nbheaders} $\geq 2$.

**Notes**

When \texttt{nbheaders} $> 2$, additional headers are written as all zeros.

\texttt{nmrglue.fileio.varian.put_trace}

\texttt{nmrglue.fileio.varian.put_trace}(f, \textit{trace})

Write a trace to file \texttt{f}.

\texttt{nmrglue.fileio.varian.put_fileheader}

\texttt{nmrglue.fileio.varian.put_fileheader}(\textit{f}, \textit{fh})

Write a fileheader list to file (32-bytes written).

\textbf{Parameters}

\texttt{f} : file object

Open file object to write to.

\texttt{fh} : list with 9 elements

Fileheader list.

\texttt{nmrglue.fileio.varian.put_blockheader}

\texttt{nmrglue.fileio.varian.put_blockheader}(\textit{f}, \textit{bh})

Write a blockheader list to file (28-bytes written)

\textbf{Parameters}

\texttt{f} : file object

Open file object to write to.

\texttt{bh} : list with 9 elements

Blockheaders list.

\texttt{nmrglue.fileio.varian.put_hyperheader}

\texttt{nmrglue.fileio.varian.put_hyperheader}(\textit{f}, \textit{hh})

Write hyperheader list to file (28-bytes written)

\textbf{Parameters}

\texttt{f} : file object

Open file object to write to.

\texttt{hh} : list with 9 elements

Hyperheader list.
nmrglue.fileio.varian.hyperheader2dic

```
mrglue.fileio.varian.hyperheader2dic(head)
    Convert a hypercomplex block header into a Python dictionary.
```

nmrglue.fileio.varian.repack_hyperheader

```
mrglue.fileio.varian.repack_hyperheader(dic)
    Repack a hyperheader dictionary bit flag parameters into status.
```

nmrglue.fileio.varian.dic2hyperheader

```
mrglue.fileio.varian.dic2hyperheader(dic)
    Convert a Python dictionary into a hypercomplex block header list.
    Does not repack status from bit flags.
```

nmrglue.fileio.varian.make_blockheader

```
mrglue.fileio.varian.make_blockheader(filedic=False, index=1)
    Make a generic blockheader dictionary with a given block index.
    filedic can be provided for status flags, if not provided creates header for float32 data
```

nmrglue.fileio.varian.blockheader2dic

```
mrglue.fileio.varian.blockheader2dic(head)
    Convert a block header list into a Python dictionary.
```

nmrglue.fileio.varian.repack_blockheader

```
mrglue.fileio.varian.repack_blockheader(dic)
    Repack blockheader dic bit flag parameters into status and mode.
```

nmrglue.fileio.varian.dic2blockheader

```
mrglue.fileio.varian.dic2blockheader(dic)
    Convert a python dictionary into block header list.
    Does not repack status and mode from bit flags.
```

nmrglue.fileio.varian.fileheader2dic

```
mrglue.fileio.varian.fileheader2dic(head)
    Convert fileheader list into a Python dictionary
```
nmrglue.fileio.varian.repack_fileheader

nmrglue.fileio.varian.repack_fileheader(dic)
   Repack blockheader dic bit flag parameters into status and mode.

nmrglue.fileio.varian.dic2fileheader

nmrglue.fileio.varian.dic2fileheader(dic)
   Convert a Python dictionary into a fileheader list
   Does not repack status from bit flags

nmrglue.fileio.varian.find_shape

nmrglue.fileio.varian.find_shape(pdic)
   Determine the shape of a Agilent/Varian file from the procpar dictionary

nmrglue.fileio.varian.find_cdtype

nmrglue.fileio.varian.find_cdtype(dic)
   Find the complex dtype from a Agilent/Varian dictionary

nmrglue.fileio.varian.find_dtype

nmrglue.fileio.varian.find_dtype(dic)
   Find the real dtype from a dictionary

nmrglue.fileio.varian.uninterleave_data

nmrglue.fileio.varian.uninterleave_data(data)
   Unpack interleaved real, imag data

<table>
<thead>
<tr>
<th>data dtype</th>
<th>Return dtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>int16</td>
<td>'complex64'</td>
</tr>
<tr>
<td>float32</td>
<td>'complex64'</td>
</tr>
<tr>
<td>int32</td>
<td>'complex128'</td>
</tr>
</tbody>
</table>

nmrglue.fileio.varian.interleave_data

nmrglue.fileio.varian.interleave_data(data_in)
   Interleave real, imag data
   Does not check if resulting dtype is a valid Agilent/Varian dtype
nmrglue Documentation, Release 0.7-dev

nmrglue.fileio.varian.get_parameter

nmrglue.fileio.varian.get_parameter(f)
   Reads a procpar parameter from a file object.
   Returns a dictionary with the attributes of the parameter.

Developer Classes

fid_nd(filename, i2t_func[, fshape, order])
   Emulate a ndarray objects without loading data into memory for low memory reading of Agilent/Varian fid files which must have one trace per block.

- slicing operations return ndarray objects.
- can iterate over with expected results.
- transpose and swapaxes methods create a new objects with correct axes ordering.
- has ndim, shape, and dtype attributes.

Parameters

- filename : str
  Filename of Agilent/Varian binary file.

- i2t_func : function
  Python function which maps an index to a trace.

- fshape : tuple of ints, optional
  Shape of data in file, if None will be assumed to be 2D data.

- order : tuple
  Ordering of axes compared to file. None will results in (0, 1, 2, ...) ordering.

__init__(filename, i2t_func, fshape=None, order=None)
   Create and set up object.

Methods

- __init__(filename, i2t_func[, fshape, order])
  Create and set up object.

- swapaxes(axis1, axis2)
  Return object with axis1 and axis2 interchanged.

- transpose(*axes)
  Return object with axes transposed.
3.2 process modules

3.2.1 nmrglue.proc_autophase

Automated phase correction These functions provide support for automatic phasing of NMR data. They consist of the core *autops* function which performs the optimisation and a set of private functions for calculating a spectral phase quality score for a provided spectrum.

This module is imported as nmrglue.proc_autophase and can be called as such.

User Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>autops</strong></td>
<td>Automatic linear phase correction</td>
</tr>
<tr>
<td><strong>manual_ps</strong></td>
<td>Manual Phase correction using matplotlib</td>
</tr>
</tbody>
</table>

**nmrglue.process.proc_autophase.autops**

*nmrglue.process.proc_autophase.*autops*(data, fn[, p0, p1])

Automatic linear phase correction

**Parameters**

*data*: ndarray

Array of NMR data.

*fn*: str or function

Algorithm to use for phase scoring. Built in functions can be specified by one of the following strings: “acme”, “peak_minima”

*p0*: float

Initial zero order phase in degrees.

*p1*: float

Initial first order phase in degrees.

**Returns**

*ndata*: ndarray

Phased NMR data.

**nmrglue.process.proc_autophase.manual_ps**

*nmrglue.process.proc_autophase.*manual_ps*(data[, notebook])

Manual Phase correction using matplotlib

A matplotlib widget is used to manually correct the phase of a Fourier transformed dataset. If the dataset has more than 1 dimensions, the first trace will be picked up for phase correction. Clicking the ‘Set Phase’ button will print the current linear phase parameters to the console. A ipywidget is provided for use with Jupyter Notebook to avoid changing backends. This can be accessed with notebook=True option in this function

**Note:** Needs matplotlib with an interactive backend.

**Parameters**

*data*: ndarray
Array of NMR data.

**notebook** : Bool


**Returns** **p0, p1** : float

Linear phase correction parameters. Zero and first order phase corrections in degrees calculated from pc0, pc1 and pivot displayed in the interactive window.

**Examples**

```python
>>> import nmrglue as ng
>>> p0, p1 = ng.process.proc_autophase.manual_ps(data)
>>> # do manual phase correction and close window
>>> phased_data = ng.proc_base.ps(data, p0=p0, p1=p1)
```

In [1] # if you are using the Jupyter Notebook In [2] ng.process.proc_autophase.manual_ps(data) Out [2] # do manual phase correction. p0 and p1 values will be updated

# continuously as you do so and are printed below the plot

In [3] phased_data = ng.proc_base.ps(data, p0=p0, p1=p1)

**Developer Functions**

These functions are called by high-level function are and most users will not use them in common processing scripts. Developers may be interested in them.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ps_acme_score(ph, data)</td>
<td>Phase correction using ACME algorithm by Chen Li et al.</td>
</tr>
<tr>
<td>_ps_peak_minima_score(ph, data)</td>
<td>Phase correction using simple minima-minimisation around highest peak</td>
</tr>
</tbody>
</table>

**nmrglue.process.proc_autophase._ps_acme_score**

**nmrglue.process.proc_autophase._ps_acme_score(ph, data)**


**Parameters** **pd** : tuple

Current p0 and p1 values

**data** : ndarray

Array of NMR data.

**Returns** **score** : float

Value of the objective function (phase score)

**nmrglue.process.proc_autophase._ps_peak_minima_score**

**nmrglue.process.proc_autophase._ps_peak_minima_score(ph, data)**

Phase correction using simple minima-minimisation around highest peak
This is a naive approach but is quick and often achieves reasonable results. The optimisation is performed by finding the highest peak in the spectra (e.g. TMSP) and then attempting to reduce minima surrounding it.

**Parameters**

- **pd**: tuple
  - Current p0 and p1 values

- **data**: ndarray
  - Array of NMR data.

**Returns**

- **score**: float
  - Value of the objective function (phase score)

### 3.2.2 nmrglue.proc_base

A collection of NMR spectral processing functions which operate on the last dimension (1) of 2D arrays. These functions are wrapped by other processing modules but can also be used directly. All parameter are assumed to be in units of points unless otherwise noted.

This module is imported as nmrglue.proc_base and can be called as such.

**Apodization**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>em</td>
<td>Exponential apodization</td>
</tr>
<tr>
<td>gp</td>
<td>Lorentz-to-Gauss apodization</td>
</tr>
<tr>
<td>gmb</td>
<td>Modified gaussian apodization</td>
</tr>
<tr>
<td>jmod</td>
<td>Exponentially damped J-modulation apodization</td>
</tr>
<tr>
<td>sp</td>
<td>Shifted sine-bell apodization</td>
</tr>
<tr>
<td>sine</td>
<td>Shifted sine-bell apodization</td>
</tr>
<tr>
<td>tm</td>
<td>Trapezoid Apodization</td>
</tr>
<tr>
<td>tri</td>
<td>Triangle apodization</td>
</tr>
</tbody>
</table>

**nmrglue.process.proc_base.em**

**nmrglue.process.proc_base.em**(data[, lb, inv, rev])

Exponential apodization

Functional form of apodization window:

\[ em(x_i) = \exp(-\pi i * lb) \]

**Parameters**

- **data**: ndarray
  - Array of NMR data.

- **lb**: float
  - Exponential line broadening, in units of points. To apply a similar apodization as NMRPipe’s EM function, use \( lb = lb_{hz} / sw_{hz} \), where \( lb_{hz} \) is the amount of broadening to apply in Hz and \( sw_{hz} \) is the spectral width of the last dimension in Hz.

- **inv**: bool, optional
  - True for inverse apodization. False (default) for standard.

- **rev**: bool, optional.
True to reverse the apodization before applying it to the data.

**Returns**  
ndata : ndarray  
Array of NMR data with apodization applied.

**nmrglue.process.proc_base.gm**

```python
c痄glue.process.proc_base.gm(data, g1=0.0, g2=0.0, g3=0.0, inv=False, rev=False)
```

Lorentz-to-Gauss apodization  
Functional form of apodization window:

\[ g(x_i) = \exp(e - g^2) \]

Where:

\[ e = \pi i * g1 \]
\[ g = 0.6 * \pi i * g2 * (g3 * (size - 1) - i) \]

**Parameters**  
data : ndarray  
Array of NMR data.  
g1 : float  
Inverse exponential width.  
g2 : float  
Gaussian broadening width.  
g3 : float  
Location of Gaussian maximum.  
inv : bool, optional  
True for inverse apodization. False (default) for standard.  
rev : bool, optional  
True to reverse the apodization before applying it to the data.

**Returns**  
data : ndarray  
Array of NMR data with apodization applied.

**nmrglue.process.proc_base.gmb**

```python
c痄glue.process.proc_base.gmb(data, a=0.0, b=0.0, inv=False, rev=False)
```

Modified gaussian apodization  
Functional form of apodization window:

\[ gmb(x_i) = \exp(-a * i - b * i^2) \]

**Parameters**  
data : ndarray  
Array of NMR data.  
a : float
Exponential term in apodization.

\[ b : \text{float} \]

Gaussian term in apodization.

\[ \text{inv} : \text{bool, optional} \]

True for inverse apodization. False (default) for standard.

\[ \text{rev} : \text{bool, optional}. \]

True to reverse the apodization before applying it to the data.

**Returns**  
\[ \text{ndata} : \text{ndarray} \]

Array of NMR data with apodization applied.

---

\[ \text{nmrglue.process.proc_base.jmod} \]

\[ \text{nmrglue.process.proc_base.jmod(data, e=0.0, off=0.0, end=0.0, inv=False, rev=False)} \]

Exponentially damped J-modulation apodization

 Functional form of apodization window:

\[
jmod(x_i) = \exp(-e) \times \sin\left(\frac{\pi \times \text{off} + \pi \times (\text{end} - \text{off}) \times i}{\text{size} - 1}\right)
\]

**Parameters**  
\[ \text{data} : \text{ndarray} \]

Array of NMR data.

\[ e : \text{float} \]

Exponential apodization term.

\[ \text{off} : \text{float} \]

Start of J-modulation in fractions of pi radians (180 degrees).

\[ \text{end} : \text{float} \]

End of J-modulation in fractions of pi radians (180 degrees).

\[ \text{inv} : \text{bool, optional} \]

True for inverse apodization. False (default) for standard.

\[ \text{rev} : \text{bool, optional}. \]

True to reverse the apodization before applying it to the data.

**Returns**  
\[ \text{ndata} : \text{ndarray} \]

Array of NMR data with apodization applied.

---

\[ \text{nmrglue.process.proc_base.sp} \]

\[ \text{nmrglue.process.proc_base.sp(data, off=0, end=1.0, pow=1.0, inv=False, rev=False)} \]

Shifted sine-bell apodization

 Functional form of apodization window:

\[
sp(x_i) = \sin\left(\frac{\pi \times \text{off} + \pi \times (\text{end} - \text{off}) \times i}{\text{size} - 1}\right)^\text{pow}
\]

---

**3.2. process modules**
Parameters data : ndarray
    Array of NMR data.

off : float
    Offset start of sine-bell as a percentage of the trace (0 -> 1).

end : float
    End of of sine-bell as a percentage of the trace (0 -> 1).

pow : float
    Power to raise sine-bell to.

inv : bool, optional
    True for inverse apodization. False (default) for standard.

rev : bool, optional.
    True to reverse the apodization before applying it to the data.

Returns ndata : ndarray
    Array of NMR data with apodization applied.

nmrglue.process.proc_base.sine

nmrglue.process.proc_base.sine(data, off=0, end=1.0, pow=1.0, inv=False, rev=False)
    Shifted sine-bell apodization
    Functional form of apodization window:
    \[ sp(x_i) = \sin\left(\frac{\pi * off + \pi * (end - off) * i}{size - 1}\right)^{pow} \]

Parameters data : ndarray
    Array of NMR data.

off : float
    Offset start of sine-bell as a percentage of the trace (0 -> 1).

end : float
    End of of sine-bell as a percentage of the trace (0 -> 1).

pow : float
    Power to raise sine-bell to.

inv : bool, optional
    True for inverse apodization. False (default) for standard.

rev : bool, optional.
    True to reverse the apodization before applying it to the data.

Returns ndata : ndarray
    Array of NMR data with apodization applied.
**nmrglue.process.proc_base.tm**

**nmrglue.process.proc_base.tm**(*data*, *t1*=<0.0>, *t2*=<0.0>, *inv*=*False*, *rev*=*False*)

Trapezoid Apodization

Functional form of apodization:

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:*t1</td>
<td>Linear increases from 0.0 to 1.0</td>
</tr>
<tr>
<td><em>t1</em>:size-*t2</td>
<td>Flat with value of 1.0</td>
</tr>
<tr>
<td>-<em>t2</em>:</td>
<td>Linear decrease from 1.0 to 0.0</td>
</tr>
</tbody>
</table>

**Parameters**
- **data** : ndarray
  - Array of NMR data.
- **t1** : int
  - Length of left ramp in points.
- **t2** : int
  - Length of right ramp in points.
- **inv** : bool, optional
  - True for inverse apodization. False (default) for standard.
- **rev** : bool, optional.
  - True to reverse the apodization before applying it to the data.

**Returns**
- **ndata** : ndarray
  - Array of NMR data with apodization applied.

**nmrglue.process.proc_base.tri**

**nmrglue.process.proc_base.tri**(*data*, *loc*='auto', *lHi*=0.0, *rHi*=0.0, *inv*=*False*, *rev*=*False*)

Triangle apodization.

Functional form of apodization window:

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:*loc</td>
<td>Linear increase/decrease from lHi to 1.0</td>
</tr>
<tr>
<td><em>loc</em>:</td>
<td>Linear increase/decrease from 1.0 to rHi</td>
</tr>
</tbody>
</table>

**Parameters**
- **data** : ndarray
  - Array of NMR data.
- **loc** : int or “auto”
  - Location of triangle apex. ‘auto’ set the apex to the middle of the trace (default).
- **lHi** : float
  - Starting height of the left side of the triangle.
- **rHi** : float
  - Starting height of the right side of the triangle.
- **inv** : bool, optional
True for inverse apodization. False (default) for standard.

**rev** : bool, optional.

True to reverse the apodization before applying it to the data.

**Returns ndata : ndarray**

Array of NMR data with apodization applied.

### Shifts

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rs(data[, pts])</code></td>
<td>Right shift and zero fill.</td>
</tr>
<tr>
<td><code>ls(data[, pts])</code></td>
<td>Left shift and fill with zero</td>
</tr>
<tr>
<td><code>cs(data[, pts, neg])</code></td>
<td>Circular shift</td>
</tr>
<tr>
<td><code>roll(data[, pts, neg])</code></td>
<td>Roll axis</td>
</tr>
<tr>
<td><code>fsh(data, pts)</code></td>
<td>Frequency shift by Fourier transform.</td>
</tr>
</tbody>
</table>

#### nmrglue.process.proc_base.rs

**nmrglue.process.proc_base.rs (data, pts=0.0)**

Right shift and zero fill.

**Parameters data :**

Array of NMR data.

**pts :** int

Number of points to right shift.

**Returns ndata : ndarray**

Array of NMR data right shifted and zero filled.

**See also:**

*roll* shift without zero filling.

#### nmrglue.process.proc_base.ls

**nmrglue.process.proc_base.ls (data, pts=0.0)**

Left shift and fill with zero.

**Parameters data :**

Array of NMR data.

**pts :** int

Number of points to left shift.

**Returns ndata : ndarray**

Array of NMR data left shifted and zero filled.

**See also:**

*roll* shift without zero filling.
**nmrglue.process.proc_base.cs**

Circular shift

*Parameters*

- **data**: ndarray
  
  Array of NMR data.

- **pts**: int
  
  Number of points to shift. Positive value will right shift the data, negative values will left shift the data.

- **neg**: bool
  
  True to negate the shifted points.

*Returns*

- **ndata**: ndarray
  
  Array of shifted NMR data.

**nmrglue.process.proc_base.roll**

Roll axis

*Parameters*

- **data**: ndarray
  
  Array of NMR data.

- **pts**: int
  
  Number of points to shift. Positive value will right shift the data, negative values will left shift the data.

- **neg**: bool
  
  True to negate the shifted points.

*Returns*

- **ndata**: ndarray
  
  Array of NMR data with last axis rolled.

**nmrglue.process.proc_base.fsh**

Frequency shift by Fourier transform. Negative signed phase correction.

*Parameters*

- **data**: ndarray
  
  Array of NMR data.

- **pts**: float
  
  Number of points to frequency shift the data. Positive value will shift the spectrum to the right, negative values to the left.

*Returns*

- **ndata**: ndarray
  
  Array of NMR data with last axis rolled.
Transforms

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rft(x)</td>
<td>Real Fourier transform.</td>
</tr>
<tr>
<td>irft(xp)</td>
<td>Inverse real fourier transform.</td>
</tr>
<tr>
<td>fft(data)</td>
<td>Fourier transform, NMR ordering of results.</td>
</tr>
<tr>
<td>fft_norm(data)</td>
<td>Fourier transform, total power preserved, NMR ordering of results.</td>
</tr>
<tr>
<td>fft_positive(data)</td>
<td>Fourier transform with positive exponential, NMR ordering of results.</td>
</tr>
<tr>
<td>ifft(data)</td>
<td>Inverse fourier transform, NMR ordering of results.</td>
</tr>
<tr>
<td>ifft_norm(data)</td>
<td>Inverse fourier transform, total power preserved, NMR ordering of results.</td>
</tr>
<tr>
<td>ifft_positive(data)</td>
<td>Inverse fourier transform with positive exponential, NMR ordering of results.</td>
</tr>
<tr>
<td>ha(data)</td>
<td>Hadamard Transform</td>
</tr>
<tr>
<td>ht(data, N)</td>
<td>Hilbert transform.</td>
</tr>
</tbody>
</table>

**nmrglue.process.proc_base.rft**

```python
nmrglue.process.proc_base.rft(x)
```

Real Fourier transform.

**Parameters**

- x : ndarray
  
  Array of NMR data.

**Returns**

- y : ndarray
  
  Array of NMR data with real Fourier transform applied.

**nmrglue.process.proc_base.irft**

```python
nmrglue.process.proc_base.irft(xp)
```

Inverse real fourier transform

**Parameters**

- x : ndarray
  
  Array of NMR data.

**Returns**

- y : ndarray
  
  Array of NMR data with an inverse real Fourier transform applied.

**nmrglue.process.proc_base.fft**

```python
nmrglue.process.proc_base.fft(data)
```

Fourier transform, NMR ordering of results.

There are a number of definitions of the discrete Fourier transform the version used in this function is as follows.

\[ A_k = \sum_{m=0}^{n-1} a_m \exp \left\{ -2\pi i \frac{mk}{n} \right\} \quad k = 0, \ldots, n - 1. \]
With the inverse DFT in the \texttt{ifft()} function defined as follows.

\[
a_m = \frac{1}{n} \sum_{k=0}^{n-1} A_k \exp \left\{ 2\pi i \frac{mk}{n} \right\}, \quad n = 0, \ldots, n - 1.
\]

Two alternative definitions are also supported by \texttt{nmrglue}. One in which both the sum in the \texttt{fft} and \texttt{ifft} are multiplied by \(\frac{1}{\sqrt{n}}\) which results in a pair of transforms in which the total power contained in the the signals before and after the transforms are equal. This is the type transforms used in the Rowland NMR Toolkit. This type of transform is performed by the \texttt{fft_norm()} and \texttt{ifft_norm()} functions.

The second definition changes the sign of the exponent to be positive while keeping the normalization factors the same. This type of transform is performed by the NMRPipe processing package and the functions \texttt{fft_positive()} and \texttt{ifft_positive()}.

All of the Fourier transforms performed by \texttt{nmrglue} return results in ‘NMR order’, in which the two half of the spectrum have been swapped and reversed.

\begin{description}
  \item[Parameters data:] \texttt{ndarray}
  Array of NMR data.
\end{description}

\begin{description}
  \item[Returns ndata:] \texttt{ndarray}
  Fourier transform of NMR data in ‘NMR order’.
\end{description}

\texttt{nmrglue.process.proc_base.fft_norm}

\texttt{nmrglue.process.proc_base.fft_norm(data)}
Fourier transform, total power preserved, NMR ordering of results

This function is similar to the transform performed by The Rowland NMR Toolkit’s FFT function.

See \texttt{fft()} for documentation of the transformation applied by this function.

\begin{description}
  \item[Parameters data:] \texttt{ndarray}
  Array of NMR data.
\end{description}

\begin{description}
  \item[Returns ndata:] \texttt{ndarray}
  Fourier transform of NMR data in ‘NMR order’.
\end{description}

\texttt{fft()}
None-norm (power) conserving Fourier transform.

\texttt{fft_positive} Forier transform with a positive exponential.

\texttt{ifft()} Inversion Fourier transform.

\texttt{fft_norm} Norm (power) conserving Fourier transform.

\texttt{fft()} for documentation of the transformation applied by this function.
nmrglue.process.proc_base.fft_positive

nmrglue.process.proc_base.fft_positive(data)

Fourier transform with positive exponential, NMR ordering of results

This is similar to the transform performed by NMRPipe’s FFT function.

See fft() for documentation of the transformation applied by this function.

Parameters data : ndarray

Array of NMR data.

Returns ndata : ndarray

Fourier transform of NMR data in ‘NMR order’.

See also:

ifft_positive Inversion Fourier transform.

fft_norm Norm (power) conserving Fourier transform.

fft_positive Fourier transform with a positive exponential.

nmrglue.process.proc_base.ifft

nmrglue.process.proc_base.ifft(data)

Inverse fourier transform, NMR ordering of results.

See fft() for documentation of the transformation applied by this function.

Parameters data : ndarray

Array of NMR data.

Returns ndata : ndarray

Inverse fourier transform of NMR data in ‘NMR order’.

See also:

fft Fourier transform.

ifft_norm Norm (power) conserving inverse Fourier transform.

ifft_positive inverse Fourier transform with a positive exponential.

nmrglue.process.proc_base.ifft_norm

nmrglue.process.proc_base.ifft_norm(data)

Inverse fourier transform, total power preserved, NMR ordering of results

This is similar to the transform performed by the Rowland NMR Toolkit’s IFFT function.

See fft() for documentation of the transformation applied by this function.

Parameters data : ndarray

Array of NMR data.

Returns ndata : ndarray
Inverse fourier transform of NMR data in ‘NMR order’.

See also:

- `fft_norm` Norm (power) conserving Fourier transform.
- `ifft` Non-norm (power) conserving inverse Fourier transform.
- `ifft_positive` inverse Fourier transform with a positive exponential.

**nmrglue.process.proc_base.ifft_positive**

```
nmrglue.process.proc_base.ifft_positive(data)
```

Inverse fourier transform with positive exponential, NMR ordered results.

This is similar to the transform performed by NMRPipe’s FFT function with the -inv flag

**Parameters**
- **data**: ndarray
  Array of NMR data.

**Returns**
- **ndata**: ndarray
  Inverse fourier transform of NMR data in ‘NMR order’.

See also:

- `fft_positive` Fourier transform with a positive exponential.
- `ifft_norm` Norm (power) conserving inverse Fourier transform.
- `ifft` Inverse Fourier transform with a negative exponential.

**nmrglue.process.proc_base.ha**

```
nmrglue.process.proc_base.ha(data)
```

Hadamard Transform

**Parameters**
- **data**: ndarray
  Array of NMR data.

**Returns**
- **ndata**: ndarray
  Hadamard transform of NMR data.

**Notes**

This function is very slow. Implement a Fast Walsh-Hadamard Transform with sequency/Walsh ordering (FWHT_w) will result in much faster transforms.


---

3.2. process modules
nmrglue.process.proc_base.ht

\texttt{nmrglue.process.proc_base.ht}(\textit{data}, N=None)

Hilbert transform.

Reconstruct imaginary data via hilbert transform.

\textbf{Parameters} \textit{data} : ndarrat

Array of NMR data.

\textit{N} : int or None

Number of Fourier components.

\textbf{Returns} \textit{ndata} : ndarray

NMR data which has been Hilvert transformed.

\textbf{Standard NMR}

\begin{tabular}{ll}
\hline
\texttt{di(data)} & Delete imaginary from data \\
\texttt{ps(data[, p0, p1, inv])} & Linear phase correction \\
\texttt{ps_exp(data[, p0, tc, inv])} & Exponential Phase Correction \\
\texttt{tp(data[, hyper])} & Transpose data. \\
\texttt{ytp(data[, hyper])} & Transpose data. \\
\texttt{xy2yx(data[, hyper])} & Transpose data. \\
\texttt{tp_hyper(data)} & Hypercomplex transpose. \\
\texttt{zf_inter(data[, pts])} & Zero fill between points. \\
\texttt{zf_pad(data[, pad, mid])} & Zero fill by padding with zeros. \\
\texttt{zf(data[, pad, mid])} & Zero fill by padding with zeros. \\
\texttt{zf_double(data, n[, mid])} & Zero fill by doubling original data size once or multiple times. \\
\texttt{zf_size(data, size[, mid])} & Zero fill to given size. \\
\texttt{zf_auto(data[, mid])} & Zero fill to next largest power of two. \\
\hline
\end{tabular}

nmrglue.process.proc_base.di

\texttt{nmrglue.process.proc_base.di}(\textit{data})

Delete imaginary from data

\textbf{Parameters} \textit{data} : ndarrat

Array of NMR data.

\textbf{Returns} \textit{ndata} : ndarray

Array of NMR data without imaginaries.

nmrglue.process.proc_base.ps

\texttt{nmrglue.process.proc_base.ps}(\textit{data, p0=0.0, p1=0.0, inv=False})

Linear phase correction

\textbf{Parameters} \textit{data} : ndarrat
Array of NMR data.

**p0** : float
Zero order phase in degrees.

**p1** : float
First order phase in degrees.

**inv** : bool, optional
True for inverse phase correction

**Returns**

**ndata** : ndarray
Phased NMR data.

### nmrglue.process.proc_base.ps_exp

#### nmrglue.process.proc_base.ps_exp(data, p0=0.0, tc=0.0, inv=False)

Exponential Phase Correction

**Parameters**

**data** : ndarray
Array of NMR data.

**p0** : float
Zero order phase in degrees.

**tc** : float
Exponential decay constant.

**inv** : bool, optional
True for inverse phase correction

**Returns**

**ndata** : ndarray
Phased NMR data.

### nmrglue.process.proc_base.tp

#### nmrglue.process.proc_base.tp(data, hyper=False)

Transpose data.

**Parameters**

**data** : ndarray
Array of NMR data.

**hyper** : bool
True if hypercomplex data.

**Returns**

**ndata** : ndarray
Array of NMR data with axes transposed.
nmrglue.process.proc_base.ytp

**nmrglue.process.proc_base.ytp** *(data, hyper=False)*

Transpose data.

**Parameters**

- **data**: ndarray
  - Array of NMR data.

- **hyper**: bool
  - True if hypercomplex data.

**Returns**

- **ndata**: ndarray
  - Array of NMR data with axes transposed.

nmrglue.process.proc_base.xy2yx

**nmrglue.process.proc_base.xy2yx** *(data, hyper=False)*

Transpose data.

**Parameters**

- **data**: ndarray
  - Array of NMR data.

- **hyper**: bool
  - True if hypercomplex data.

**Returns**

- **ndata**: ndarray
  - Array of NMR data with axes transposed.

nmrglue.process.proc_base.tp_hyper

**nmrglue.process.proc_base.tp_hyper** *(data)*

Hypercomplex transpose.

Use when both dimension are complex.

**Parameters**

- **data**: ndarray
  - Array of hypercomplex NMR data.

**Returns**

- **ndata**: ndarray
  - Array of hypercomplex NMR data with axes transposed.

nmrglue.process.proc_base.zf_inter

**nmrglue.process.proc_base.zf_inter** *(data, pts=1)*

Zero fill between points.

**Parameters**

- **data**: ndarray
  - Array of NMR data.

- **pts**: int
  - Number zeros to add between points.
Returns **ndata** : ndarray

Array of NMR data to which *pts* zero have been added between all points.

**nmrglue.process.proc_base.zf_pad**

```python
nmrglue.process.proc_base.zf_pad(data, pad=0, mid=False)
```

Zero fill by padding with zeros.

**Parameters**

- **data** : ndarray
  
  Array of NMR data.

- **pad** : int
  
  Number of zeros to pad data with.

- **mid** : bool
  
  True to zero fill in middle of data.

**Returns** **ndata** : ndarray

Array of NMR data to which *pad* zeros have been appended to the end or middle of the data.

**nmrglue.process.proc_base.zf**

```python
nmrglue.process.proc_base.zf(data, pad=0, mid=False)
```

Zero fill by padding with zeros.

**Parameters**

- **data** : ndarray
  
  Array of NMR data.

- **pad** : int
  
  Number of zeros to pad data with.

- **mid** : bool
  
  True to zero fill in middle of data.

**Returns** **ndata** : ndarray

Array of NMR data to which *pad* zeros have been appended to the end or middle of the data.

**nmrglue.process.proc_base.zf_double**

```python
nmrglue.process.proc_base.zf_double(data, n, mid=False)
```

Zero fill by doubling original data size once or multiple times.

**Parameters**

- **data** : ndarray
  
  Array of NMR data.

- **n** : int
  
  Number of times to double the size of the data.

- **mid** : bool
  
  True to zero fill in middle of data.

True to zero fill in the middle of data.

**Returns** ndata: ndarray

Zero filled array of NMR data.

**nmrglue.process.proc_base zf_size**

**nmrglue.process.proc_base.zf_size**(data, size, mid=False)

Zero fill to given size.

**Parameters**

- data: ndarray
  Array of NMR data.

- size: int
  Size of data after zero filling.

- mid: bool
  True to zero fill in the middle of data.

**Returns** ndata: ndarray

Zero filled array of NMR data.

**nmrglue.process.proc_base zf_auto**

**nmrglue.process.proc_base zf_auto**(data, mid=False)

Zero fill to next largest power of two.

**Parameters**

- data: ndarray
  Array of NMR data.

- mid: bool
  True to zero fill in the middle of data.

**Returns** ndata: ndarray

Zero filled array of NMR data.

**Basic Utilities**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(data[, r, i, c])</td>
<td>Add constant.</td>
</tr>
<tr>
<td>add_r(data)</td>
<td>Add real and imaginary components.</td>
</tr>
<tr>
<td>dx(data)</td>
<td>Derivative by central difference.</td>
</tr>
<tr>
<td>ext(data[, x0, xn, y0, yn])</td>
<td>Extract region.</td>
</tr>
<tr>
<td>ext_left(data)</td>
<td>Extract the left half of spectrum.</td>
</tr>
<tr>
<td>ext_mid(data)</td>
<td>Extract the middle of the spectrum.</td>
</tr>
<tr>
<td>integ(data)</td>
<td>Integrate by cumulative sum.</td>
</tr>
<tr>
<td>mc(data)</td>
<td>Modulus calculation.</td>
</tr>
<tr>
<td>mc_pow(data)</td>
<td>Modulus calculation.</td>
</tr>
<tr>
<td>mir_left(data)</td>
<td>Append a mirror image of the data on the left.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>mir_right(data)</code></td>
<td>Append a mirror image of the data on the right.</td>
</tr>
<tr>
<td><code>mir_center(data)</code></td>
<td>Append a mirror image of the data in the center.</td>
</tr>
<tr>
<td><code>mir_center_onepoint(data)</code></td>
<td>Append a mirror image of the data in the center with a one point shift amd negate appended imaginary data.</td>
</tr>
<tr>
<td><code>mult(data[, r, i, c])</code></td>
<td>Multiply by a constant.</td>
</tr>
<tr>
<td><code>rev(data)</code></td>
<td>Reverse data.</td>
</tr>
<tr>
<td><code>set(data, c)</code></td>
<td>Set to a constant.</td>
</tr>
<tr>
<td><code>set_complex(data, v)</code></td>
<td>Set real and imaginary portions of data to a constant.</td>
</tr>
<tr>
<td><code>set_real(data, v)</code></td>
<td>Set real component of data to a constant.</td>
</tr>
<tr>
<td><code>set_imag(data, v)</code></td>
<td>Set imaginary portion of data to a constant.</td>
</tr>
<tr>
<td><code>ri2c(data)</code></td>
<td>Interleave real and imaginary data into a real array.</td>
</tr>
<tr>
<td><code>interleave_complex(data)</code></td>
<td>Unpack complex data into an interleaved real, imaginary array.</td>
</tr>
<tr>
<td><code>unpack_complex(data)</code></td>
<td>Unpacks complex array into real array (interleaves values).</td>
</tr>
<tr>
<td><code>c2ri(data)</code></td>
<td>Separate interleaved real, imaginary data into complex array.</td>
</tr>
<tr>
<td><code>seperate_interleaved(data)</code></td>
<td>Separate interleaved real, imaginary data into complex array.</td>
</tr>
<tr>
<td><code>pack_complex(data)</code></td>
<td>Packs interleaved real array into complex array.</td>
</tr>
<tr>
<td><code>decode_States(data)</code></td>
<td>Decode data collected using States (seperates interleaved data).</td>
</tr>
<tr>
<td><code>ri2rr(data)</code></td>
<td>Append imaginary data to end of real data, returning a real array.</td>
</tr>
<tr>
<td><code>append_imag(data)</code></td>
<td>Append imaginary data to end of real data, returning a real array.</td>
</tr>
<tr>
<td><code>rr2ri(data)</code></td>
<td>Unappend real and imaginary data returning a complex array.</td>
</tr>
<tr>
<td><code>unappend_imag(data)</code></td>
<td>Unappend real and imaginary data returning a complex array.</td>
</tr>
<tr>
<td><code>exlr(data)</code></td>
<td>Exchange left and right halves of array.</td>
</tr>
<tr>
<td><code>exchange_lr(data)</code></td>
<td>Exchange left and right halves of array.</td>
</tr>
<tr>
<td><code>rolr(data)</code></td>
<td>Rotate left and right halves of array.</td>
</tr>
<tr>
<td><code>rotate_lr(data)</code></td>
<td>Rotate left and right halves of array.</td>
</tr>
<tr>
<td><code>swap(data)</code></td>
<td>Swap real and imaginary data.</td>
</tr>
<tr>
<td><code>swap_ri(data)</code></td>
<td>Swap real and imaginary data.</td>
</tr>
<tr>
<td><code>bswap(data)</code></td>
<td>Byteswap data.</td>
</tr>
<tr>
<td><code>byte_swap(data)</code></td>
<td>Byteswap data.</td>
</tr>
<tr>
<td><code>neg_left(data)</code></td>
<td>Negate left half.</td>
</tr>
<tr>
<td><code>neg_right(data)</code></td>
<td>Negate right half.</td>
</tr>
<tr>
<td><code>neg_middle(data)</code></td>
<td>Negate middle half.</td>
</tr>
<tr>
<td><code>neg_edges(data)</code></td>
<td>Negate edge half (non-middle) of spectra.</td>
</tr>
<tr>
<td><code>neg_all(data)</code></td>
<td>Negate data</td>
</tr>
<tr>
<td><code>neg_real(data)</code></td>
<td>Negate real data.</td>
</tr>
<tr>
<td><code>neg_imag(data)</code></td>
<td>Negate imaginary data</td>
</tr>
<tr>
<td><code>neg_even(data)</code></td>
<td>Negate even points</td>
</tr>
<tr>
<td><code>neg_odd(data)</code></td>
<td>Negate odd points</td>
</tr>
<tr>
<td><code>neg_alt(data)</code></td>
<td>Negate alternate (odd) points.</td>
</tr>
<tr>
<td><code>abs(data)</code></td>
<td>Replace data with absolute value of data (abs of real, imag seperately)</td>
</tr>
</tbody>
</table>
sign((data))

Replace data with sign (-1 or 1) of data (separately on each channel)

```
nmrglue.process.proc_base.add
```

Add constant.

**Parameters**

- **data**: ndarray
  - Array of NMR data.
- **r**: float
  - Constant to add to real data.
- **i**: float
  - Constant to add to imaginary data, ignored if no data does not have imaginary data.
- **c**: float
  - Constant to add to both real and imaginary data. This constant is only added to the real data if no imaginary data is present.

**Returns**

- **ndata**: ndarray
  - Array of NMR data with constant added.

```
nmrglue.process.proc_base.add_ri
```

Add real and imaginary components.

**Parameters**

- **data**: ndarray
  - Array of complex NMR data.

**Returns**

- **ndata**: ndarray
  - Sum of real and imaginary component of NMR data.

```
nmrglue.process.proc_base.dx
```

Derivative by central difference

Edges are takes as difference between nearest points

**Parameters**

- **data**: ndarray
  - Array of NMR data.

**Returns**

- **ndata**: ndarray
  - Derivate of NMR data.
nmrglue.process.proc_base.ext

nmrglue.process.proc_base.ext \((data, x0=None, xn=None, y0=None, yn=None)\)
Extract region.

Data should be two dimensional. Axes are labeled Y, X.

**Parameters**

- **data**: ndarray
  Array of NMR data.
- **x0**: int
  X-axis extract region start.
- **xn**: int
  X-axis extract region end.
- **y0**: int
  Y-axis extract region start.
- **yn**: int
  Y-axis extract region end.

**Returns**

- **ndata**: ndarray
  Extracted region.

nmrglue.process.proc_base.ext_left

nmrglue.process.proc_base.ext_left \((data)\)
Extract the left half of spectrum.

**Parameters**

- **data**: ndarray
  Array of NMR data.

**Returns**

- **ndata**: ndarray
  Left half of spectrum.

nmrglue.process.proc_base.ext_right

nmrglue.process.proc_base.ext_right \((data)\)
Extract the right half of the spectrum.

**Parameters**

- **data**: ndarray
  Array of NMR data.

**Returns**

- **ndata**: ndarray
  Right half of spectrum.
nmrglue.process.proc_base.ext_mid

nmrglue.process.proc_base.ext_mid(data)
Extract the middle of the spectrum.

Parameters data: ndarray
Array of NMR data.

Returns ndata: ndarray
Middle half of spectrum.

nmrglue.process.proc_base.integ

nmrglue.process.proc_base.integ(data)
Integrate by cumulative sum. The units of the integral is in points (i.e. dx is equal to 1).

Parameters data: ndarray
Array of NMR data.

Returns ndata: ndarray
Integrated NMR data.

nmrglue.process.proc_base.mc

nmrglue.process.proc_base.mc(data)
Modulus calculation.
Calculates sqrt(real^2 + imag^2)

nmrglue.process.proc_base.mc_pow

nmrglue.process.proc_base.mc_pow(data)
Modulus calculation. Squared version.
Calculated real^2+imag^2

nmrglue.process.proc_base.mir_left

nmrglue.process.proc_base.mir_left(data)
Append a mirror image of the data on the left.

nmrglue.process.proc_base.mir_right

nmrglue.process.proc_base.mir_right(data)
Append a mirror image of the data on the right.
nmrglue.process.proc_base.mir_center

nmrglue.process.proc_base.mir_center(data)
Append a mirror image of the data in the center.

nmrglue.process.proc_base.mir_center_onpoint

nmrglue.process.proc_base.mir_center_onpoint(data)
Append a mirror image of the data in the center with a one point shift amd negate appended imaginary data.

nmrglue.process.proc_base.mult

nmrglue.process.proc_base.mult(data, r=1.0, i=1.0, c=1.0)
Multiply by a constant.

Parameters data : ndarray
Array of NMR data.

r : float
Constant to multiply real channel by.

i : float
Constant to multiply imaginary channel by. If data has no imaginary component, this parameter is ignored.

c : float
Constant to multiply both channels by.

Returns ndata : ndarray
Array of NMR data which has been multiplied by a constant.

nmrglue.process.proc_base.rev

nmrglue.process.proc_base.rev(data)
Reverse data.

nmrglue.process.proc_base.set

nmrglue.process.proc_base.set(data, c)
Set to a constant.

Parameters data : ndarray
Array of NMR data.

c : float or complex
Constant to set data to.

Returns ndata : ndarray
Array of constants.
Notes

data is set in place, if original data is required make a copy before calling this function.

nmrglue.process.proc_base.set_complex

nmrglue.process.proc_base.set_complex(data, v)
Set real and imaginary portions of data to a constant.

Parameters
data : ndarray
    Array of NMR data.

v : float
    Constant to set both real and imaginary component of data to.

Returns
ndata : ndarray
    Array with real and imaginary components set to v.

Notes

data is set in place, if original data is required make a copy before calling this function.

nmrglue.process.proc_base.set_real

nmrglue.process.proc_base.set_real(data, v)
Set real component of data to a constant.

Parameters
data : ndarray
    Array of NMR data.

v : float
    Constant to set both real component of data to.

Returns
ndata : ndarray
    Array of NMR data with real components set to v.

Notes

data is set in place, if original data is required make a copy before calling this function.

nmrglue.process.proc_base.set_imag

nmrglue.process.proc_base.set_imag(data, v)
Set imaginary portion of data to a constant.

Parameters
data : ndarray
    Array of NMR data.

v : float
Constant to set both imaginary component of `data` to.

Returns `ndata`: ndarray

Array of NMR data with imaginary components set to v.

Notes

data is set in place, if original data is required make a copy before calling this function.

```
nmrglue.process.proc_base.ri2c
```

```
nmrglue.process.proc_base.ri2c(data)
```
Interleave real and imaginary data into a real array.

```
nmrglue.process.proc_base.interleave_complex
```

```
nmrglue.process.proc_base.interleave_complex(data)
```
Unpack complex data into an interleaved real, imaginary array.

```
nmrglue.process.proc_base.unpack_complex
```

```
nmrglue.process.proc_base.unpack_complex(data)
```
Unpacks complex array into real array (interleaves values).

```
nmrglue.process.proc_base.c2ri
```

```
nmrglue.process.proc_base.c2ri(data)
```
Seperate interleaved real, imaginary data into complex array.

Assumes data is real only, ignores imaginary portion of data.

```
nmrglue.process.proc_base.seperate_interleaved
```

```
nmrglue.process.proc_base.seperate_interleaved(data)
```
Seperate interleaved real, imaginary data into complex array.

```
nmrglue.process.proc_base.pack_complex
```

```
nmrglue.process.proc_base.pack_complex(data)
```
Packs interleaved real array into complex array.

```
nmrglue.process.proc_base.decode_States
```

```
nmrglue.process.proc_base.decode_States(data)
```
Decode data collected using States (seperates interleaved data).
nmrglue.process.proc_base.ri2rr

nmrglue.process.proc_base.ri2rr(data)
Append imaginary data to end of real data, returning a real array.

nmrglue.process.proc_base.append_imag

nmrglue.process.proc_base.append_imag(data)
Append imaginary data to end of real data, returning a real array.

nmrglue.process.proc_base.rr2ri

nmrglue.process.proc_base.rr2ri(data)
Unappend real and imaginary data returning a complex array.

nmrglue.process.proc_base.unappend_imag

nmrglue.process.proc_base.unappend_imag(data)
Unappend real and imaginary data returning a complex array.

nmrglue.process.proc_base.exlr

nmrglue.process.proc_base.exlr(data)
Exchange left and right halves of array.

nmrglue.process.proc_base.exchange_lr

nmrglue.process.proc_base.exchange_lr(data)
Exchange left and right halves of array.

nmrglue.process.proc_base.rolr

nmrglue.process.proc_base.rolr(data)
Rotate left and right halves of array.

nmrglue.process.proc_base.rotate_lr

nmrglue.process.proc_base.rotate_lr(data)
Rotate left and right halves of array.

nmrglue.process.proc_base.swap

nmrglue.process.proc_base.swap(data)
Swap real and imaginary data.
**nmrglue.process.proc_base.swap_ri**

```python
nmrglue.process.proc_base.swap_ri(data)
```

Swap real and imaginary data.

**nmrglue.process.proc_base.bswap**

```python
nmrglue.process.proc_base.bswap(data)
```

Byteswap data

**nmrglue.process.proc_base.byte_swap**

```python
nmrglue.process.proc_base.byte_swap(data)
```

Byteswap data

**nmrglue.process.proc_base.neg_left**

```python
nmrglue.process.proc_base.neg_left(data)
```

Negate left half.

**nmrglue.process.proc_base.neg_right**

```python
nmrglue.process.proc_base.neg_right(data)
```

Negate right half.

**nmrglue.process.proc_base.neg_middle**

```python
nmrglue.process.proc_base.neg_middle(data)
```

Negate middle half.

**nmrglue.process.proc_base.neg_edges**

```python
nmrglue.process.proc_base.neg_edges(data)
```

Negate edge half (non-middle) of spectra.

**nmrglue.process.proc_base.neg_all**

```python
nmrglue.process.proc_base.neg_all(data)
```

Negate data

**nmrglue.process.proc_base.neg_real**

```python
nmrglue.process.proc_base.neg_real(data)
```

Negate real data
nmrglue.process.proc_base.neg_imag

nmrglue.process.proc_base.neg_imag(data)
Negate imaginary data

nmrglue.process.proc_base.neg_even

nmrglue.process.proc_base.neg_even(data)
Negate even points

nmrglue.process.proc_base.neg_odd

nmrglue.process.proc_base.neg_odd(data)
Negate odd points

nmrglue.process.proc_base.neg_alt

nmrglue.process.proc_base.neg_alt(data)
Negate alternate (odd) points.

nmrglue.process.proc_base.abs

nmrglue.process.proc_base.abs(data)
Replace data with absolute value of data (abs of real, imag seperately)

nmrglue.process.proc_base.sign

nmrglue.process.proc_base.sign(data)
Replace data with sign (-1 or 1) of data (seperately on each channel)

Misc

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>coadd</strong>(data, clist[, axis])</td>
<td>Coadd data.</td>
</tr>
<tr>
<td><strong>coac</strong>(data, clist[, axis])</td>
<td>Coadd data.</td>
</tr>
<tr>
<td><strong>thres</strong>(data[, thres])</td>
<td>Mark values less than thres as invalid (for use with filters)</td>
</tr>
<tr>
<td><strong>conv</strong>(data[, kern, m, c])</td>
<td>Convolute data with kernel.</td>
</tr>
<tr>
<td><strong>convolve</strong>(data[, kern, m, c])</td>
<td>Convolute data with kernel.</td>
</tr>
<tr>
<td><strong>corr</strong>(data[, kern, m, c])</td>
<td>Correlate data with a kernel (weights).</td>
</tr>
<tr>
<td><strong>correlate</strong>(data[, kern, m, c])</td>
<td>Correlate data with a kernel (weights).</td>
</tr>
<tr>
<td><strong>filter_median</strong>(data[, s, m, c])</td>
<td>Apply a median filter.</td>
</tr>
<tr>
<td><strong>filter_min</strong>(data[, s, m, c])</td>
<td>Apply a minimum filter.</td>
</tr>
<tr>
<td><strong>filter_max</strong>(data[, s, m, c])</td>
<td>Apply a maximum filter.</td>
</tr>
<tr>
<td><strong>filter_percentile</strong>(data, percentile[, s, m, c])</td>
<td>Apply a percentile filter.</td>
</tr>
<tr>
<td><strong>filter_rank</strong>(data, rank[, s, m, c])</td>
<td>Apply a rank filter.</td>
</tr>
<tr>
<td><strong>filter_amin</strong>(data[, s, m, c])</td>
<td>Apply an absolute minimum filter.</td>
</tr>
<tr>
<td><strong>filter_amax</strong>(data[, s, m, c])</td>
<td>Apply an absolute maximum filter.</td>
</tr>
</tbody>
</table>
### Table 3.46 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filter_range(data[, s, m, c])</code></td>
<td>Apply a range filter.</td>
</tr>
<tr>
<td><code>filter_avg(data[, s, m, c])</code></td>
<td>Apply an average filter.</td>
</tr>
<tr>
<td><code>filter_dev(data[, s, m, c])</code></td>
<td>Apply a standard deviation filter.</td>
</tr>
<tr>
<td><code>filter_sum(data[, s, m, c])</code></td>
<td>Apply a summation filter.</td>
</tr>
<tr>
<td><code>filter_generic(data, filter[, s, m, c])</code></td>
<td>Apply a generic filter.</td>
</tr>
<tr>
<td><code>nmr_reorder(data)</code></td>
<td>Reorder spectrum after FT transform to NMR order (swap halves and reverse).</td>
</tr>
<tr>
<td><code>qart(data[, a, f])</code></td>
<td>Scale Quad Artifacts.</td>
</tr>
<tr>
<td><code>qart_auto(data)</code></td>
<td>Scale quad artifacts by values from Gram-Schmidt orthogonalization.</td>
</tr>
<tr>
<td><code>gram_schmidt(data)</code></td>
<td>Calculate Gram-Schmidt orthogonalization parameters.</td>
</tr>
<tr>
<td><code>qmix(data, carr)</code></td>
<td>Mix input and output channels provided coefficient array.</td>
</tr>
<tr>
<td><code>smo(data, n)</code></td>
<td>Smooth data.</td>
</tr>
<tr>
<td><code>center(data, n)</code></td>
<td>Center data.</td>
</tr>
<tr>
<td><code>zd(data, window[, x0, slope])</code></td>
<td>Zero Diagonal band with generic window function.</td>
</tr>
<tr>
<td><code>zd_boxcar(data[, wide, x0, slope])</code></td>
<td>Zero diagonal band with a boxcar function.</td>
</tr>
<tr>
<td><code>zd_triangle(data[, wide, x0, slope])</code></td>
<td>Zero diagonal band with triangle function</td>
</tr>
<tr>
<td><code>zd_sinebell(data[, wide, x0, slope])</code></td>
<td>Zero diagonal band with sinebell function.</td>
</tr>
<tr>
<td><code>zd_gaussian(data[, wide, x0, slope, g])</code></td>
<td>Zero Diagonal band with gaussian function</td>
</tr>
</tbody>
</table>

#### `nmrglue.process.proc_base.coadd`

```python
nmrglue.process.proc_base.coadd(data, clist, axis=-1)
```

Coadd data.

Reduce data along axis by blocks data and multiplying by coefficients in clist. Incomplete blocks are discarded.

**Parameters**
- **data**: 2D ndarray
  - Array of NMR data data.
- **clist**: list
  - List of Coefficients
- **axis**: `{0, 1, -1}`
  - Axis to reduce. 0 for Y, 1 or -1 for X.

#### `nmrglue.process.proc_base.coad`

```python
nmrglue.process.proc_base.coad(data, clist, axis=-1)
```

Coadd data.

Reduce data along axis by blocks data and multiplying by coefficients in clist. Incomplete blocks are discarded.

**Parameters**
- **data**: 2D ndarray
  - Array of NMR data data.
- **clist**: list
  - List of Coefficients
- **axis**: `{0, 1, -1}`
  - Axis to reduce. 0 for Y, 1 or -1 for X.
nmrglue.process.proc_base.thres

Mark values less than thres as invalid (for use with filters)

Parameters data : ndarray
    Array of NMR data.

    thres : float
        Threshold value.

Returns ndata : masked ndarray
    Masked array of NMR data with values less than thres masked.

nmrglue.process.proc_base.conv

Convolute data with kernel.
Real and imaginary components of data are convolved separately.

Parameters data : ndarray
    Array of NMR data.

    kern : array_like
        Convolution kernel

    m : {'reflect', 'constant', 'nearest', 'wrap'}
        How edges of convolution are dealt with.

    c : float
        Constant value for use when m is 'constant'.

Returns ndata : ndarray
    NMR data which has been convolved with the given kernel.

nmrglue.process.proc_base.convolute

Convolute data with kernel.
Real and imaginary components of data are convolved separately.

Parameters data : ndarray
    Array of NMR data.

    kern : array_like
        Convolution kernel

    m : {'reflect', 'constant', 'nearest', 'wrap'}
        How edges of convolution are dealt with.

    c : float
Constant value for use when m is ‘constant’.

Returns ndata : ndarray
    NMR data which has been convolved with the given kernel.

nmrglue.process.proc_base.corr

nmrglue.process.proc_base.corr(data, kern=[1.0], m='wrap', c=0.0)
Correlate data with a kernel (weights).
Real and imaginary components of data are correlated separately.

Parameters data : ndarray
    Array of NMR data.

    kern : array_like
        Correlation kernel (weights).

    m : {'reflect', 'constant', 'nearest', 'wrap'}
        How edges of correlation are dealt with.

    c : float
        Constant value for use when m is ‘constant’.

Returns ndata : ndarray
    NMR data which has been correlated with the given kernel.

nmrglue.process.proc_base.correlate

nmrglue.process.proc_base.correlate(data, kern=[1.0], m='wrap', c=0.0)
Correlate data with a kernel (weights).
Real and imaginary components of data are correlated separately.

Parameters data : ndarray
    Array of NMR data.

    kern : array_like
        Correlation kernel (weights).

    m : {'reflect', 'constant', 'nearest', 'wrap'}
        How edges of correlation are dealt with.

    c : float
        Constant value for use when m is ‘constant’.

Returns ndata : ndarray
    NMR data which has been correlated with the given kernel.
**nmrglue Documentation, Release 0.7-dev**

**nmrglue.process.proc_base.filter_median**

```python
nmrglue.process.proc_base.filter_median(data, s=(1, 1), m='wrap', c=0.0)
```
Apply a median filter.
Real and imaginary components are filtered separately.

**Parameters**
- `data` : ndarray
  Array of NMR data.
- `s` : tuple
  Shape or size taken for each step of the filter.
- `m` : {'reflect', 'constant', 'nearest', 'wrap'}
  How edges of are dealt with.
- `c` : float
  Constant value for use when m is 'constant'.

**Returns**
- `ndata` : ndarray
  Filtered NMR data.

**nmrglue.process.proc_base.filter_min**

```python
nmrglue.process.proc_base.filter_min(data, s=(1, 1), m='wrap', c=0.0)
```
Apply a minimum filter.
Real and imaginary components are filtered separately.

**Parameters**
- `data` : ndarray
  Array of NMR data.
- `s` : tuple
  Shape or size taken for each step of the filter.
- `m` : {'reflect', 'constant', 'nearest', 'wrap'}
  How edges of are dealt with.
- `c` : float
  Constant value for use when m is 'constant'.

**Returns**
- `ndata` : ndarray
  Filtered NMR data.

**nmrglue.process.proc_base.filter_max**

```python
nmrglue.process.proc_base.filter_max(data, s=(1, 1), m='wrap', c=0.0)
```
Apply a maximum filter.
Real and imaginary components are filtered separately.

**Parameters**
- `data` : ndarray
  Array of NMR data.
s : tuple
    Shape or size taken for each step of the filter.
m : {'reflect', 'constant', 'nearest', 'wrap'}
    How edges of are dealt with.
c : float
    Constant value for use when m is 'constant'.

Returns ndata : ndarray
    Filtered NMR data.

nmrglue.process.proc_base.filter_percentile

nmrglue.process.proc_base.filter_percentile(data, percentile, s=(1, 1), m='wrap', c=0.0)
    Apply a percentile filter.
    Real and imaginary components are filtered separately.

Parameters data : ndarray
    Array of NMR data.
percentile : float
    Filter percentile parameter.
s : tuple
    Shape or size taken for each step of the filter.
m : {'reflect', 'constant', 'nearest', 'wrap'}
    How edges of are dealt with.
c : float
    Constant value for use when m is 'constant'.

Returns ndata : ndarray
    Filtered NMR data.

nmrglue.process.proc_base.filter_rank

nmrglue.process.proc_base.filter_rank(data, rank, s=(1, 1), m='wrap', c=0.0)
    Apply a rank filter.
    Real and imaginary components are filtered separately.

Parameters data : ndarray
    Array of NMR data.
rank : int
    Filter rank parameter.
s : tuple
    Shape or size taken for each step of the filter.
m : {'reflect', 'constant', 'nearest', 'wrap'}
How edges of are dealt with.
c : float
Constant value for use when m is 'constant'.

Returns ndata : ndarray
Filtered NMR data.

nmrglue.process.proc_base.filter_amin

nmrglue.process.proc_base.filter_amin(data, s=(1, 1), m='wrap', c=0.0)
Apply an absolute minimum filter.
Real and imaginary components are filtered separately.

Parameters data : ndarray
Array of NMR data.
s : tuple
Shape or size taken for each step of the filter.
m : {'reflect', 'constant', 'nearest', 'wrap'}
How edges of are dealt with.
c : float
Constant value for use when m is 'constant'.

Returns ndata : ndarray
Filtered NMR data.

nmrglue.process.proc_base.filter_amax

nmrglue.process.proc_base.filter_amax(data, s=(1, 1), m='wrap', c=0.0)
Apply an absolute maximum filter.
Real and imaginary components are filtered separately.

Parameters data : ndarray
Array of NMR data.
s : tuple
Shape or size taken for each step of the filter.
m : {'reflect', 'constant', 'nearest', 'wrap'}
How edges of are dealt with.
c : float
Constant value for use when m is 'constant'.

Returns ndata : ndarray
Filtered NMR data.
nmrglue.process.proc_base.filter_range

nmrglue.process.proc_base.filter_range(data, s=(1,1), m='wrap', c=0.0)
Apply a range filter.
Real and imaginary components are filtered separately.

**Parameters**

- **data**: ndarray
  Array of NMR data.
- **s**: tuple
  Shape or size taken for each step of the filter.
- **m**: {'reflect', 'constant', 'nearest', 'wrap'}
  How edges of are dealt with.
- **c**: float
  Constant value for use when m is 'constant'.

**Returns**

- **ndata**: ndarray
  Filtered NMR data.

nmrglue.process.proc_base.filter_avg

nmrglue.process.proc_base.filter_avg(data, s=(1,1), m='wrap', c=0.0)
Apply an average filter.
Real and imaginary components are filtered separately.

**Parameters**

- **data**: ndarray
  Array of NMR data.
- **s**: tuple
  Shape or size taken for each step of the filter.
- **m**: {'reflect', 'constant', 'nearest', 'wrap'}
  How edges of are dealt with.
- **c**: float
  Constant value for use when m is 'constant'.

**Returns**

- **ndata**: ndarray
  Filtered NMR data.

nmrglue.process.proc_base.filter_dev

nmrglue.process.proc_base.filter_dev(data, s=(1,1), m='wrap', c=0.0)
Apply a standard deviation filter.
Real and imaginary components are filtered separately.

**Parameters**

- **data**: ndarray
  Array of NMR data.
s : tuple
    Shape or size taken for each step of the filter.

m : {'reflect', 'constant', 'nearest', 'wrap'}
    How edges of are dealt with.

c : float
    Constant value for use when m is 'constant'.

Returns ndata : ndarray
    Filtered NMR data.

nmrglue.process.proc_base.filter_sum

nmrglue.process.proc_base.filter_sum(data, s=(1, 1), m='wrap', c=0.0)
    Apply a summation filter.
    Real and imaginary components are filtered seperately.

Parameters data : ndarray
    Array of NMR data.

    s : tuple
        Shape or size taken for each step of the filter.

    m : {'reflect', 'constant', 'nearest', 'wrap'}
        How edges of are dealt with.

    c : float
        Constant value for use when m is 'constant'.

Returns ndata : ndarray
    Filtered NMR data.

nmrglue.process.proc_base.filter_generic

nmrglue.process.proc_base.filter_generic(data, filter, s=(1, 1), m='wrap', c=0.0)
    Apply a generic filter.
    Real and imaginary components are filtered seperately.

Parameters data : ndarray
    Array of NMR data.

    filter : functions
        Python function which takes an array and returns a single value.

    s : tuple
        Shape or size taken for each step of the filter.

    m : {'reflect', 'constant', 'nearest', 'wrap'}
        How edges of are dealt with.

c : float
    Constant value for use when m is ‘constant’.

Returns ndata : ndarray
    Filtered NMR data.

nmrglue.process.proc_base.nmr_reorder

nmrglue.process.proc_base.nmr_reorder(data)
    Reorder spectrum after FT transform to NMR order (swap halves and reverse).

nmrglue.process.proc_base.qart

nmrglue.process.proc_base.qart(data, a=0.0, f=0.0)
    Scale Quad Artifacts.

    Replaces imaginary data with (1 + a) * data.imag + f * data.real

Parameters data : ndarray
    Array of NMR data.

    a : float
        Amplitude adjustment.

    f : float
        Phase adjustment.

Returns ndata : ndarray
    Array of NMR data with quadrature artifacts scaled.

nmrglue.process.proc_base.qart_auto

nmrglue.process.proc_base.qart_auto(data)
    Scale quad artifacts by values from Gram-Schmidt orthogonalization.

nmrglue.process.proc_base.gram_schmidt

nmrglue.process.proc_base.gram_schmidt(data)
    Calculate Gram-Schmidt orthogonalization parameters.

nmrglue.process.proc_base.qmix

nmrglue.process.proc_base.qmix(data, carr)
    Mix input and output channels provided coefficient array.

Parameters data : 2D ndarray
    Array of NMR data.

    carr : array_like
Array of coefficients for mixing. The size of carr must evenly divide qmix.

**Returns** ndata : ndarray

Array of NMR data with channels mixed.

### nmrglue.process.proc_base.smo

nmrglue.process.proc_base.smo(data, n)

Smooth data.

**Parameters** data : ndarray

Array of NMR data.

n : int

Size of smoothing window (+/- points)

**Returns** ndata : ndarray

Array of smoothed NMR data.

### nmrglue.process.proc_base.center

nmrglue.process.proc_base.center(data, n)

Center data.

**Parameters** data : ndarray

Array of NMR data.

n : int

Size of centering window (+/- points)

**Returns** ndata : ndarray

Array of centered NMR data.

### nmrglue.process.proc_base.zd

nmrglue.process.proc_base.zd(data, window, x0=0.0, slope=1.0)

Zero Diagonal band with generic window function.

**Parameters** data : ndarray

Array of NMR data.

window : ndarray

Window to apply to diagonal band.

wide : int

Diagonal band half width in points.

x0 : int

Starting location of diagonal band in points.

slope : float
Slope of diagonal band.

**Returns**  
**ndata** : ndarray  
Array of NMR data with diagonal band set to zero.

**nmrglue.process.proc_base.zd_boxcar**

```python
nmrglue.process.proc_base.zd_boxcar(data, wide=1, x0=0.0, slope=1.0)
```

*Zero* diagonal band with a boxcar function.

**Parameters**  
**data** : ndarray  
Array of NMR data.

**wide** : int  
Diagonal band half width in points.

**x0** : int  
Starting location of diagonal band in points.

**slope** : float  
Slope of diagonal band.

**Returns**  
**ndata** : ndarray  
Array of NMR data with diagonal band set to zero.

**nmrglue.process.proc_base.zd_triangle**

```python
nmrglue.process.proc_base.zd_triangle(data, wide=1.0, x0=0.0, slope=1.0)
```

*Zero* diagonal band with triangle function.

**Parameters**  
**data** : ndarray  
Array of NMR data.

**wide** : int  
Diagonal band half width in points.

**x0** : int  
Starting location of diagonal band in points.

**slope** : float  
Slope of diagonal band.

**Returns**  
**ndata** : ndarray  
Array of NMR data with diagonal band set to zero.

**nmrglue.process.proc_base.zd_sinebell**

```python
nmrglue.process.proc_base.zd_sinebell(data, wide=1.0, x0=0.0, slope=1.0)
```

*Zero* diagonal band with sinebell function.

**Parameters**  
**data** : ndarray
Array of NMR data.

**wide**: int
Diagonal band half width in points.

**x0**: int
Starting location of diagonal band in points.

**slope**: float
Slope of diagonal band.

**Returns** **ndata**: ndarray
Array of NMR data with diagonal band set to zero.

### nmrglue.process.proc_base.zd_gaussian

`nmrglue.process.proc_base.zd_gaussian(data, wide=1.0, x0=0.0, slope=1.0, g=1)`

Zero Diagonal band with gaussian function

**Parameters** **data**: ndarray
Array of NMR data.

**wide**: int
Diagonal band half width in points.

**x0**: int
Starting location of diagonal band in points.

**slope**: float
Slope of diagonal band.

**g**: float
Width of Gaussian function.

**Returns** **ndata**: ndarray
Array of NMR data with diagonal band set to zero.

### Low-Level Functions

The following are functions called by other processing functions. They are included here for completeness.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int2bin(n[, digits])</code></td>
<td>Integer to binary string</td>
</tr>
<tr>
<td><code>bin2int(s)</code></td>
<td>Binary string to integer</td>
</tr>
<tr>
<td><code>gray(n)</code></td>
<td>Calculate n-bit gray code</td>
</tr>
<tr>
<td><code>amin_flt(arr)</code></td>
<td></td>
</tr>
<tr>
<td><code>amax_flt(arr)</code></td>
<td></td>
</tr>
<tr>
<td><code>range_flt(arr)</code></td>
<td></td>
</tr>
<tr>
<td><code>avg_flt(arr)</code></td>
<td></td>
</tr>
<tr>
<td><code>std_flt(arr)</code></td>
<td></td>
</tr>
<tr>
<td><code>sum_flt(arr)</code></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.47 – continued from previous page

| largest_power_of_2(value) | Find the nearest power of two equal to or larger than a value. |

**nmrglue.process.proc_base.int2bin**

- `int2bin(n, digits=8)`: Integer to binary string

**nmrglue.process.proc_base.bin2int**

- `bin2int(s)`: Binary string to integer

**nmrglue.process.proc_base.gray**

- `gray(n)`: Calculate n-bit gray code

**nmrglue.process.proc_base.amin_flt**

- `amin_flt(arr)`: Minimum value in the array

**nmrglue.process.proc_base.amax_flt**

- `amax_flt(arr)`: Maximum value in the array

**nmrglue.process.proc_base.range_flt**

- `range_flt(arr)`: Range of values in the array

**nmrglue.process.proc_base.avg_flt**

- `avg_flt(arr)`: Average value in the array

**nmrglue.process.proc_base.std_flt**

- `std_flt(arr)`: Standard deviation of the array

**nmrglue.process.proc_base.sum_flt**

- `sum_flt(arr)`: Sum of values in the array
nmrglue.process.proc_base.largest_power_of_2

Find the nearest power of two equal to or larger than a value.

Parameters

value : int

Value to find nearest power of two equal to or larger than.

Returns

pw : int

Power of 2.

3.2.3 nmrglue.proc_bl

A collection of NMR processing functions for filtering, smoothing, and correcting spectral baselines.

This module is imported as nmrglue.proc_bl and can be called as such.

User Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base(data, nl[, nw])</td>
<td>Linear (first-order) baseline correction based on node list.</td>
</tr>
<tr>
<td>cbf(data[, last, apply])</td>
<td>Constant baseline correction from percentage of data.</td>
</tr>
<tr>
<td>cbf_explicit(data[, calc, apply])</td>
<td>Constant Baseline correction from an explicit region of the data.</td>
</tr>
<tr>
<td>med(data[, mw, sf, sigma])</td>
<td>Median baseline correction.</td>
</tr>
<tr>
<td>baseline_corrector(data[, wd])</td>
<td>Calculate a baseline using a distribution based classification method.</td>
</tr>
<tr>
<td>sol_general(data, filter[, w, mode])</td>
<td>Solvent filter with generic filter.</td>
</tr>
<tr>
<td>sol_boxcar(data[, w, mode])</td>
<td>Solvent filter with boxcar filter.</td>
</tr>
<tr>
<td>sol_sine(data[, w, mode])</td>
<td>Solvent filter with sine-bell filter.</td>
</tr>
<tr>
<td>sol_sine2(data[, w, mode])</td>
<td>Solvent filter with square sine-bell filter.</td>
</tr>
<tr>
<td>sol_gaussian(data[, w, mode])</td>
<td>Solvent filter with square gaussian filter.</td>
</tr>
</tbody>
</table>

nmrglue.process.proc_bl.base

nmrglue.process.proc_bl.base(data, nl[, nw=0])

Linear (first-order) baseline correction based on node list.

Parameters

data : 1D or 2D ndarray

Array of 1D or 2D NMR data.

nl : list

List of baseline nodes.

nw : float, optional

Node half-width in points.

Returns

ndata : ndarray

NMR data with first order baseline correction applied. For 2D data baseline correction is applied for each trace along the last dimension.
**nmrglue.process.proc_bl.cbf**

\[ \text{nmrglue.process.proc_bl.cbf (data, last=10, apply=slice(\text{None, None, None}))} \]

Constant baseline correction from percentage of data.

**Parameters**

- **data**: 1D or 2D ndarray
  - Array of 1D or 2D NMR data.
- **last**: float, optional
  - Percent (0 - 100) of last axis used to calculate the baseline correction.
- **apply**: slice, optional
  - Slice describing first-axis region(s) to which the baseline correction should be applied. Parameter is ignored for 1D data.

**Returns**

- **ndata**: 1D or 2D ndarray
  - NMR data with a constant baseline subtracted.

**nmrglue.process.proc_bl.cbf_explicit**

\[ \text{nmrglue.process.proc_bl.cbf_explicit (data, calc=slice(\text{None, None, None}), apply=slice(\text{None, None, None}))} \]

Constant Baseline correction from an explicit region of the data.

**Parameters**

- **data**: 1D or 2D ndarray
  - Array of 1D or 2D NMR data.
- **calc**: slice, optional
  - Slice describing region to use for calculating the baseline correction.
- **apply**: slice, optional
  - Slice describing first-axis region(s) to which the baseline correction should be applied. Parameter is ignored for 1D data.

**Returns**

- **ndata**: 1D or 2D ndarray
  - NMR data with a constant baseline subtracted.

**nmrglue.process.proc_bl.med**

\[ \text{nmrglue.process.proc_bl.med (data, mw=24, sf=16, sigma=5.0)} \]

Median baseline correction.


**Parameters**

- **data**: 1D or 2D ndarray
  - Array of 1D or 2D NMR data.
- **mw**: float
  - Median window size in pts.
- **sf**: float
  - Smooth window size in pts.
\texttt{nmrglue Documentation, Release 0.7-dev}

\begin{verbatim}
  \texttt{sigma} : float
  Standard-deviation of convoluted Gaussian window.

  \textbf{Returns ndata} : 1D or 2D ndarray
  NMR data with the median baseline subtracted.
\end{verbatim}

\texttt{nmrglue.process.proc_bl.baseline_corrector}

\begin{verbatim}
\texttt{nmrglue.process.proc_bl.baseline_corrector(data, wd=20)}
Calculate a baseline using a distribution based classification method.

\textbf{Parameters data} : 1D ndarray
  One dimensional NMR data with real values

\texttt{wd} : float
  Median window size in pts.

\textbf{Returns data} : 1D ndarray
  Baseline corrected spectrum calculated using distribution based classification
\end{verbatim}

\texttt{nmrglue.process.proc_bl.sol_general}

\begin{verbatim}
\texttt{nmrglue.process.proc_bl.sol_general(data, filter, w=16, mode='same')} 
Solvent filter with generic filter.
Algorithm described in: Marion et al. JMR 1989 84 425-430

\textbf{Parameters data} : 1D or 2D ndarray
  Array of 1D or 2D NMR data.

\texttt{filter} : ndarray
  Filter to convolve with data. Not used in solvent filter functions which specific the filter, e.g. sol_boxcar.

\texttt{w} : int, optional
  Filter length. Not used here but is used in solent filter functions which specificy the filter, e.g. sol_boxcar.

\texttt{mode} : {'valid', 'same', 'full'}, optional
  Convolution mode, ‘same’ should be used.

\textbf{Returns ndata} : 1D or 2D ndarray
  NMR data with solvent filter applied
\end{verbatim}

\texttt{nmrglue.process.proc_bl.sol_boxcar}

\begin{verbatim}
\texttt{nmrglue.process.proc_bl.sol_boxcar(data, w=16, mode='same')} 
Solvent filter with boxcar filter. See \texttt{sol_general()}. 
\end{verbatim}
**nmrglue.process.proc_bl.sol_sine**

nmrglue.process.proc_bl.sol_sine\((data, w=16, mode='same')\)
Solvent filter with sine-bell filter. See sol_general().

**nmrglue.process.proc_bl.sol_sine2**

nmrglue.process.proc_bl.sol_sine2\((data, w=16, mode='same')\)
Solvent filter with square sine-bell filter. See sol_general().

**nmrglue.process.proc_bl.sol_gaussian**

nmrglue.process.proc_bl.sol_gaussian\((data, w=16, mode='same')\)
Solvent filter with square gaussian filter. See sol_general().

**Developer Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calc_bl_linear((x, nl[, nw]))</td>
<td>Calculate a baseline using linear approximation between nodes</td>
</tr>
<tr>
<td>calc_bl_med((x, mw, sf, sigma))</td>
<td>Calculate a baseline using median baseline correction.</td>
</tr>
</tbody>
</table>

**nmrglue.process.proc_bl.calc_bl_linear**

nmrglue.process.proc_bl.calc_bl_linear\((x, nl, nw=0)\)
Calculate a baseline using linear approximation between nodes

**Parameters**

- **x**: 1D ndarray
  - One-dimensional NMR data.
- **nl**: list
  - List of baseline nodes
- **nw**: float
  - Node half-width in points

**Returns**

- **baseline**: ndarray
  - Base calculated using linear approximation between nodes.

**nmrglue.process.proc_bl.calc_bl_med**

nmrglue.process.proc_bl.calc_bl_med\((x, mw, sf, sigma)\)
Calculate a baseline using median baseline correction.

Algorithm described in: Friedrichs, M.S. JBNMR 1995 5 147-153

**Parameters**

- **x**: 1D ndarray
  - One dimensional NMR data
- **mw**: float
Median window size in pts.

\texttt{sf} : float

Smooth window size in pts.

\texttt{sigma} : float

Standard-deviation of convoluted Gaussian window.

**Returns**  
\textbf{baseline} : 1D ndarray

Baseline calculated using median baseline correction

### 3.2.4 nmrglue.proc\_lp

Linear Prediction (LP) functions for extrapolating and modeling NMR signals.

This module is imported as \texttt{nmrglue.proc\_lp} and can be called as such.

#### User Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{lp}</td>
<td>Linear prediction extrapolation of 1D or 2D data.</td>
</tr>
<tr>
<td>\texttt{lp_svd}</td>
<td>Linear Prediction extrapolation of 1D or 2D data using SVD decomposition.</td>
</tr>
<tr>
<td>\texttt{lp_qr}</td>
<td>Linear Prediction extrapolation of 1D or 2D data using QR decomposition.</td>
</tr>
<tr>
<td>\texttt{lp_cho}</td>
<td>Linear Prediction extrapolation of 1D or 2D data using Cholesky decomposition.</td>
</tr>
<tr>
<td>\texttt{lp_tls}</td>
<td>Linear Prediction extrapolation of 1D or 2D data using Total Least Squares.</td>
</tr>
<tr>
<td>\texttt{lp_2d}</td>
<td>Perform a forward 2D linear prediction extrapolation on data.</td>
</tr>
<tr>
<td>\texttt{cadzow}</td>
<td>Perform a (row wise) Cadzow-like signal enhancement on 1D or 2D data.</td>
</tr>
<tr>
<td>\texttt{lp_model}</td>
<td>Use Linear Prediction to model 1D NMR time domain data.</td>
</tr>
</tbody>
</table>

**nmrglue.process.proc\_lp.lp**

\texttt{nmrglue.process.proc\_lp.lp} \( (\texttt{data, \ pred=1, slice=slice(\text{None}, \text{None}, \text{None}), order=8, mode='f', append='after', bad_roots='auto', fix_mode='on', \text{mirror=}'\text{None}, \text{method=}'\text{svd'}) \)

Linear prediction extrapolation of 1D or 2D data.

**Parameters**  
\textbf{data} : ndarray

1D or 2D NMR data with the last (-1) axis in the time domain.

\textbf{pred} : int

Number of points to predict along the last axis.

\textbf{slice} : slice object, optional

Slice object which selects the region along the last axis to use in LP equation. The default (slice(\text{None})) will use all points.

\textbf{order} : int
Prediction order, number of LP coefficients calculated.

**mode** : {‘f’, ‘b’, ‘fb’ or ‘bf’}

Mode to generate LP filter. ‘f’ for forward,’b’ for backward, fb for ‘forward-backward and ‘bf’ for backward-forward.

**extend** : {‘before’, ‘after’}

Location to extend the data, either ‘before’ the current data, or ‘after’ the existing data. This is independent of the **mode** parameter.

**bad_roots** : {‘incr’, ‘decr’, None, ‘auto’}

Type of roots which to consider bad and to stabilize. Option are those with increasing signals ‘incr’ or decreasing signals ‘decr’. None will perform no root stabiliting. The default (‘auto’) will set the parameter based on the **mode** parameter. ‘f’ or ‘fb’ **mode** will results in a ‘incr’ **bad_roots** parameter, ‘b’ or ‘bf’ in ‘decr’

**fix_mode** : {‘on’, ‘reflect’}

Method used to stabilize bad roots, ‘on’ to move the roots onto the unit circle, ‘reflect’ to reflect bad roots across the unit circle. This parameter is ignored when **bad_roots** is None.

**mirror** : {None, ‘0’, ‘180’}

Mode to form mirror image of data before processing. None will process the data trace as provided (no mirror image). ‘0’ or ‘180’ forms a mirror image of the sliced trace to calculate the LP filter. ‘0’ should be used with data with no delay, ‘180’ with data with an initial half-point delay.


Method to use to calculate the LP filter. Choices are a SVD (‘svd’), QR (‘qr’), or Choleskey (‘choleskey’) decomposition, or Total Least Squares (‘tls’).

**Returns ndata** : ndarray

NMR data with **pred** number of points linear predicted and appended to the original data.

**Notes**

When given 2D data a series of 1D linear predictions are made to each row in the array, extending each by **pred** points. To perform a 2D linear prediction using a 2D prediction matrix use \( \text{lp2d} \).

In forward-backward or backward-forward mode root stabilizing is done on both sets of signal roots as calculated in the first mode direction. After averaging the coefficient the roots are again stabilized.

When the append parameter does not match the LP mode, for example if a backward linear prediction (mode='b') is used to predict points after the trace (append='after'), any root fixing is done before reversing the filter.

**nmrglue.process.proc_lp.lp_svd**

\[
\text{nmrglue.process.proc_lp.lp_svd}(data, \text{pred}=1, \text{slice}=\text{slice}(\text{None, None, None}), \text{order}=8, \\
\text{mode}=\text{’f’}, \text{append}=\text{’after’}, \text{bad_roots}=\text{’auto’}, \text{fix_mode}=\text{’on’}, \\
\text{mirror}=\text{None})
\]

Linear Prediction extrapolation of 1D or 2D data using SVD decomposition.
See \texttt{lp()} for documentation.

\texttt{nmrglue.process.proc_lp.lp_qr}

\texttt{nmrglue.process.proc_lp.lp_qr(data, pred=1, slice=slice(None, None, None), order=8, mode='f', append='after', bad_roots='auto', fix_mode='on', mirror=None)}

Linear Prediction extrapolation of 1D or 2D data using QR decomposition.

See \texttt{lp()} for documentation

\texttt{nmrglue.process.proc_lp.lp_cho}

\texttt{nmrglue.process.proc_lp.lp_cho(data, pred=1, slice=slice(None, None, None), order=8, mode='f', append='after', bad_roots='auto', fix_mode='on', mirror=None)}

Linear Prediction extrapolation of 1D or 2D data using Cholesky decomposition.

See \texttt{lp()} for documentation

\texttt{nmrglue.process.proc_lp.lp_tls}

\texttt{nmrglue.process.proc_lp.lp_tls(data, pred=1, slice=slice(None, None, None), order=8, mode='f', append='after', bad_roots='auto', fix_mode='on', mirror=None)}

Linear Prediction extrapolation of 1D or 2D data using Total Least Squares.

See \texttt{lp()} for documentation.

\texttt{nmrglue.process.proc_lp.lp2d}

\texttt{nmrglue.process.proc_lp.lp2d(data, pred, P, M, mirror='0', fix_points=True, method='svd')}

Perform a forward 2D linear prediction extrapolation on data.

Use the 2D linear prediction algorithm presented in: G. Zhu and A. Bax, Journal of Magnetic Resonance, 1992, 98, 192-199. to extend the last (1) axis by \texttt{pred} points. A PxM prediction matrix, C, is formed by solving the modified linear prediction equation given by:

\[ \text{data}[n,m] = \sum_{l=0}^{P-1} \sum_{k=1}^{M} C_{l,k} \times \text{data}[n-l,m-k] \]

For all valid points in data. This prediction matrix together with the data matrix with a mirror image appended is used to extend the last (1) axis by \texttt{pred} points resulting in a new array of size \([N_0, N_1+\text{pred}]\) where \(N_0\) and \(N_1\) are the sizes of the original data. To linear predict both dimensions this function should be used twice with a transpose between the uses.

Backward linear prediction using this method is not possible as the method depends on being able to mirror the data before the first collected point. In backwards mode this would correspond to being able to correctly determine points after the last point which cannot be determined using the mirror method. A backward prediction matrix can be calculated but would not prove useful.

The forward-backward averaging of the linear prediction coefficients is not possible as there is no characteristic polynomial to root and reflect. Therefore the backward prediction matrix cannot be reversed.

\textbf{Parameters} \texttt{data} : ndarray

2D NMR data (time domain for last axes).
pred : int
Number of points to predict along the last (1) axes.

P : int
Prediction matrix length along the non-predicted (0) axis.

M : int
Prediction matrix length along the predicted (1) axis.

mirror : {'0' or '180'}
Method to use for forming the mirror image of the non-predicted axis. ‘0’ indicated no initial delay, ‘180’ for a half-point delay.

fix_points : bool
True to reduce predicted points with magnitude larger than the largest data point. False leaved predicted points unaltered.

method : {'svd', 'qr', 'cholesky', 'tls'}
Method used to calculate the LP prediction matrix. See \texttt{lp()} for a description of theses methods.

Returns ndata : ndarray
2D NMR data with \textit{pred} points appended to the last (1) axes.

Notes
The axes in this function are reversed as compared to the JMR paper.

\texttt{nmrglue.process.proc\_lp.cadzow}

\texttt{nmrglue.process.proc\_lp.cadzow(data, M, K, niter, min\_var=False)}
Perform a (row wise) Cadzow-like signal enhancement on 1D or 2D data.


For 2D data performs independant enhancement on each row of data array.

Parameters data : ndarray
1D or 2D NMR data to enhance.

M : int
Large prediction order. For best results should be between \textit{K} + 5 and 2 * \textit{K}.

K : int
Reduced prediction order.

niter : int
Number if iteration of the Cadzow procedure to perform.

min_var : bool
True to adjust retained singular values using the minimum variance method. False does not correct the singular values and is the Cadzow method.

**Returns**

ndata : ndarray

Array of enhanced data

### nmrglue.process.proc_lp.lp_model

Use Linear Prediction to model 1D NMR time domain data.

**Parameters**

- **trace** : 1D ndarray
  
  One dimensional time domain NMR data to model.

- **slice** : slice object, optional
  
  Slice object which selects the region along the last axis to use in LP equation. The default, slice(None), will use all points.

- **order** : int
  
  Prediction order, number of LP coefficients calculated.

- **mode** : {‘f’, ‘b’}
  
  Mode to generate LP filter. ‘f’ for forward,’b’ for backward.

- **mirror** : {None, ‘0’, ‘180’}
  
  Mode to form mirror image of data before processing. None will process the data trace as provided (no mirror image). ‘0’ or ‘180’ forms a mirror image of the sliced trace to calculate the LP filter. ‘0’ should be used with data with no delay, ‘180’ with data with an initial half-point delay.

  
  Method to use to calculate the LP filter. Choices are a SVD (‘svd’), QR (‘qr’), or Choleskey (‘choleskey’) decomposition, or Hankel SVD (‘hsvd’).

- **full** : bool
  
  True to return amplitudes and phases calculated by performing a least squares fitting to the data after LP modeling. False will return only the damping (relaxation) factors and signal frequencies.

**Returns**

- **damp** : list
  
  List of damping (relaxation) factors found from LP modeling.

- **freq** : list
  
  List of signal frequencies found from LP modeling.

- **amp** : list, optional
  
  List of signal amplitudes found by least squares fitting of data after LP modeling, only returned when full parameter is True.

- **phase** : list, optional
  
  List of signal phases found by least squares fitting of data after LP modeling, only returned when full parameter is True.
Notes

When backward LP is used the signal roots are reflected before calculating model parameters.

Developer Functions

__developer_info__ = """" Notes

This module contains functions for performing linear prediction on NMR data. The algorithms used were selected for simplicity to show how linear prediction works not for computational speed nor stability. Locations where significant improvements can be made to improve speed or stability are indicated with SPEED and STABILITY within the source code with discussion following.

The notation for the Linear Prediction equation, coefficients, roots, etc. closely match those in “NMR Data Processing” by Hoch snd Stern. This book was references for many of the algorithms in this module.

Reduced order LP-SVD and LP-TLS methods are not implemented but should

These functions are called by high-level function are and most users will not use them in common processing scripts. Developers may be interested in them.

- **lp_1d**(trace[, pred, slice, order, mode, ...]) Linear Prediction extrapolation of 1D data.
- **extrapolate_2d**(x, C, pred, fix_points, mirror) Extrapolate points along the 1st axis using the lp2d algorithm.
- **make_lp2d_Dd**(x, P, M[, mode]) Form the lp2d equation matrix and vector.
- **cadzow_single**(x, M, K[, min_var]) Perform a single iteration of Cadzow signal enhancement on a 1D vector
- **root2damp**(pole) Calculate the damping factor from a LP root
- **root2freq**(pole) Calculate the frequency from a LP root
- **cof2amp**(z) Calculate a signal amplitude from a model coefficient
- **cof2phase**(z) Calculate a signal phase from a model coefficient
- **make_D**(x, order, mode) Make the LP equation D matrix (Da = d')
- **make_little_d**(x, order, mode) Make the LP equation d' vector (Da = d')
- **make_Dd**(x, order, mode) make the LP equation D matrix and d' vector (Da=d')
- **make_mirror**(x, mode) Make a mirror image trace.
- **find_lpc**(D, d, method) Find linear prediction filter using a provided method.
- **find_lpc_svd**(D, d) Find linear prediction filter using single value decomposition.
- **pinv_diagsvd**(s, m, L) Construct the pseudo-inverse of the sigma matrix from singular values
- **find_lpc_qr**(D, d) Find linear prediction filter using QR decomposition.
- **find_lpc_cholesky**(D, d) Find linear prediction filter using a Cholesky decomposition.
- **find_lpc_tls**(D, d) Find linear prediction filter using the Total Least Squares method
- **find_lpc_fb**(x, order, bad_roots, fix_mode, ...) Determind LP coefficient using forward-backward linear prediction.
- **find_lpc_bf**(x, order, bad_roots, fix_mode, ...) Determind LP coefficient using backward-forward linear prediction.
- **find_lproots_hsvd**(x, M, K, mode[, zmethod]) Find LP roots (poles) using the HSVD method
- **find_roots**(a[, mode]) Find LP roots (poles) from a set of LP coefficients.
- **find_coeff**(poles[, mode]) Find LP coefficients from a set of LP roots (poles).
- **reverse_filter**(a, mode) Reverse a filter (change forward LP to backwards LP).
- **fix_roots**(poles[, fix_roots, fix_mode]) Fix (stabilize) LP roots.
- **extrapolate**(trace, a, pred, append) Extrapolate points using LP prediction filter.
nmrglue Documentation, Release 0.7-dev

nmrglue.process.proc_lp.lp_1d

nmrglue.process.proc_lp.lp_1d(trace, pred=1, slice=slice(None, None, None), order=8, mode='f', append='after', bad_roots='auto', fix_mode='on', mirror=None, method='svd')

Linear Prediction extrapolation of 1D data.

**Parameters**

- **trace** : ndarray
  
  1D NMR data in the time domain.

- **pred** : int
  
  Number of points to predict along the last axis.

- **slice** : slice object, optional
  
  Slice object which selects the region along the last axis to use in LP equation. The default (slice(None)) will use all points.

- **order** : int
  
  Prediction order, number of LP coefficients calculated.

- **mode** : {'f', 'b', 'fb' or 'bf'}
  
  Mode to generate LP filter. ‘f’ for forward,’b’ for backward, fb for ‘forward-backward and ‘bf’ for backward-forward.

- **extend** : {'before', 'after'}
  
  Location to extend the data, either ‘before’ the current data, or ‘after’ the existing data. This is independent of the `mode` parameter.

- **bad_roots** : {'incr', 'decr', None, 'auto'}
  
  Type of roots which to consider bad and to stabilize. Option are those with increasing signals ‘incr’ or decreasing signals ‘decr’. None will perform no root stabiliting. The default (‘auto’) will set the parameter based on the `mode` parameter. ‘f’ or ‘fb’ `mode` will results in a ‘incr’ `bad_roots` parameter, ‘b’ or ‘bf’ in ‘decr’

- **fix_mode** : {'on', 'reflect'}
  
  Method used to stabilize bad roots, ‘on’ to move the roots onto the unit circle, ‘reflect’ to reflect bad roots across the unit circle. This parameter is ignored when `bad_roots` is None.

- **mirror** : {None, ‘0’, ‘180’}
  
  Mode to form mirror image of data before processing. None will process the data trace as provided (no mirror image). ‘0’ or ‘180’ forms a mirror image of the sliced trace to calculate the LP filter. ‘0’ should be used with data with no delay, ‘180’ with data with an initial half-point delay.

- **method** : {'svd', ‘qr’, ‘choleskey’, ‘tls’}
  
  Method to use to calculate the LP filter. Choices are a SVD (‘svd’), QR (‘qr’), or Choleskey (‘choleskey’) decomposition, or Total Least Squares (‘tls’).

**Returns**

- **ntrace** : ndarray
  
  NMR data with `pred` number of points linear predicted and appended to the original data.

See also:
**1p** 1D or 2D linear prediction extrapolation.

**Notes**

In forward-backward or backward-forward mode root stabilizing is done on both sets of signal roots as calculated in the first mode direction. After averaging the coefficient the roots are again stabilized.

When the append parameter does not match the LP mode, for example if a backward linear prediction (mode='b') is used to predict points after the trace (append='after'), any root fixing is done before reversing the filter.

```python
nmrglue.process.proc_lp.extrapolate_2d
```

Extrapolate points along the 1st axis using the lp2d algorithm.

```python
nmrglue.process.proc_lp.make_lp2d_Dd
```

Form the lp2d equation matrix and vector.

```python
nmrglue.process.proc_lp.cadzow_single
```

Perform a single iteration of Cadzow signal enhancement on a 1D vector

See `cadzow()` for documentation.

```python
nmrglue.process.proc_lp.root2damp
```

Calculate the damping factor from a LP root

```python
nmrglue.process.proc_lp.root2freq
```

Calculate the frequency from a LP root

```python
nmrglue.process.proc_lp.cof2amp
```

Calculate a signal amplitude from a model coefficient

```python
nmrglue.process.proc_lp.cof2phase
```

Calculate a signal phase from a model coefficient
nmrglue.process.proc_lp.make_D

nmrglue.process.proc_lp.make_D(x, order, mode)
Make the LP equation D matrix (Da = d’)

nmrglue.process.proc_lp.make_little_d

nmrglue.process.proc_lp.make_little_d(x, order, mode)
Make the LP equation d’ vector (Da = d’)

nmrglue.process.proc_lp.make_Dd

nmrglue.process.proc_lp.make_Dd(x, order, mode)
make the LP equation D matrix and d’ vector (Da=d’)

nmrglue.process.proc_lp.make_mirror

nmrglue.process.proc_lp.make_mirror(x, mode)
Make a mirror image trace.

Reflects trace over zero as described in: G. Zhu and A. Bax, Journal of Magnetic Resonance, 1990, 90, 405
When mode is “0” (no initial delay) form the an array with length 2N-1: x_{n-1} ... x_1 x_0 x_1 ... x_{n-1}
When mode is “180” (half point delay) form an array with length 2N: x_{n-1} ... x_1 x_0 x_0 x_1 ... x_{n-1}

Parameters x: ndarray
1D array to form mirrored trace from.
mode: {'180', '0'}
Mirror mode, see above.

nmrglue.process.proc_lp.find_lpc

nmrglue.process.proc_lp.find_lpc(D, d, method)
Find linear prediction filter using a provided method.

nmrglue.process.proc_lp.find_lpc_svd

nmrglue.process.proc_lp.find_lpc_svd(D, d)
Find linear prediction filter using single value decomposition.

nmrglue.process.proc_lp.pinv_diagsvd

nmrglue.process.proc_lp.pinv_diagsvd(s, m, L)
Construct the pseudo-inverse of the sigma matrix from singular values
nmrglue.process.proc_LP.find_lpc_qr

nmrglue.process.proc_LP.find_lpc_qr(D, d)
Find linear prediction filter using QR decomposition.

nmrglue.process.proc_LP.find_lpc_cholesky

nmrglue.process.proc_LP.find_lpc_cholesky(D, d)
Find linear prediction filter using a Cholesky decomposition.

nmrglue.process.proc_LP.find_lpc_tls

nmrglue.process.proc_LP.find_lpc_tls(D, d)
Find linear prediction filter using the Total Least Squares method

nmrglue.process.proc_LP.find_lpc_fb

nmrglue.process.proc_LP.find_lpc_fb(x, order, bad_roots, fix_mode, method)
Determin LP coefficient using forward-backward linear prediction.
Averages LP coefficients generated from solving the forward and backward linear prediction equations after reversing the roots of characteristic polynomial of the backward solution. Method is described in: G. Zhu and A. Bax, Journal of Magnetic Resonance, 1992, 100, 202-207.
Description of parameters can be found in lp().

nmrglue.process.proc_LP.find_lpc_bf

nmrglue.process.proc_LP.find_lpc_bf(x, order, bad_roots, fix_mode, method)
Determin LP coefficient using backward-forward linear prediction.
Averages LP coefficients generated from solving the forward and backward linear prediction equations after reversing the roots of characteristic polynomial of the forward solution. Similar to method described in: G. Zhu and A. Bax, Journal of Magnetic Resonance, 1992, 100, 202-207.
Description of parameters can be found in lp() function.

nmrglue.process.proc_LP.find_lproots_hsvd

nmrglue.process.proc_LP.find_lproots_hsvd(x, M, K, mode, zmethod='sm')
Find LP roots (poles) using the HSVD method
Perform a HSVD linear prediction to determine signal roots (poles) as described in: Barkhuijsen, DeBeer, and Van Ormondt, JMR, 1987, 73, 553
Parameters x, M and K are the same as those described in the above article. zmethod refer to the method used to calculate Z', either a least-squares method (lstsq) can be used to solve U_b*Z'=U_t or the Sherman-Morrison formula (sm) can be used to avoid the full matrix inversion with equation [12] being used to find Z'. The Sherman-Morrison method should be faster with similar precision.

Parameters x : 1D ndarray
1D trace of NMR data in the time domain, the FID.

3.2. process modules
M : int
Length (M+1) of data matrix to form.

K : int
Reduced prediction order (number of signal roots) Must be less than the smaller of M + 1 or len(x) - M.

mode : {'f', 'b'}
Mode to perform LP. ‘f’ for forward,’b’ for backward.

zmethod : {'lstsq', 'sm'}
Method used to find Z’ ‘lstsq’ for least squares, ‘sm’ for Sherman-Morrison.

Returns y : ndarray
Array of signal roots (poles)

nmrglue.process.proc_lp.find_roots

nmrglue.process.proc_lp.find_roots(a, mode='f')
Find LP roots (poles) from a set of LP coefficients.

Parameters a : array
LP coefficients.

mode : {'f', 'b'}
Mode of LP coefficients. ‘f’ for coefficients ordered m, m - 1,..., 1. ‘b’ for coefficients ordered 1, 2,..., m

Returns roots : array
LP roots (poles)

nmrglue.process.proc_lp.find_coeff

nmrglue.process.proc_lp.find_coeff(poles, mode='f')
Find LP coefficients from a set of LP roots (poles).

Parameters poles : ndarray
Array of LP roots (poles)

mode : {'f', 'b'}
Mode in which LP coefficients should be returned. ‘f’ for coefficients ordered m, m - 1,..., 1. ‘b’ for coefficients ordered 1, 2,..., m.

Returns c : ndarray
LP coefficients ordered according to mode.

nmrglue.process.proc_lp.reverse_filter

nmrglue.process.proc_lp.reverse_filter(a, mode)
Reverse a filter (change forward LP to backwards LP).
**nmrglue.process.proc_lp.fix_roots**

`nmrglue.process.proc_lp.fix_roots(poles, fix_roots='incr', fix_mode='reflect')`

Fix (stabilize) LP roots.

**Parameters**
- **poles**: ndarray
  - Array of LP roots (poles).
- **fix_roots**: {'incr', 'decr'}
  - Type of roots which to consider bad and to stabilize. Either those with increasing signals 'incr' or decreasing signals 'decr'.
- **fix_mode**: {'on', 'reflect'}
  - Method used to stabilize bad roots, 'on' to move the roots onto the unit circle, 'reflect' to reflect bad roots across the unit circle.

**Returns**
- **npoles**: ndarray
  - Array of stabilized LP roots (poles).

**nmrglue.process.proc_lp.extrapolate**

`nmrglue.process.proc_lp.extrapolate(trace, a, pred, append)`

Extrapolate points using LP prediction filter.

**Parameters**
- **trace**: 1D ndarray
  - 1D array to extrapolate from and append to.
- **a**: ndarray
  - LP coefficients, must be ordered according to direction of extrapolation.
- **pred**: int
  - Number of points to predict using LP.
- **append**: {'a', 'b'}
  - Location to append new points, 'a' for after the current data, 'b' for before the current data.

**Returns**
- **ntrace**: 1D ndarray
  - 1D array with extrapolated points appended

### 3.2.5 nmrglue.pipe_proc

NMRPipe like processing functions for use with the `nmrglue.fileio.pipe` module.

These functions attempt to mimic NMRPipe’s processing functions but small differences exist between to two implementations. In particular when using this module:

- **hdr=True** overrides all values in the calling function.
- A di flag is not used, rather the `di()` function should be used to delete the imaginary portion of a spectra.
- x1, xn and other limits must be expressed in points. A unit conversion object function should be used before calling the processing function to calculate these values.
• No functions implement the dmx or nodmx flags.

Additional differences from NMRPipe’s functions are documented in the individual processing functions.

The following functions have not been implemented and will raise a NotImplemented exception:

• ann Fourier Analysis by Neural Net
• ebs EBS Reconstruction
• mem Maximum Entropy
• ml Maximum likelihood frequency
• poly Polynomail baseline correction
• xyz2zyx 3D matrix transpose
• ztp 3D matrix transpose

This module is imported as nmrglue.pipe_proc and can be called as such.

**Apodization**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>apod</code></td>
<td>Generic apodization.</td>
</tr>
<tr>
<td><code>em</code></td>
<td>Exponential apodization.</td>
</tr>
<tr>
<td><code>gm</code></td>
<td>Lorentz-to-Gauss apodization.</td>
</tr>
<tr>
<td><code>gmb</code></td>
<td>Modified Gaussian Apodization.</td>
</tr>
<tr>
<td><code>jmod</code></td>
<td>Exponentially Damped J-Modulation Apodation.</td>
</tr>
<tr>
<td><code>sp</code></td>
<td>Sine bell apodization.</td>
</tr>
<tr>
<td><code>sine</code></td>
<td>Sine bell apodization.</td>
</tr>
<tr>
<td><code>tm</code></td>
<td>Trapezoid apodization.</td>
</tr>
<tr>
<td><code>tri</code></td>
<td>Triangular apodization.</td>
</tr>
</tbody>
</table>

nmrglue.process.pipe_proc.apod

```python
def apod(dic, data[, qName, q1, q2, q3, c, ...])
```

Generic apodization.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dic</td>
<td>dict</td>
<td>Dictionary of NMRPipe parameters.</td>
</tr>
<tr>
<td>data</td>
<td>ndarray</td>
<td>Array of NMR data.</td>
</tr>
<tr>
<td>qName</td>
<td><code>{'SP', 'EM', 'GM', 'GMB', 'TM', 'TRI', 'JMOD'}</code></td>
<td>Abbreviation of apodization function the apply. See the specific apodization function for a description.</td>
</tr>
<tr>
<td>q1</td>
<td>float</td>
<td>First apodization function parameter. See specific apodization function for details.</td>
</tr>
<tr>
<td>q2</td>
<td>float</td>
<td>Second apodization function parameter. See specific apodization function for details.</td>
</tr>
<tr>
<td>q3</td>
<td>float</td>
<td></td>
</tr>
</tbody>
</table>

nmrglue Documentation, Release 0.7-dev
Third apodization function parameter. See specific apodization function for details.

c : float
   First point scale value.

start : int, optional
   Starting location of apodization window. Default is the first point, 1.

size : int, optional
   Size of the apodization window. Default (‘default’) is the full size of the active dimension.

inv : bool, optional
   True for inverse apodization, False for normal apodization.

one : bool, optional
   True to set points outside of window to 1. False leaves points outside the apodization window as is.

hdr : bool, optional
   True to read apodization parameters from the the parameters in dic.

Returns ndic : dict
   Dictionary of updated NMRPipe parameters.

ndata : ndarray
   Array of NMR data with apodization applied.

See also:

   em Exponential apodization.
   gm Lorentz-to-Gauss apodization.
   gmb Modified Gaussian apodization.
   jmod Exponentially damped J-modulation apodization.
   sp Sine bell apodization.
   tm Trapezoid apodization.
   tri Triangular apodization.

nmrglue.process.pipe_proc.em

nmrglue.process.pipe_proc.em(dic, data, lb=0.0, c=1.0, start=1, size='default', inv=False, one=False, hdr=False)

Exponential apodization.

Parameters dic : dict
   Dictionary of NMRPipe parameters.

data :.ndarray
   Array of NMR data.

lb : float
Exponential line broadening in Hz.

c : float
    First point scale value.

start : int, optional
    Starting location of apodization window. Default is the first point, 1.

size : int, optional
    Size of the apodization window. Default (‘default’) is the full size of the active dimension.

inv : bool, optional
    True for inverse apodization, False for normal apodization.

one : bool, optional
    True to set points outside of window to 1. False leaves points outside the apodization window as is.

hdr : bool, optional
    True to read apodization parameters from the the parameters in dic.

Returns ndic : dict
    Dictionary of updated NMRPipe parameters.

ndata : ndarray
    Array of NMR data with exponential apodization applied.

**nmrglue.process.pipe_proc.gm**

```
        nmrglue.process.pipe_proc.gm(dic, data, g1=0.0, g2=0.0, g3=0.0, c=1.0, start=1, size='default', inv=False, one=False, hdr=False)
```

Lorentz-to-Gauss apodization

Parameters dic : dict
    Dictionary of NMRPipe parameters.

data : ndarray
    Array of NMR data.

g1 : float
    Inversion exponential width in Hz.

g2 : float
    Gaussian broadening width in Hz.

g3 : float
    Location of Gaussian maximum, should be between 0.0 and 1.0.

c : float
    First point scale value.

start : int, optional
Starting location of apodization window. Default is the first point, 1.

size : int, optional

Size of the apodization window. Default (‘default’) is the full size of the active dimension.

inv : bool, optional

True for inverse apodization, False for normal apodization.

one : bool, optional

True to set points outside of window to 1. False leaves points outside the apodization window as is.

hdr : bool, optional

True to read apodization parameters from the the parameters in dic.

Returns ndic : dict

Dictionary of updated NMRPipe parameters.

ndata : ndarray

Array of NMR data with lorentz-to-gauss apodization applied.

nmrglue.process.pipe_proc.gmb

nmrglue.process.pipe_proc.gmb(dic, data, lb=0.0, gb=0.0, c=1.0, start=1, size='default', inv=False, one=False, hdr=False)

Modified Gaussian Apodization

Parameters dic : dict

Dictionary of NMRPipe parameters.

data : ndarray

Array of NMR data.

lb : float

Exponential apodization term in Hz.

gb : float

Gaussian apodization term in Hz.

c : float

First point scale value.

start : int, optional

Starting location of apodization window. Default is the first point, 1.

size : int, optional

Size of the apodization window. Default (‘default’) is the full size of the active dimension.

inv : bool, optional

True for inverse apodization, False for normal apodization.

one : bool, optional
True to set points outside of window to 1. False leaves points outside the apodization window as is.

hdr : bool, optional

True to read apodization parameters from the the parameters in dic.

Returns
ndic : dict

Dictionary of updated NMRPipe parameters.

ndata : ndarray

Array of NMR data with a modified gaussian apodization applied.

nmrglue.process.pipe_proc.jmod

nmrglue.process.pipe_proc.jmod(dic, data, off=0.0, j=0.0, lb=0.0, sin=False, cos=False, c=1.0, start=1, size='default', inv=False, one=False, hdr=False)

Exponentially Damped J-Modulation Apodization

Parameters
dic : dict

Dictionary of NMRPipe parameters.

data : ndarray

Array of NMR data.

off : float

Starting location of J-modulation in a fractions of pi radians. This parameter is ignored if sin or cos parameters are True.

j : float

J-modulation in Hz.

lb :

Exponentntial line broadening in Hz.

sin : bool

True for sine modulation, off parameter is ignored.

cos : bool

True for cosine modulation, off parameter is ignored.

c : float

First point scale value.

start : int, optional

Starting location of apodization window. Default is the first point, 1.

size : int, optional

Size of the apodization window. Default (‘default’) is the full size of the active dimension.

inv : bool, optional

True for inverse apodization, False for normal apodization.

one : bool, optional
True to set points outside of window to 1. False leaves points outside the apodization window as is.

hdr : bool, optional
    True to read apodization parameters from the the parameters in dic.

Returns ndic : dict
    Dictionary of updated NMRPipe parameters.

ndata : ndarray
    Array of NMR data with a exponentially damped J-modulation apodization applied.

nmrglue.process.pipe_proc.sp

nmrglue.process.pipe_proc.sp(dic, data, off=0.0, end=1.0, pow=1.0, c=1.0, start=1, size='default', inv=False, one=False, hdr=False)

Sine bell apodization.

Parameters dic : dict
    Dictionary of NMRPipe parameters.

data : ndarray
    Array of NMR data.

off : float
    Starting location of sine-bell as a fraction of pi radians.

d : float
    Ending location of sine-bell as a fraction of pi radians.

pow : int
    Sine-bell power.

c : float
    First point scale value.

start : int, optional
    Starting location of apodization window. Default is the first point, 1.

size : int, optional
    Size of the apodization window. Default (`default`) is the full size of the active dimension.

inv : bool, optional
    True for inverse apodization, False for normal apodization.

one : bool, optional
    True to set points outside of window to 1. False leaves points outside the apodization window as is.

hdr : bool, optional
    True to read apodization parameters from the the parameters in dic.

Returns ndic : dict
Dictionary of updated NMRpipe parameters.

\textbf{ndata} : ndarray

Array of NMR data with a sine-bell apodization applied.

\textbf{nmrglue.process.pipe_proc}sine

\texttt{nmrglue.process.pipe_proc.sine}(\texttt{dic}, \texttt{data}, \texttt{off}=0.0, \texttt{end}=1.0, \texttt{pow}=1.0, \texttt{c}=1.0, \texttt{start}=1, \texttt{size}='default', \texttt{inv}=\texttt{False}, \texttt{one}=\texttt{False}, \texttt{hdr}=\texttt{False})

Sine bell apodization.

\textbf{Parameters} \textbf{dic} : dict

Dictionary of NMRPipe parameters.

\textbf{data} : ndarray

Array of NMR data.

\textbf{off} : float

Starting location of sine-bell as a fraction of pi radians.

\textbf{end} : float

Ending location of sine-bell as a fraction of pi radians.

\textbf{pow} : int

Sine-bell power.

\textbf{c} : float

First point scale value.

\textbf{start} : int, optional

Starting location of apodization window. Default is the first point, 1.

\textbf{size} : int, optional

Size of the apodization window. Default (‘default’) is the full size of the active dimension.

\textbf{inv} : bool, optional

True for inverse apodization, False for normal apodization.

\textbf{one} : bool, optional

True to set points outside of window to 1. False leaves points outside the apodization window as is.

\textbf{hdr} : bool, optional

True to read apodization parameters from the the parameters in \texttt{dic}.

\textbf{Returns} \textbf{ndic} : dict

Dictionary of updated NMRPipe parameters.

\textbf{ndata} : ndarray

Array of NMR data with a sine-bell apodization applied.
nmrglue.process.pipe_proc.tm

nmrglue.process.pipe_proc.tm(dic, data, t1=0.0, t2=0.0, c=1.0, start=1, size='default', inv=False, one=False, hdr=False)

Trapezoid apodization.

Parameters
dic : dict
  Dictionary of NMRPipe parameters.
data : ndarray
  Array of NMR data.
t1 : float
  Length in points of left side of the trapezoid.
t2 : float
  Length in points of right side of the trapezoid.
c : float
  First point scale value.
start : int, optional
  Starting location of apodization window. Default is the first point, 1.
size : int, optional
  Size of the apodization window. Default ('default') is the full size of the active dimension.
inv : bool, optional
  True for inverse apodization, False for normal apodization.
one : bool, optional
  True to set points outside of window to 1. False leaves points outside the apodization window as is.
hdr : bool, optional
  True to read apodization parameters from the the parameters in dic.

Returns
ndic : dict
  Dictionary of updated NMRPipe parameters.
ndata : ndarray
  Array of NMR data with a trapezoid apodization applied.

nmrglue.process.pipe_proc.tri

nmrglue.process.pipe_proc.tri(dic, data, loc='auto', lHi=0.0, rHi=0.0, c=1.0, start=1, size='default', inv=False, one=False, hdr=False)

Triangular apodization

Parameters
dic : dict
  Dictionary of NMRPipe parameters.
data : ndarray

Array of NMR data.

loc : int or “auto”

Location in points of triangle apex. The default (“auto”) is to place the apex in the middle.

lHi : float

Starting height of the left side of the triangle.

rHi : float

Starting height of the right side of the triangle.

c : float

First point scale value.

start : int, optional

Starting location of apodization window. Default is the first point, 1.

size : int, optional

Size of the apodization window. Default (‘default’) is the full size of the active dimension.

inv : bool, optional

True for inverse apodization, False for normal apodization.

one : bool, optional

True to set points outside of window to 1. False leaves points outside the apodization window as is.

hdr : bool, optional

True to read apodization parameters from the the parameters in dic.

Returns ndic : dict

Dictionary of updated NMRPipe parameters.

ndata : ndarray

Array of NMR data with a triangular apodization applied.

Notes

The right side of the apodization is differs slightly from NMRPipe’s tri function.

Shifts

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs(dic, data[, rs, sw])</td>
<td>Right shift and zero pad.</td>
</tr>
<tr>
<td>ls(dic, data[, ls, sw])</td>
<td>Left Shift and Zero Pad</td>
</tr>
<tr>
<td>cs(dic, data, dir[, pts, neg, sw])</td>
<td>Circular shift</td>
</tr>
<tr>
<td>fsh(dic, data, dir, pts[, sw])</td>
<td>Frequency shift.</td>
</tr>
</tbody>
</table>
nmrglue.process.pipe_proc.rs

nmrglue.process.pipe_proc.rs(dic, data, rs=0.0, sw=False)
Right shift and zero pad.

Parameters
dic : dict
  Dictionary of NMRPipe parameters.
data : ndarray
  Array of NMR data.rs : float
  Number of points to right shift. Negative values will left shift.
sw : bool
  True to update chemical shift calibration parameters.

Returns
ndic : dict
  Dictionary of updated NMRPipe parameters.
data : ndarray
  Array of NMR data which has been right shifted.

nmrglue.process.pipe_proc.ls

nmrglue.process.pipe_proc.ls(dic, data, ls=0.0, sw=False)
Left Shift and Zero Pad.

Parameters
dic : dict
  Dictionary of NMRPipe parameters.
data : ndarray
  Array of NMR data.ls : float
  Number of points to left shift. Negative values will right shift.
sw : bool
  True to update chemical shift calibration parameters.

Returns
ndic : dict
  Dictionary of updated NMRPipe parameters.
data : ndarray
  Array of NMR data which has been left shifted.

nmrglue.process.pipe_proc.cs

nmrglue.process.pipe_proc.cs(dic, data, dir, pts=0.0, neg=False, sw=False)
Circular shift.
  The syntax of this function is different from NMRPipe’s CS function.

3.2. process modules
Parameters  

**dic**: dict  
Dictionary of NMRPipe parameters.

**data**: ndarray  
Array of NMR data.

**dir**: ['rs' or 'ls']  
Direction to shift spectra, 'rs' for right shifting, 'ls' for left shifting.

**pts**: float  
Number of points to shift.

**neg**: bool  
True to negative points which are shifted.

**sw**: bool  
True to update chemical shift calibration parameters.

Returns  

**ndic**: dict  
Dictionary of updated NMRPipe parameters.

**ndata**: ndarray  
Array of NMR data which has been circular shifted.

```python
import nmrglue.process.pipe_proc

# Define parameters
dic = {  
    'param1': value1,  
    'param2': value2  
}

data = np.array([values])  
dir = 'rs'  
pts = 10.0  
sw = True

# Call function
ndic, ndata = nmrglue.process.pipe_proc.fsh(dic, data, dir, pts, sw=sw)
```

The `fsh` function performs a frequency shift on the NMR data. It takes in a dictionary of parameters and an array of NMR data, and returns a dictionary of updated parameters and an array of circular shifted NMR data.

**nmrglue.process.pipe_proc.fsh**

Frequency shift.

Parameters  

**dic**: dict  
Dictionary of NMRPipe parameters.

**data**: ndarray  
Array of NMR data.

**dir**: ['rs' or 'ls']  
Direction to shift spectra, 'rs' for right shifting, 'ls' for left shifting.

**pts**: float  
Number of points to shift.

**sw**: bool  
True to update chemical shift calibration parameters.

Returns  

**ndic**: dict  
Dictionary of updated NMRPipe parameters.

**ndata**: ndarray  
Array of NMR data which has been frequency shifted.

```python
import nmrglue.process.pipe_proc

# Define parameters
dic = {  
    'param1': value1,  
    'param2': value2  
}

data = np.array([values])  
dir = 'rs'  
pts = 10.0  
sw = True

# Call function
ndic, ndata = nmrglue.process.pipe_proc.fsh(dic, data, dir, pts, sw=sw)
```

The `fsh` function performs a frequency shift on the NMR data. It takes in a dictionary of parameters and an array of NMR data, and returns a dictionary of updated parameters and an array of frequency shifted NMR data.
Notes

This function does not perform a Hilbert transform when data is complex, NMRPipe’s FSH function appears to. As such the results of the imaginary channel differs from NMRPipe. In addition MAX/MIN value are slightly different than those in NMRPipe.

Transforms

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft(dic, data[, auto, real, inv, alt, neg, ...])</td>
<td>Complex Fourier transform.</td>
</tr>
<tr>
<td>rft(dic, data[, inv])</td>
<td>Real Fourier transform.</td>
</tr>
<tr>
<td>ha(dic, data[, inv])</td>
<td>Hadamard transform.</td>
</tr>
<tr>
<td>ht(dic, data[, mode, zf, td, auto])</td>
<td>Hilbert transform.</td>
</tr>
</tbody>
</table>

nmrglue.process.pipe_proc.ft

Complex Fourier transform.

Parameters

dic : dict
    Dictionary of NMRPipe parameters.

data : ndarray
    Array of NMR data.

auto : bool
    True will choose mode automatically, not recomended.

real : bool
    True to transform real-only data.

inv : bool
    True to perform an inverse transform.

alt : bool
    True to alternative the sign of points before transforming.

neg : bool
    True will negate the imaginary channel before transforming.

null : bool
    True will not apply transform but will update the parameter dictionary.

bruk : bool
    True to process Redfield sequential data, this is the same as setting alt and real to True.

debug : bool
    True to print debug info.

Returns

ndic : dict
    Dictionary of updated NMRPipe parameters.
ndata : ndarray

Array of NMR data which has been Fourier transformed.

Notes

Choosing multiply conflicting modes can produces results different from NMRPipe’s FT function.

nmrglue.process.pipe_proc.rft

nmrglue.process.pipe_proc.rft(dic, data, inv=False)
Real Fourier transform.

Parameters
dic : dict
    Dictionary of NMRPipe parameters.
data : ndarray
    Array of NMR data.
inv : bool
    True to perform an inverse transform.

Returns
ndic : dict
    Dictionary of updated NMRPipe parameters.
ndata : ndarray
    Array of NMR data which has been real Fourier transformed.

Notes

This function gives results which slightly differ from NMRPipe’s RFT function in some cases.

nmrglue.process.pipe_proc.ha

nmrglue.process.pipe_proc.ha(dic, data, inv=False)
Hadamard transform.

Parameters
dic : dict
    Dictionary of NMRPipe parameters.
data : ndarray
    Array of NMR data.
inv : bool
    True to perform an inverse transform.

Returns
ndic : dict
    Dictionary of updated NMRPipe parameters.
ndata : ndarray
    Array of NMR data which has been real Fourier transformed.
Array of NMR data which has been Hadamard transformed.

**Notes**

This function is slow. Implemented a FWHT in proc_base would significantly improve the speed of this functions.

### nmrglue.process.pipe_proc.ht

```python
nmrglue.process.pipe_proc.ht(dic, data, mode='ps0-0', zf=False, td=False, auto=False)
```

Hilbert transform.

**Parameters**

- **dic**: dict
  - Dictionary of NMRPipe parameters.
- **data**: ndarray
  - Array of NMR data.
- **mode**: {'ps0-0', 'ps90-180'}
  - Mirror image mode.
- **zf**: bool
  - True to zero fill before transform for speed.
- **td**: bool
  - True to set the time-domain parameter to half size.
- **auto**: bool
  - True to select mode and zf parameters automatically from dic.

**Returns**

- **ndic**: dict
  - Dictionary of updated NMRPipe parameters.
- **ndata**: ndarray
  - Array of NMR data which has been Hilbert transformed.

**Notes**

“ps90-180” mirror image mode gives different results than NMRPipe’s HT function.

### Standard NMR

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>di(dic, data)</code></td>
<td>Delete imaginaries</td>
</tr>
<tr>
<td><code>ps(dic, data[, p0, p1, inv, hdr, noup, ht, ...])</code></td>
<td>Phase shift</td>
</tr>
<tr>
<td><code>tp(dic, data[, hyper, nohyper, auto, nohdr])</code></td>
<td>Transpose data (2D).</td>
</tr>
<tr>
<td><code>zf(dic, data[, zf, pad, size, mid, inter, ...])</code></td>
<td>Zero fill</td>
</tr>
</tbody>
</table>
nmrglue.process.pipe_proc.di

nmrglue.process.pipe_proc.di(dic, data)
Delete imaginaries

Parameters
dic : dict
    Dictionary of NMRPipe parameters.
data : ndarray
    Array of NMR data.

Returns
ndic : dict
    Dictionary of updated NMRPipe parameters.
data : ndarray
    Array of NMR data from which imaginaries have been removed.

nmrglue.process.pipe_proc.ps

nmrglue.process.pipe_proc.ps(dic, data, p0=0.0, p1=0.0, inv=False, hdr=False, noup=False, ht=False, zf=False, exp=False, tc=0.0)
Phase shift

Parameters
dic : dict
    Dictionary of NMRPipe parameters.
data : ndarray
    Array of NMR data.
p0 : float
    Zero order phase in degrees.
p1 : float
    First order phase in degrees.
inv : bool
    True to perform inverse phase correction.
hdr : bool
    True to use phasing parameters from dic.
noup : bool
    True to not update phasing parameters in returned ndic.
ht : bool
    True to perform a Hilbert transform to reconstruction imaginaries before phasing.
zf : bool
    True to zero fill before applied Hilbert transform.
exp : bool
    True to perform exponential phase correction. False performs linear phase correction.
**tc**: float, optional

Exponential decay constant. User when exp is True.

**Returns**

- **ndic**: dict
  
  Dictionary of updated NMRpipe parameters.

- **ndata**: ndarray
  
  Array of NMR data which has been phased.

**Notes**

When `inv` is True this function will correctly invert an exponential phase correction, NMRPipe’s PS function does not. In addition, FDFNP0 and FDFNP1 are updated unless `noup=True`. There are not `rs` and `ls` parameter, if the data need to be shifted before phasing use the `rs()` or `ls` function before using this function.

### nmrglue.process.pipe_proc.tp

```python
nmrglue.process.pipe_proc.tp(dic, data, hyper=False, nohyper=False, auto=False, nohdr=False)
```

Transpose data (2D).

**Parameters**

- **dic**: dict
  
  Dictionary of NMRPipe parameters.

- **data**: ndarray
  
  Array of NMR data.

- **hyper**: bool
  
  True to perform hypercomplex transpose.

- **nohyper**: bool
  
  True to suppress hypercomplex transpose.

- **auto**: bool
  
  True to choose transpose mode automatically.

- **nohdr**: bool
  
  True to not update the transpose parameters in ndic.

**Returns**

- **ndic**: dict
  
  Dictionary of updated NMRPipe parameters.

- **ndata**: ndarray
  
  Array of NMR data which has been transposed.

### nmrglue.process.pipe_proc.zf

```python
nmrglue.process.pipe_proc.zf(dic, data, zf=1, pad='auto', size='auto', mid=False, inter=False, auto=False, inv=False)
```

Zero fill

**Parameters**

- **dic**: dict
Dictionary of NMRPipe parameters.

data : ndarray
    Array of NMR data.

zf : int, optional.
    Number of times to double the current dimensions size.

pad : int
    Number of zeros to pad the data with.

size : int
    Desired final size of the current dimension.

mid : bool
    True to zero fill in the middle of the current dimension

inter : bool
    True to zero fill between points.

auto : bool
    True to round final size to nearest power of two.

inv : bool
    True to extract the time domain data (remove zero filling).

Returns ndic : dict
    Dictionary of updated NMRPipe parameters.

ndata : ndarray
    Array of NMR data which has zero filled.

Notes

Only one of the zf, pad and size parameter should be used, the other should be left as the default value. If any of the mid, inter, auto and inv parameters are True other parameter may be ignored.

Baseline

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>Linear baseline correction.</td>
</tr>
<tr>
<td>cbf</td>
<td>Constant baseline correction.</td>
</tr>
<tr>
<td>med</td>
<td>Median baseline correction</td>
</tr>
<tr>
<td>sol</td>
<td>Solvent filter</td>
</tr>
</tbody>
</table>

nmrglue.process.pipe_proc.base

nmrglue.process.pipe_proc.base (dic, data[, nl=None, nw=0, first=False, last=False])
    Linear baseline correction.

Parameters dic : dict
    Dictionary of NMRPipe parameters.
**data** : ndarray
  Array of NMR data.

**nl** : list
  List of baseline node points.

**nw** : int
  Node width in points.

**first** : bool
  True to include the first point in the node list.

**last** : bool
  True to include the last point in the node list.

**Returns ndic** : dict
  Dictionary of updated NMRPipe parameters.

**ndata** : ndarray
  Array of NMR data with a linear baseline correction applied.

**nmrglue.process.pipe_proc.cbf**

*nmrglue.process.pipe_proc.cbf*(dic, data, last=10, reg=False, slice=slice(None, None, None))

Constant baseline correction.

**Parameters dic** : dict
  Dictionary of NMRPipe parameters.

**data** : ndarray
  Array of NMR data.

**last** : float
  Percentage of trace to use in calculating baseline correction.

**reg** : slice object, optional
  Python slice object describing region(s) from which to calculate the baseline correction.
  If False (default) the last parameter will be used to calculate the correction.

**slice** : slice object
  Python slice describing regions to apply the baseline correction to.

**Returns ndic** : dict
  Dictionary of updated NMRPipe parameters.

**ndata** : ndarray
  Array of NMR data with a constant baseline correction applied.
Notes

The parameters of this function differ significantly from NMRPipe’s cbf function. The parameters `ref` and `slice` are Python slice objects if explicit correction regions are desired. The `noseq` and `nodmx` parameters are not implemented.

**nmrglue.process.pipe_proc.med**

```python
nmrglue.process.pipe_proc.med(dic, data, nw=24, sf=16, sigma=5.0)
```

Median baseline correction

**Parameters**
- `dic` : dict
  Dictionary of NMRPipe parameters.
- `data` : ndarray
  Array of NMR data.
- `nw` : int
  Median window size in points.
- `sf` : int
  Smoothing filter size in points.
- `sigma` : float
  Gaussian convolution width.

**Returns**
- `ndic` : dict
  Dictionary of updated NMRPipe parameters.
- `ndata` : ndarray
  Array of NMR data with a median baseline correction applied.

Notes

This function applies Friendrich’s model-free baseline flattening algorithm (Friendrichs JBNMR 1995 5 147-153). NMRPipe applies a different and unknown algorithm.

**nmrglue.process.pipe_proc.sol**

```python
nmrglue.process.pipe_proc.sol(dic, data, mode='low', fl=16, fs=1, head=0)
```

Solvent filter

**Parameters**
- `dic` : dict
  Dictionary of NMRPipe parameters.
- `data` : ndarray
  Array of NMR data.
- `mode` : `{'low'}`
  Filter mode. Currently only ‘low’ is implemented.
**nmrglue Documentation, Release 0.7-dev**

- **fl**: int
  
  Length of filter in points.

- **fs**: {1, 2, 3}
  
  Shape of lowpass filter 1: boxcar, 2: sine 3: sine squared.

- **head**: :
  
  Number of points to skip when applying filter.

**Returns**

- **ndic**: dict
  
  Dictionary of updated NMRPipe parameters.

- **ndata**: ndarray
  
  Array of NMR data with a solvent filter applied.

**Notes**

This different from NMRPipe’s SOL function in the only the low pass filter has been implemented. In addition the **mir**, **noseq** and **nodmx** parameters are not implemented.

**Utilities**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(dic, data[, r, i, c, ri, x1, xn])</code></td>
<td>Add a constant</td>
</tr>
<tr>
<td><code>dx(dic, data)</code></td>
<td>Derivative by central difference.</td>
</tr>
<tr>
<td><code>ext(dic, data[, x1, xn, y1, yn, round, ...])</code></td>
<td>Extract a region.</td>
</tr>
<tr>
<td><code>integ(dic, data)</code></td>
<td>Integral by simple sum</td>
</tr>
<tr>
<td><code>mc(dic, data[, mode])</code></td>
<td>Modules or magnitude calculation.</td>
</tr>
<tr>
<td><code>mir(dic, data[, mode, invl, invr, sw])</code></td>
<td>Append mirror image.</td>
</tr>
<tr>
<td><code>mult(dic, data[, r, i, c, inv, hdr, x1, xn])</code></td>
<td>Multiple by a constant.</td>
</tr>
<tr>
<td><code>rev(dic, data[, sw])</code></td>
<td>Reverse data.</td>
</tr>
<tr>
<td><code>set(dic, data[, r, i, c, x1, xn])</code></td>
<td>Set data to a constant.</td>
</tr>
<tr>
<td><code>shuf(dic, data[, mode])</code></td>
<td>Shuffle Utilities</td>
</tr>
<tr>
<td><code>sign(dic, data[, ri, r, i, left, right, ...])</code></td>
<td>Sign manipulation utilities</td>
</tr>
</tbody>
</table>

**nmrglue.process.pipe_proc.add**

```
```

Add a constant

**Parameters**

- **dic**: dict
  
  Dictionary of NMRPipe parameters.

- **data**: ndarray
  
  Array of NMR data.

- **r**: float
  
  Constant to add to real data.

- **i**: float
  
  Constant to add to imaginary data.

**3.2. process modules**
**nmrglue Documentation, Release 0.7-dev**

**c** : float
Constant to add to both real and imaginary data.

**ri** : bool
True to add real and imaginary data into real channel.

**xl** : int
First point of region to add constant to.

**xn** : int or ‘default’
Last point of region to add constant to. ‘default’ specifies the last point.

**Returns**

**ndic** : dict
Dictionary of updated NMRPipe parameters.

**ndata** : ndarray
Array of NMR data with a constant added.

**Notes**

Parameter *c* is added to the real and imaginary data even when *r* and *i* are defined. NMRPipe’s ADD function ignores *c* when *r* or *i* are defined.

### nmrglue.process.pipe_proc.dx

**nmrglue.process.pipe_proc.dx**

**Parameters**

**dic** : dict
Dictionary of NMRPipe parameters.

**data** : ndarray
Array of NMR data.

**Returns**

**ndic** : dict
Dictionary of updated NMRPipe parameters.

**ndata** : ndarray
Derivative of of NMR data.

### nmrglue.process.pipe_proc.ext

**nmrglue.process.pipe_proc.ext**

**Extract a region.**

**Parameters**

**dic** : dict
Dictionary of NMRPipe parameters.

**data** : ndarray

---

182 Chapter 3. Reference Guide
Array of NMR data.

**x1** : int or ‘default’
Starting point of the X-axis extraction. ‘default’ will start the extraction at the first point.

**xn** : int or ‘default’
Ending point of the X-axis extraction. ‘default’ will stop the extraction at the last point.

**y1** : int or ‘default’
Starting point of the Y-axis extraction. ‘default’ will start the extraction at the first point.

**yn** : int or ‘default’
Ending point of the Y-axis extraction. ‘default’ will stop the extraction at the last point.

**round** : int
Multiple to round extraction size to.

**left** : bool
True to extract the left half of the data.

**right** : bool
True to extract the right half of the data.

**mid** : bool
True to extract the central half of the data.

**pow2** : bool
True will round the extracted size to the nearest power of 2.

**sw** : bool
True to update the sweep width and ppm calibration parameters, recommended.

**Returns**

**ndic** : dict
Dictionary of updated NMRPipe parameters.

**ndata** : ndarray
Extracted region of NMR data.

**Notes**

The **time** parameter is not implemented. Using multiple conflicting parameters may result in different results than NMRPipe.

**nmrglue.process.pipe_proc.integ**

**nmrglue.process.pipe_proc.integ**(dic, data)
Integral by simple sum

**Parameters**

**dic** : dict
Dictionary of NMRPipe parameters.

**data** : ndarray
Array of NMR data.

**Returns**  
`ndic : dict`

Dictionary of updated NMRPipe parameters.

`ndata : ndarray`

Integrated NMR data.

---

**nmrglue.process.pipe_proc.mc**

`nmrglue.process.pipe_proc.mc(dic, data, mode='mod')`

Modules or magnitude calculation.

**Parameters**  
`dic : dict`

Dictionary of NMRPipe parameters.

`data : ndarray`

Array of NMR data.

`mode : {'mod' or 'pow'}`

‘mod’ to perform modules calculation, ‘pow’ to calculate square modules.

**Returns**  
`ndic : dict`

Dictionary of updated NMRPipe parameters.

`ndata : ndarray`

Modules of NMR data.

---

**nmrglue.process.pipe_proc.mir**

`nmrglue.process.pipe_proc.mir(dic, data, mode='left', invl=False, invr=False, sw=True)`

Append mirror image.

**Parameters**  
`dic : dict`

Dictionary of NMRPipe parameters.

`data : ndarray`

Array of NMR data.

`mode : {'left', 'right', 'center', 'ps90-180', 'pw0-0'}`

Type of mirror image to append.

`invl : bool`

True to negate left half.

`inrr : bool`

True to negate right half.

`sw : bool`

True to update ppm parameters, recommended.

**Returns**  
`ndic : dict`
Dictionary of updated NMRPipe parameters.

ndata : ndarray
    Array of NMR data with mirror image appended.

Notes

Negations of selected region are applied regardless of the mode selected.

nmrglue.process.pipe_proc.mult

nmrglue.process.pipe_proc.mult(dic, data, r=1.0, i=1.0, c=1.0, inv=False, hdr=False, x1=1.0, xn='default')

Multiple by a constant.

Parameters dic : dict
    Dictionary of NMRPipe parameters.

data : ndarray
    Array of NMR data.

r : float
    Constant to multiply real data by.

i : float
    Constant to multiply imaginary data by.

C : float
    Constant to multiply both real and imaginary data by.

inv : bool
    True to multiply by the inverse of the constant.

hdr : bool
    True to use constant defined in dic.

x1 : int
    First point of region to multiply constant by.

xn : int or 'default'
    Last point of region to multiply constant by. ‘default’ specifies the last point.

Returns ndic : dict
    Dictionary of updated NMRPipe parameters.

ndata : ndarray
    Array of NMR data which has been multiplied by a constant.

Notes

Parameter c is used even when r and i are defined. NMRPipe’s MULT function ignores c when r or i are defined.
\texttt{nmrglue.process.pipe\_proc.rev} \footnote{nmrglue Documentation, Release 0.7-dev}

\texttt{nmrglue.process.pipe\_proc.rev} \texttt{(dic, data, sw=True)}

Reverse data.

**Parameters**
\begin{itemize}
\item \texttt{dic} : dict
\hspace{1cm} Dictionary of NMRPipe parameters.
\item \texttt{data} : ndarray
\hspace{1cm} Array of NMR data.
\item \texttt{sw} : bool
\hspace{1cm} True to update carrier parameters in ndic, recommended.
\end{itemize}

**Returns**
\begin{itemize}
\item \texttt{ndic} : dict
\hspace{1cm} Dictionary of updated NMRPipe parameters.
\item \texttt{ndata} : ndarray
\hspace{1cm} Array of NMR data which has been reversed.
\end{itemize}

\texttt{nmrglue.process.pipe\_proc.set} \footnote{nmrglue Documentation, Release 0.7-dev}

\texttt{nmrglue.process.pipe\_proc.set} \texttt{(dic, data, r='a', i='a', c='a', x1=1.0, xn='default')} \footnote{nmrglue Documentation, Release 0.7-dev}

Set data to a constant.

**Parameters**
\begin{itemize}
\item \texttt{dic} : dict
\hspace{1cm} Dictionary of NMRPipe parameters.
\item \texttt{data} : ndarray
\hspace{1cm} Array of NMR data.
\item \texttt{r} : float, or ‘a’
\hspace{1cm} Constant to set real data to. “a” sets real data to 0 unless \texttt{c} is defined.
\item \texttt{i} : float or ‘a’
\hspace{1cm} Constant to set imaginary data to. “a” sets imaginary data to 0 unless \texttt{c} is defined.
\item \texttt{c} : float
\hspace{1cm} Constant to set both real and imaginary data by. ‘a’ sets both channels to 0 unless \texttt{r} or \texttt{i} in defined.
\item \texttt{x1} : int
\hspace{1cm} First point of region to set to the constant.
\item \texttt{xn} : int or ‘default’
\hspace{1cm} Last point of region to set to the constant. ‘default’ specifies the last point.
\end{itemize}

**Returns**
\begin{itemize}
\item \texttt{ndic} : dict
\hspace{1cm} Dictionary of updated NMRPipe parameters.
\item \texttt{ndata} : ndarray
\hspace{1cm} Array of NMR data which has been set to a constant.
**nmrglue.process.pipe_proc.shuf**

**nmrglue.process.pipe_proc.shuf**(dic, data, mode=None)

Shuffle Utilities

**Parameters**

- **dic**: dict
  - Dictionary of NMRPipe parameters.
- **data**: ndarray
  - Array of NMR data.
- **modes**: str
  - Shuffle mode. See table below for valid modes.

**Returns**

- **ndic**: dict
  - Dictionary of updated NMRPipe parameters.
- **ndata**: ndarray
  - Array of NMR data after shuffle.

**Notes**

'rr2ri’ mode ignores any imaginary vector refusing to create a mis-sized vector. 'bswap' mode may results in NaN in the data. 'r2i' and 'i2r' not implemented. All modes correctly update minimum and maximum values. This behavior may differ from NMRPipe’s SHUF function.

Valid modes are:

<table>
<thead>
<tr>
<th>string</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ri2c'</td>
<td>Interleave real and imaginary data.</td>
</tr>
<tr>
<td>'c2ri'</td>
<td>Separate real and imaginary data.</td>
</tr>
<tr>
<td>'ri2rr'</td>
<td>Append real and imaginary data.</td>
</tr>
<tr>
<td>'rr2ri'</td>
<td>Unappend real and imaginary data.</td>
</tr>
<tr>
<td>'exlr'</td>
<td>Exchange left and right halfs.</td>
</tr>
<tr>
<td>'rolr'</td>
<td>Rotate left and right halfs.</td>
</tr>
<tr>
<td>'swap'</td>
<td>Swap real and imaginary data.</td>
</tr>
<tr>
<td>'bswap'</td>
<td>Byte-swap data.</td>
</tr>
<tr>
<td>'inv'</td>
<td>Do nothing.</td>
</tr>
</tbody>
</table>

**nmrglue.process.pipe_proc.sign**

**nmrglue.process.pipe_proc.sign**(dic, data, ri=False, r=False, i=False, left=False, right=False, alt=False, abs=False, sign=False)

Sign manipulation utilities

**Parameters**

- **dic**: dict
  - Dictionary of NMRPipe parameters.
- **data**: ndarray
  - Array of NMR data.
- **ri**: bool
  - True to negate all data points.
**r** : bool
    True to negate real data points.

**i** : bool
    True to negate imaginary data points.

**left** : bool
    True to negate the left half of the data.

**right** : bool
    True to negate the right half of the data.

**alt** : bool
    True to negate alternating data points.

**abs** : bool
    True to replace both real and imaginary data with its absolute value.

**sign** : bool
    True to replace data with the sign (-1 or 1) of the data.

**Returns**

**ndic** : dict
    Dictionary of updated NMRPipe parameters.

**ndata** : ndarray
    Array of NMR data after sign manipulation.

**Notes**

All sign manipulation modes set True are applied in the order they appear in the function parameter list.

**Misc**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>coadd(dic, data[, cList, axis, time])</code></td>
<td>Co-addition of data</td>
</tr>
<tr>
<td><code>coac(dic, data[, cList, axis, time])</code></td>
<td>Co-addition of data</td>
</tr>
<tr>
<td><code>dev(dic, data)</code></td>
<td>Development function (does nothing)</td>
</tr>
<tr>
<td><code>img(dic, data, filter[, dx, dy, kern, conv, ...])</code></td>
<td>Image processing utilities</td>
</tr>
<tr>
<td><code>mac(dic, data[, macro, noRd, noWr, all])</code></td>
<td>Dispatcher similar to the MAC command.</td>
</tr>
<tr>
<td><code>null(dic, data)</code></td>
<td>Null function</td>
</tr>
<tr>
<td><code>qart(dic, data[, a, f, auto])</code></td>
<td>Scale Quad Artifacts</td>
</tr>
<tr>
<td><code>qmix(dic, data[, ic, oc, cList, time])</code></td>
<td>Complex mixing of input to outputs</td>
</tr>
<tr>
<td><code>save(dic, data, name[, overwrite])</code></td>
<td>Save the current vector.</td>
</tr>
<tr>
<td><code>smo(dic, data[, n, center])</code></td>
<td>Smooth data.</td>
</tr>
<tr>
<td><code>zd(dic, data[, wide, x0, slope, func, g])</code></td>
<td>Zero diagonal band.</td>
</tr>
</tbody>
</table>

```
import nmrglue

ndic = nmrglue.process.pipe_proc.coadd(dic, data[, cList=1, 1], axis='x', time=False)
```

Co-addition of data
Parameters  

- **dic** : dict  
  Dictionary of NMRPipe parameters.

- **data** : ndarray  
  Array of NMR data.

- **cList** : list  
  List of co-addition coefficients.

- **axis** : {‘x’, ‘y’}  
  Axis to co-add to and from.

- **time** : bool  
  True will adjust time-domain parameters in dic to account for the size reduction.

Returns  

- **ndic** : dict  
  Dictionary of updated NMRPipe parameters.

- **ndata** : ndarray  
  Array of NMR data after co-addition.

```
3.2. process modules
```

```python
nmrglue.process.pipe_proc.coad
```

```
nmrglue.process.pipe_proc.coad (dic, data, cList=[1, 1], axis='x', time=False)  
Co-addition of data
```

Parameters  

- **dic** : dict  
  Dictionary of NMRPipe parameters.

- **data** : ndarray  
  Array of NMR data.

- **cList** : list  
  List of co-addition coefficients.

- **axis** : {‘x’, ‘y’}  
  Axis to co-add to and from.

- **time** : bool  
  True will adjust time-domain parameters in dic to account for the size reduction.

Returns  

- **ndic** : dict  
  Dictionary of updated NMRPipe parameters.

- **ndata** : ndarray  
  Array of NMR data after co-addition.
**nmrglue Documentation, Release 0.7-dev**

**nmrglue.process.pipe_proc.dev**

`nmrglue.process.pipe_proc.dev(dic, data)`

Development function (does nothing)

**Parameters**

- **dic** : dict
  Dictionary of NMRPipe parameters.
- **data** : ndarray
  Array of NMR data.

**Returns**

- **ndic** : dict
  Original dictionary of NMRPipe parameters.
- **ndata** : ndarray
  Original array of NMR data.

**nmrglue.process.pipe_proc.img**

`nmrglue.process.pipe_proc.img(dic, data, filter, dx=1.0, dy=1.0, kern=[1], conv=False, thres=None)`

Image processing utilities

**Parameters**

- **dic** : dict
  Dictionary of NMRPipe parameters.
- **data** : ndarray
  Array of NMR data.
- **filter** : {'median', 'min', 'max', 'amin', 'amax', 'range', 'avg', 'dev'}
  Image processing filter to apply. See table below for descriptions.
- **dx** : float
  Filter width along X-axis in points.
- **dy** : float
  Filter width along Y-axis in points.
- **kern** : list
  List of convolution filter kernel values, only used when conv is True.
- **conv** : bool
  True to apply convolution filter with kernel of kern.
- **thres** : float or None or True
  Threshold value. Only points above this value will have the filter applied. None turns thresholding off and the filter will be applied to all points. True will set a threshold value of 0.

**Returns**

- **ndic** : dict
  Dictionary of updated NMRPipe parameters.
- **ndata** : ndarray
Imaged processed array of NMR data.

Notes

This function wraps when regions extend past the edges (NMRPipe doesn’t). The filter is applied to both the real and imaginary channels.

Supported filters are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>median</td>
<td>Median</td>
</tr>
<tr>
<td>min</td>
<td>Minimum</td>
</tr>
<tr>
<td>max</td>
<td>Maximum</td>
</tr>
<tr>
<td>amin</td>
<td>Absolute Minimum</td>
</tr>
<tr>
<td>amax</td>
<td>Absolute Maximum</td>
</tr>
<tr>
<td>range</td>
<td>Range</td>
</tr>
<tr>
<td>avg</td>
<td>Average</td>
</tr>
<tr>
<td>dev</td>
<td>Standard Deviation</td>
</tr>
</tbody>
</table>

nmrglue.process.pipe_proc.mac

nmrglue.process.pipe_proc.mac(dic, data, macro=None, noRd=False, noWr=False, all=False, **kwargs)

Dispatcher similar to the MAC command.

Parameters

dic : dict
   Dictionary of NMRPipe parameters

data : ndarray
   Array of NMR data.

macro : callable
   Python function to apply.

kwargs : keyword arguments
   Keyword arguments that get passed to the macro

Returns

dic : dict
   Dictionary of updated NMRPipe parameters.

ndata : ndarray
   Array of data with macro function applied.

Notes

This is not intended to be a true macro interpreter. Instead, the purpose is to act as a dispatch mechanism to other Python code so that the look and feel of nmrPipe is maintained. The -var and -str parameters are not used, as they can be passed directly to the macro as keyword arguments.

3.2. process modules
nmrglue Documentation, Release 0.7-dev

nmrglue.process.pipe_proc.null

nmrglue.process.pipe_proc.null(dic, data)

Null function

**Parameters**

- **dic** : dict
  - Dictionary of NMRPipe parameters.
- **data** : ndarray
  - Array of NMR data.

**Returns**

- **ndic** : dict
  - Dictionary of updated NMRPipe parameters.
- **ndata** : ndarray
  - Unmodified array of NMR data.

nmrglue.process.pipe_proc.qart

nmrglue.process.pipe_proc.qart(dic, data, a=0.0, f=0.0, auto=False)

Scale Quad Artifacts

**Parameters**

- **dic** : dict
  - Dictionary of NMRPipe parameters.
- **data** : ndarray
  - Array of NMR data.
- **a** : float
  - Amplitude adjustment value.
- **f** : float
  - Phase adjustment value.
- **auto** : bool
  - True will perform a Gram-Schmidt orthogonalization to fund a and f automatically. Provided a and f parameters are ignored.

**Returns**

- **ndic** : dict
  - Dictionary of updated NMRPipe parameters.
- **ndata** : ndarray
  - Array of NMR data with scaled quadrature artifacts.

**Notes**

Auto mode performs Gram-Schmidt orthogonalization, a different approach using a grid search is used in NM-RPipe’s QART function.
nmrglue.process.pipe_proc.qmix

Complex mixing of input to outputs

Parameters

- **dic**: dict
  - Dictionary of NMRPipe parameters.
- **data**: ndarray
  - Array of NMR data.
- **ic**: int
  - Number of input channels
- **oc**: int
  - Number of output channels
- **cList**: array_like
  - Array or mixing coefficients. This parameter must be able to be converted to an array and reshaped to (ic, oc).

Returns

- **ndic**: dict
  - Dictionary of updated NMRPipe parameters.
- **ndata**: ndarray
  - Array of NMR data after complex mixing.

Notes

`ic` and `oc` must evenly divide the data. This function refuses to make invalid length files, NMRPipe’s qmix function will create such files.

nmrglue.process.pipe_proc.save

Save the current vector.

Parameters

- **dic**: dict
  - Dictionary of NMRPipe parameters.
- **data**: ndarray
  - Array of NMR data.
- **name**: str
  - Filename to save vector to.
- **overwrite**: bool
  - True will overwrite existing files, False will raise a Warning if the file already exists.

Returns

- **ndic**: dict
  - Dictionary of updated NMRPipe parameters.
ndata : ndarray
Unmodified array of NMR data.

Notes

The resulting FDPIPECOUNT header parameter does not match the one created using NMRPipe’s SAVE function.

nmrglue.process.pipe_proc.smo

nmrglue.process.pipe_proc.smo(dic, data, n=1, center=False)
Smooth data.

Parameters dic : dict
Dictionary of NMRPipe parameters.

data : ndarray
Array of NMR data.

n : int
Size of smoothing window in points.

center : bool
True will perform a centering on the data (subtract the smoothed data). False returns the smoothed data.

Returns ndic : dict
Dictionary of updated NMRPipe parameters.

ndata : ndarray
Array of NMR data which has been smoothed or centered.

nmrglue.process.pipe_proc.zd

nmrglue.process.pipe_proc.zd(dic, data, wide=1.0, x0=1.0, slope=0, func=0, g=1)
Zero diagonal band.

Parameters dic : dict
Dictionary of NMRPipe parameters.

data : ndarray
Array of NMR data.

wide : int
Width of the diagonal band in points.

x0 : int
Starting location of the diagonal band in points.

slope : float
Slope of the diagonal band (X/Y ratio). A value of 0 will determine the slope automatically.

func : {0, 1, 2, 3}
Function to perform zero-ing with. 0 for a boxcar window, 1 for a triangle window, 2 for a sine bell, 3 for a Gaussian.

g : float
Width of Gaussian window in points. Only used if func is 3.

**Returns**

ddic : dict
Dictionary of updated NMRPipe parameters.

ndata : ndarray
Array of NMR data with a diagonal band zero-ed.

### Linear Prediction

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lp(dic, data[, pred, x1, xn, ord, mode, ...])</code></td>
<td>Linear Prediction</td>
</tr>
<tr>
<td><code>lpc(dic, data[, pred, x1, xn, ord, mode, ...])</code></td>
<td>Linear Prediction</td>
</tr>
<tr>
<td><code>lp2d(dic, data[, xOrd, yOrd, xSize, ySize, ...])</code></td>
<td>2D Linear Prediction using LP2D procedure</td>
</tr>
</tbody>
</table>

**nmrglue.process.pipe_proc.lp**

nmrglue.process.pipe_proc.\texttt{lp}(dic, data[, pred='default', x1='default', xn='default', ord=8, mode='f', append='after', bad_roots='auto', mirror=None, fix_mode='on', method='ils')

**Parameters**

dic : dict
Dictionary of NMRPipe parameters.

data : ndarray
Array of NMR data.

pred : int
Number of points to predict, “default” chooses the vector size for forward prediction, 1 for backward prediction

x1 : int or ‘default’
First point in 1D vector to use to extract LP filter. ‘default’ will use the first or last point depending on the mode.

xn : int or ‘default’
Last point in 1D vector to use to extract LP filter. ‘default’ will use the first or last point depending on the mode.

ord : int
Prediction order, number of LP coefficients used in prediction.

mode : {‘f’, ‘b’, ‘fb’}
Mode to generate LP filter, ‘f’ for forward,’b’ for backward, ‘fb’ for forward-backward.
append : {'before' or 'after'}

Location to append predicted data, ‘before’ or ‘after’ the existing data.

bad_roots {'incr', 'decr', None, ‘auto’} :

Type of roots which are will be marked as bad and stabilized. Choices are ‘incr’ for
increasing roots, ‘decr’ for decreasing roots, or None for not root stabilization. The
default ‘auto’ will set this parameter based upon the LP mode parameter: ‘f’ and ‘fb’
will results in an ‘incr’ parameter. ‘b’ in ‘decr’.

mirror : {'90-180', '0-0', None}

Mirror mode, option are ‘90-180’ for a one point shifted mirror image, ‘0-0’ for an exact
mirror image, and None for no mirror imaging of the data.

fix_mode : {'on', 'reflect'}

Method used to stabilize bad roots, ‘on’ moves bad roots onto the unit circle, ‘reflect’
reflect bad roots across the unit circle.

method : {'svd', 'qr', ‘choleskey’, ‘tls’}

Method to use to calculate the LP filter.

Returns ndic : dict

Dictionary of updated NMRPipe parameters.

ndata : ndarray

Array of NMR data with linear prediction applied.

Notes

The results from this function do not match NMRPipe’s LP function. Also some additional parameter and
different parameter in this function.

nmrglue.process.pipe_proc.lpc

nmrglue.process.pipe_proc.lpc(dic, data, pred='default', x1='default', xn='default', ord=8,
mode='f', append='after', bad_roots='auto', mirror=None,
fix_mode='on', method='tls')

Linear Prediction

Parameters dic : dict

Dictionary of NMRPipe parameters.

data : ndarray

Array of NMR data.

pred : int

Number of points to predict, “default” chooses the vector size for forward prediction, 1
for backward prediction

x1 : int or ‘default’

First point in 1D vector to use to extract LP filter. ‘default’ will use the first or last point
depending on the mode.
xn : int or ‘default’
    Last point in 1D vector to use to extract LP filter. ‘default’ will use the first or last point
    depending on the mode.

ord : int
    Prediction order, number of LP coefficients used in prediction.

mode : {'f', 'b', 'fb'}
    Mode to generate LP filter, ‘f’ for forward,’b’ for backward, ‘fb’ for forward-backward.

append : {'before' or ‘after’}
    Location to append predicted data, ‘before’ or ‘after’ the existing data.

bad_roots {'incr', 'decr', None, ‘auto’} :
    Type of roots which are will be marked as bad and stabilized. Choices are ‘incr’ for
    increasing roots, ‘decr’ for decreasing roots, or None for not root stabilization. The
    default ‘auto’ will set this parameter based upon the LP mode parameter: ‘f’ and ‘fb’
    will results in an ‘incr’ parameter. ‘b’ in ‘decr’.

mirror : {'90-180', '0-0', None}
    Mirror mode, option are ‘90-180’ for a one point shifted mirror image, ‘0-0’ for an exact
    mirror image, and None for no mirror imaging of the data.

fix_mode : {'on', ‘reflect’}
    Method used to stabilize bad roots, ‘on’ moves bad roots onto the unit circle, ‘reflect’
    reflect bad roots across the unit circle.

    Method to use to calculate the LP filter.

Returns ndic : dict
    Dictionary of updated NMRPipe parameters.

ndata : ndarray
    Array of NMR data with linear prediction applied.

Notes

The results from this function do not match NMRPipe’s LP function. Also some additional parameter and
different parameter in this function.

```
nmrglue.process.pipe_proc.lp2d
```

```
nmrglue.process.pipe_proc.lp2d(dic, data, xOrd=8, yOrd=8, xSize=’default’, ySize=’default’,
xMirror=’0’, yMirror=’0’, fix_pts=True, method=’svd’)
```

2D Linear Prediction using LP2D procedure

Parameters dic : dict
    Dictionary of NMRPipe parameters.

data : ndarray
    Array of NMR data.
xOrd : int
   X dimension linear prediction order.

yOrd : int
   Y dimension linear prediction order.

xSize : int
   New size of Y-axis, ‘default’ doubles the current size.

ySize : int
   New size of Y-axis, ‘default’ double the current size.

xMirror : {‘0’, ‘180’}
   ‘Mode in which the mirror image of the X-axis should be formed. ‘0’: indicated no delay, ‘180’ for a half-point delay.

yMirror : {‘0’, ‘180’}
   Mode in which the mirror image of the Y-axis should be formed.

fix_pts : bool
   True to reduce predicted points with magnitude larger than the largest data point. False left predicted points unaltered.

   Method used to calculate the LP prediction filter.

Returns ndic : dict
   Dictionary of updated NMRPipe parameters.

ndata : ndarray
   Array of NMR data with 2D linear prediction applied.

Notes

This function applies the LP2D procedure as described in: G. Zhu and A. Bax, Journal of Magnetic Resonance, 1992, 98, 192-199. to the data matrix. The parameters and algorith used in NMRPipe’s LP2D function are not well documented and are not replicated here.

Not Implemented

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ann(dic, data)</td>
<td>Fourier Analysis by Neural Net</td>
</tr>
<tr>
<td>ebs(dic, data)</td>
<td>EBS Reconstruction</td>
</tr>
<tr>
<td>mac(dic, data[, macro, noRd, noWr, all])</td>
<td>Dispatcher similar to the MAC command.</td>
</tr>
<tr>
<td>mem(dic, data)</td>
<td>Maximum Entropy Reconstruction</td>
</tr>
<tr>
<td>ml(dic, data)</td>
<td>Maximum Likelihood Frequency Map</td>
</tr>
<tr>
<td>poly(dic, data)</td>
<td>Polynomial Baseline Correction</td>
</tr>
<tr>
<td>xyz2zyx(dic, data)</td>
<td>3D Matrix transpose</td>
</tr>
<tr>
<td>ztp(dic, data)</td>
<td>3D Matrix Transpose</td>
</tr>
</tbody>
</table>
nmrglue.process.pipe_proc.ann

nmrglue.process.pipe_proc.ann\( (dic, data) \)
Fourier Analysis by Neural Net

nmrglue.process.pipe_proc.ebs

nmrglue.process.pipe_proc.ebs\( (dic, data) \)
EBS Reconstruction

nmrglue.process.pipe_proc.mem

nmrglue.process.pipe_proc.mem\( (dic, data) \)
Maximum Entropy Reconstruction

nmrglue.process.pipe_proc.ml

nmrglue.process.pipe_proc.ml\( (dic, data) \)
Maximum Likelihood Frequency Map

nmrglue.process.pipe_proc.poly

nmrglue.process.pipe_proc.poly\( (dic, data) \)
Polynomial Baseline Correction

nmrglue.process.pipe_proc.xyz2zyx

nmrglue.process.pipe_proc.xyz2zyx\( (dic, data) \)
3D Matrix transpose

nmrglue.process.pipe_proc.ztp

nmrglue.process.pipe_proc.ztp\( (dic, data) \)
3D Matrix Transpose

3.3 analysis modules

3.3.1 nmrglue.analysisbase

analysisbase provides general purpose analysis functions and classes used by several nmrglue.analysis modules
This module is imported as nmrglue.analysisbase and can be called as such.
Developer Functions

These functions are typically not used directly by users. They are called by high level functions.
neighbors\((pt, shape, structure)\)

Generate a list of all neighbors to a point.

valid\_pt\((pt, shape)\)

Determine if a point (indices) is valid for a given shaped.

find\_limits\((pts)\)

Find the limits which outline the provided list of points.

limits2slice\((limits)\)

Create a set of slice objects given an array of min, max limits.

slice2limits\((slices)\)

Create a tuple of minimum, maximum limits from a set of slices.

squish\((r, axis)\)

Squish array along an axis.

**nmrglue\_analysis\_analysisbase\_neighbors**

Generate a list of all neighbors to a point.

**Parameters**

- **pt**: tuple of ints
  - Index of the point to find neighbors of.
- **shape**: tuple of ints
  - Shape of the region.
- **structure**: ndarray of bools
  - Structure element that defines connections.

**Returns**

- **pts**: list of int tuples
  - List of tuples which represent indices for all points neighboring pt. Edges are treated as stopping points.

**nmrglue\_analysis\_analysisbase\_valid\_pt**

Determine if a point (indices) is valid for a given shaped.

**nmrglue\_analysis\_analysisbase\_find\_limits**

Find the limits which outline the provided list of points.

**Parameters**

- **pts**: list of int tuples
  - List of points \([z0, y0, x0], (z1, y1, x1), ...\)

**Returns**

- **min**: ndarray
  - Array of minimum indices: \([z\text{min}, y\text{min}, x\text{min}]\)
- **max**: ndarray
  - Array of maximum indices: \([z\text{max}, y\text{max}, x\text{max}]\)

**See also:**

- **limits2slice**: Create a list of slices from min, max limits

3.3. analysis modules
nmrglue.analysis.analysisbase.limits2slice

nmrglue.analysis.analysisbase.limits2slice(limits)
Create a set of slice objects given an array of min, max limits.

Parameters limits: tuple, (ndarray, ndarray):
Two tuple consisting of array of the minimum and maximum indices.

Returns slices: list
List of slice objects which return points between limits

See also:
find_limits Find the minimum and maximum limits from a list of points.
slice2limits Find a minimum and maximum limits for a list of slices.

nmrglue.analysis.analysisbase.slice2limits

nmrglue.analysis.analysisbase.slice2limits(slices)
Create a tuple of minimum, maximum limits from a set of slices.

Parameters slices: list
List of slice objects which return points between limits

Returns limits: tuple, (ndarray, ndarray):
Two tuple consisting of array of the minimum and maximum indices.

See also:
limits2slice Find a list of slices given minimum and maximum limits.

nmrglue.analysis.analysisbase.squish

nmrglue.analysis.analysisbase.squish(r, axis)
Squish array along an axis.
Determine the sum along all but one axis for an array.

Parameters r: ndarray
Array to squish.

axis: int
Axis of r to squish along.

Returns s: 1D ndarray
Array r squished into a single dimension.

Continued on next page
Table 3.62 – continued from previous page

Developer Classes

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ndwindow(shape, wsize)</code></td>
<td>An N-dimensional iterator to slice arrays into windows.</td>
</tr>
<tr>
<td><code>ndwindow_index(shape, wsize)</code></td>
<td>An N-dimensional iterator object which returns the index of the window center and a <code>ndwindow</code> slice array.</td>
</tr>
<tr>
<td><code>ndwindow_inside(shape, wsize)</code></td>
<td>An N-dimentional iterator to slice arrays into uniform size windows.</td>
</tr>
<tr>
<td><code>ndwindow_inside_index(shape, wsize)</code></td>
<td>An N-dimensional iterator object which returns the index of the window top-left and a <code>ndwindow_inside</code> slice array.</td>
</tr>
</tbody>
</table>

**nmrglue.analysis.analysisbase.ndwindow**

**class** `nmrglue.analysis.analysisbase.ndwindow(shape, wsize)`

An N-dimensional iterator to slice arrays into windows.

Given the shape of an array and a window size, an ‘ndwindow’ instance iterators over tuples of slices which slice an the array into wsize sub-arrays. At each iteration, the index of the center of the sub-array is incremented by one along the last dimension. Array borders are ignored so the resulting sub-array can be smaller than wsize. If wsize contains even values the window is off center containing an additional point with lower index.

**Parameters**

- `size` : tuple of ints
  Size of array to generate tuples of slices from.
- `wsize` : tuple of ints
  Window/sub-array size. Size of the area to select from array. This is the maximum size of the window.

**See also:**

- `ndwindow_index` Iterator of a ndwindow and index of the window center
- `ndwindow_inside` Iterator over equal sized windows in the array.

**Examples**

```python
>>> a = np.arange(12).reshape(3,4)
>>> for s in ndwindow(a.shape,(3,3)):
...     print(a[s])
[[[0 1]
  [4 5]]
[[0 1 2]
  [4 5 6]]
[[1 2 3]
  [5 6 7]]
[[2 3]
  [6 7]]
[[0 1]
  [4 5]
  [8 9]]
[[ 0 1 2]]
```

3.3. analysis modules
__init__(shape, wsize)
Set up the ndwindow object

Methods

__init__(shape, wsize)
next(() -> the next value, ...)

nmrglue.analysis.analysisbase.ndwindow_index

class nmrglue.analysis.analysisbase.ndwindow_index(shape, wsize)
An N-dimensional iterator object which returns the index of the window center and a ndwindow slice array.
See ndwindow for additional documentation.

This class is equivalent to:

for slices, index in zip(np.ndindex(shape), ndwindow(shape,wshape)): return (index, slice)

See also:

ndwindow  Iterator over only the window slices.
ndwindow_inside  Iterator over equal sized windows in the array.

__init__(shape, wsize)
Set up the object

Methods

__init__(shape, wsize)
next(() -> the next value, ...)

Chapter 3. Reference Guide
nmrglue.analysis.analysisbase.ndwindow_inside

class nmrglue.analysis.analysisbase.ndwindow_inside(shape, wsize)
    An N-dimensional iterator to slice arrays into uniform size windows.
    Given the shape of an array and a window size, an `ndwindow_inside` instance
    iterates over tuples of slices which slice an the array into uniform size wsize windows/sub-arrays. At each iteration, the index of the top left
    of the sub-array is incremented by one along the last dimension until the resulting windows would extend past
    the array border. All sub-arrays are equal sized (wsize).

    Parameters
    size : tuple of ints
        Size of array to generate tuples of slices from.
    wsize : tuple of ints
        Size of the area to select from array (window size).

    See also:
    ndwindow  Iterator over non-uniform windows.
    ndwindow_inside_index  Iterator of a ndwindow_inside and the index of the window’s top left point.

Examples

```python
>>> a = np.arange(9).reshape(3,3)
>>> for s in ndwindow_inside(a.shape,(2,2)):
...    print(a[s])
    [[0 1]
    [3 4]]
    [[1 2]
    [4 5]]
    [[3 4]
    [6 7]]
    [[4 5]
    [7 8]]
```

__init__(shape, wsize)
    Set up the object

Methods

__init__(shape, wsize)  Set up the object
next() -> the next value, ...

nmrglue.analysis.analysisbase.ndwindow_inside_index

class nmrglue.analysis.analysisbase.ndwindow_inside_index(shape, wsize)
    An N-dimensional iterator object which returns the index of the window top-left and a
    ndwindow_inside slice array.
    Similar to `ndwindow_index` but reports top left index of window.
    See `ndwindow_inside` and :py:class:`ndwindow_index` for addition documentation.
__init__(shape, wsize)
Set up the object

Methods

__init__(shape, wsize)
next() -> the next value, or raiseStopIteration

3.3.2 nmrglue.helpers

Helper functions for lineshape fitting.
This modules is imported as nmrglue.helpers and can be called as such.

User Functions

delta(v, d) Limits of v +/- d for array v and d.
limit(vmin, vmax, npeaks) Fixed limits, vmin to vmax for all peaks.
scale_params(rec, prefix, first, last) scale lineshape parameters in rec with name prefixX with X from from
no_limits(nparams, npeaks) No limits on nparameters for npeaks.
no_limits_amp(npeaks) No amplitude limits for npeaks
super_zip(l) Zip a list after zipping each element.
scale_columns(prefix, first, last) Create a list of scale parameter column names with name prefixX with X

nmrglue.analysis.helpers.delta

nmrglue.analysis.helpers.delta(v, d)
Limits of v +/- d for array v and d.

nmrglue.analysis.helpers.limit

nmrglue.analysis.helpers.limit(vmin, vmax, npeaks)
Fixed limits, vmin to vmax for all peaks.

nmrglue.analysis.helpers.scale_params

nmrglue.analysis.helpers.scale_params(rec, prefix, first, last)
scale lineshape parameters in rec with name prefixX with X from from to last (inclusive)

nmrglue.analysis.helpers.no_limits

nmrglue.analysis.helpers.no_limits(nparams, npeaks)
No limits on nparameters for npeaks.
nmrglue.analysis.helpers.no_limits_amp

nmrglue.analysis.helpers.no_limits_amp(npeaks)
No amplitude limits for npeaks

nmrglue.analysis.helpers.super_zip

nmrglue.analysis.helpers.super_zip(l)
Zip a list after zipping each element.
Useful for lineshape parameter and bounds.

nmrglue.analysis.helpers.scale_columns

nmrglue.analysis.helpers.scale_columns(prefix, first, last)
Create a list of scale parameter column names with name prefixX with X from first to last (inclusive)

3.3.3 nmrglue.integration

This modules is imported as nmrglue.integration and can be called as such.

User Functions

These are functions which are targeted for users of nmrglue.

\[
\begin{align*}
\text{integrate} & \quad \text{(data, unit_conv, limits[, unit, ...])} \quad \text{Integrate one 1D data array within limits given in units.} \\
\text{ndintegrate} & \quad \text{(data, unit_conv, limits[, unit, ...])} \quad \text{Integrate one nD data array within limits given in units.}
\end{align*}
\]

nmrglue.analysis.integration.integrate

nmrglue.analysis.integration.integrate(data, unit_conv, limits, unit='ppm', noise_limits=None, norm_to_range=None, calibrate=1.0)
Integrate one 1D data array within limits given in units. Data points must be equally spaced.

Functional form of integration is:

\[
\text{value} = \sum_{a}^{b} s(x_i) dx
\]

Where: s is the signal, a and b are the limits of integration and dx is the width of each bin.

The integration error due to baseline noise is calculated as:

\[
\text{error} = \sigma_{\text{vol}} = \sigma \sqrt{n}
\]

if the noise_limits are set.

Where:

\[
n = \frac{|b - a|}{dx} + 1
\]

sigma is the standard deviation of the baseline noise. n is the number of bins in the integration range.

3.3. analysis modules
Parameters data: array like:

1d array of intensity

unit_conv: ‘fileiobase.unit_conversion‘ object:

unit_conversion object associated with data

limits: array like:

With shape (2,) or (P, 2). Array with lower and upper integration limit. Or array with P rows of lower and upper integration limits.

noise_limits: Optional[array like]

With shape(2, ). Array with lower and upper limits to section of data with only noise. A larger range will likely yield a more accurate estimate. It is unwise to use the very end of the spectrum for the noise range.

norm_to_range: Optional[int]

If given, all values are normalized to the value determined within the range given by limits[norm_to_range, :]

calibrate: Optional[float]

If norm_to_range is given, the values are re-normalized so that the integral within limits[norm_to_range, :] equals some number.

Returns array:

[value, ...] integration values

if noise_limits is given:

array:

[[value, error], ...] where error a one sigma estimate of the error only from the spectrum noise

nmrglue.analysis.integration.ndintegrate

nmrglue.analysis.integration.ndintegrate(data, unit_conv, limits, unit='ppm', noise_limits=None)

Integrate one nD data array within limits given in units. Data points must be equally spaced. Can only integrate one region per function call.

The integration error due to baseline noise is calculated as:

\[ \text{error} = \sigma_{\text{vol}} = \sigma \sqrt{\prod_{i} n_i}, \]

if the noise_limits are set.

Where: sigma is the standard deviation of the baseline noise. n is the number of bins in the integration range for each d dimensions.

See integrate for more information.

Parameters data: array like:

1d array of intensity

unit_convs: ['fileiobase.unit_conversion', ] list:
list of unit_conversion object associated with each dim of data.

**limits**: array like :

With shape (2,) or (d, 2). 1D Array with lower and upper integration limits for 1D. Or array with d rows of lower and upper integration limits for each dimension.

**noise_limits**: Optional[array like] :

With shape(2, ). Array with lower and upper limits to section of data with only noise. A larger range will likely yield a more accurate estimate. It is unwise to use the very end of the spectrum for the noise range.

Returns **array** :

[value, ...] integration values

if **noise_limits** is given: :

array :

[[value, error], ...] where error a one sigma estimate of the error only from the spectrum noise

### 3.3.4 nmrglue.leastsqbound

Constrained multivariate least-squares optimization

This module is imported as nmrglue.leastsqbound and can be called as such.

#### Developer Functions

nmrglue.analysis.leastsqbound.leastsqbound(func, x0, args=(), bounds=None, Dfun=None, full_output=0, col_deriv=0, ftol=1.49012e-08, xtol=1.49012e-08, gtol=0.0, maxfev=0, epsfcn=0.0, factor=100, diag=None)

Bounded minimization of the sum of squares of a set of equations.

\[
x = \text{arg min} \left( \text{sum} (\text{func}(y) \times 2, \text{axis}=0) \right)
\]

**Parameters** func : callable

should take at least one (possibly length N vector) argument and returns M floating point numbers.

x0 : ndarray

The starting estimate for the minimization.

args : tuple

Any extra arguments to func are placed in this tuple.

bounds : list

(min, max) pairs for each element in x, defining the bounds on that parameter. Use None for one of min or max when there is no bound in that direction.

Dfun : callable
A function or method to compute the Jacobian of func with derivatives across the rows. If this is None, the Jacobian will be estimated.

**full_output**: bool

non-zero to return all optional outputs.

**col_deriv**: bool

non-zero to specify that the Jacobian function computes derivatives down the columns (faster, because there is no transpose operation).

**ftol**: float

Relative error desired in the sum of squares.

**xtol**: float

Relative error desired in the approximate solution.

**gtol**: float

Orthogonality desired between the function vector and the columns of the Jacobian.

**maxfev**: int

The maximum number of calls to the function. If zero, then 100*(N+1) is the maximum where N is the number of elements in x0.

**epsfcn**: float

A suitable step length for the forward-difference approximation of the Jacobian (for Dfun=None). If epsfcn is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.

**factor**: float

A parameter determining the initial step bound (factor * || diag * x||). Should be in interval (0.1, 100).

**diag**: sequence

N positive entries that serve as a scale factors for the variables.

**Returns**

- **x**: ndarray

  The solution (or the result of the last iteration for an unsuccessful call).

- **cov_x**: ndarray

  Uses the fjac and ipvt optional outputs to construct an estimate of the jacobian around the solution. None if a singular matrix encountered (indicates very flat curvature in some direction). This matrix must be multiplied by the residual standard deviation to get the covariance of the parameter estimates – see curve_fit.

- **infodict**: dict

  a dictionary of optional outputs with the key s:

  - 'nfev' : the number of function calls
  - 'fvec' : the function evaluated at the output
  - 'fjac' : A permutation of the R matrix of a QR factorization of the final approximate Jacobian matrix, stored column wise. Together with ipvt, the covariance of the estimate can be approximated.
  - 'ipvt' : an integer array of length N which defines
a permutation matrix, p, such that
\[ \text{fjac} \cdot p = q \cdot r, \]
where \( r \) is upper triangular
with diagonal elements of nonincreasing magnitude. Column \( j \) of \( p \) is column \( \text{ipvt}(j) \)
of the identity matrix.

\[- 'qtf' : \text{the vector } (\text{transpose}(q) \cdot \text{fvec}).\]

**mesg** : str
A string message giving information about the cause of failure.

**ier** : int
An integer flag. If it is equal to 1, 2, 3 or 4, the solution was found. Otherwise, the solution was not found. In either case, the optional output variable ‘mesg’ gives more information.

**Notes**

“leastsq” is a wrapper around MINPACK’s lmdif and lmder algorithms.

cov_x is a Jacobian approximation to the Hessian of the least squares objective function. This approximation assumes that the objective function is based on the difference between some observed target data (ydata) and a (non-linear) function of the parameters \( f(xdata, params) \)

\[
\text{func}(\text{params}) = ydata - f(xdata, \text{params})
\]

so that the objective function is

\[
\text{min} \quad \sum_{\text{params}} ((ydata - f(xdata, \text{params}))^2, \text{axis}=0)
\]

Constraints on the parameters are enforced using an internal parameter list with appropriate transformations such that these internal parameters can be optimized without constraints. The transformation between a given internal parameter, \( p_i \), and a external parameter, \( p_e \), are as follows:

**With min and max bounds defined**

\[
p_i = \arcsin((2 \cdot (p_e - \text{min}) / (\text{max} - \text{min})) - 1.) \\
p_e = \text{min} + ((\text{max} - \text{min}) / 2.) \cdot (\sin(p_i) + 1.)
\]

**With only max defined**

\[
p_i = \sqrt{((\text{max} - p_e + 1.)^2 - 1.)} \\
p_e = \text{max} + 1. - \sqrt{p_i^2 + 1.}
\]

**With only min defined**

\[
p_i = \sqrt{((p_e - \text{min} + 1.)^2 - 1.)} \\
p_e = \text{min} - 1. + \sqrt{p_i^2 + 1.}
\]

These transformations are used in the MINUIT package, and described in detail in the section 1.3.1 of the MINUIT User’s Guide.

**References**

3.3.5 nmrglue.lineshapes1d

One-dimensional lineshape functions and classes
This modules is imported as nmrglue.lineshapes1d and can be called as such.

Developer Functions

These functions are typically not used directly by users. Developers who want to segment spectra will be interested in these functions.

- `sim_gauss_fwhm(x, x0, fwhm)` Simulate a Gaussian (normal) lineshape with unit height at the center.
- `sim_lorentz_fwhm(x, x0, fwhm)` Simulate a Lorentzian lineshape with unit height at the center.
- `sim_voigt_fwhm(x, x0, fwhm_g, fwhm_l)` Simulate a Voigt lineshape with unit height at the center.
- `sim_pvoigt_fwhm(x, x0, fwhm, eta)` Simulate a Pseudo Voigt lineshape with unit height at the center.
- `sim_gauss_sigma(x, x0, sigma)` Simulate a Gaussian (normal) lineshape with unit height at the center.
- `sim_lorentz_gamma(x, x0, gamma)` Simulate a Lorentzian lineshape with unit height at the center.
- `sim_voigt_sigmagamma(x, x0, sigma, gamma)` Simulate a Voigt lineshape with unit height at the center.
- `ls_str2class(l)`
- `center_fwhm(signal)` Estimate the center and full-width half max of a signal.
- `center_fwhm_bymoments(signal)` Estimate the center and full-width half max of a signal using moments.

**nmrglue.analysis.lineshapes1d.sim_gauss_fwhm**

`nmrglue.analysis.lineshapes1d.sim_gauss_fwhm(x, x0, fwhm)` Simulate a Gaussian (normal) lineshape with unit height at the center.

Simulate discrete points of a continuous Gaussian (normal) distribution with unit height at the center. FWHM (full-width at half-maximum) is used as the distribution scale parameter.

Functional form:

\[ f(x; x_0, \text{fwhm}) = \exp\left(-\frac{(x - x_0)^2 * 4 * \ln(2)}{\text{fwhm}^2}\right) \]

**Parameters**

- `x` : ndarray
  
  Array of values at which to evaluate distribution.

- `x0` : float
  
  Center (mean) of Gaussian distribution.

- `fwhm` : float
  
  Full-width at half-maximum of distribution.

**Returns**

- `f` : ndarray
  
  Distribution evaluated at points in `x`. 

Chapter 3. Reference Guide
nmrglue.analysis.lineshapes1d.sim_lorentz_fwhm

nmrglue.analysis.lineshapes1d.sim_lorentz_fwhm(x, x0, fwhm)

Simulate a Lorentzian lineshape with unit height at the center.
Simulates discrete points of the continuous Cauchy-Lorentz (Breit-Wigner) distribution with unit height at the center. FWHM (full-width at half-maximum) is used as the scale parameter.

Functional form:
\[ f(x; x_0, fwhm) = \frac{(0.5 \times fwhm)^2}{(x-x_0)^2 + (0.5 \times fwhm)^2} \]

Parameters
\( x \): ndarray
Array of values at which to evaluate distribution.
\( x_0 \): float
Center of the distribution.
\( fwhm \): float
Full-width at half-maximum of distribution.

Returns
\( f \): ndarray
Distribution evaluated at points in \( x \).

nmrglue.analysis.lineshapes1d.sim_voigt_fwhm

nmrglue.analysis.lineshapes1d.sim_voigt_fwhm(x, x0, fwhm_g, fwhm_l)

Simulate a Voigt lineshape with unit height at the center.
Simulates discrete points of the continuous Voigt profile with unit height at the center. Full-width at half-maximum (FWHM) of each component are used as the scale parameters for the Gaussian and Lorentzian distribution.

Functional Form:
\[ f(x; x_0, fwhm_g, fwhm_l) = \frac{\text{Re}[w(z)]}{\text{Re}[w(z_0)]} \]

Where:
\[ z = \sqrt{\ln(2)} \times (2 \times (x - x_0) + 1j \times fwhm_l) / fwhm_g \]
\[ z_0 = \sqrt{\ln(2)} \times 1j \times fwhm_l / fwhm_g \]
\[ w(z) \] is the complex error function of \( z \)

Parameters
\( x \): ndarray
Array of values at which to evaluate distribution.
\( x_0 \): float
Center of the distribution.
\( fwhm_g \): float
Full-width at half-maximum of the Gaussian component.
\( fwhm_l \): float
Full-width at half-maximum of the Lorentzian component.

Returns
\( f \): ndarray
Distribution evaluated at points in \( x \).
nmrglue.analysis.lineshapes1d.sim_pvoigt_fwhm

```python
nmrglue.analysis.lineshapes1d.sim_pvoigt_fwhm(x, x0, fwhm, eta)
```

Simulate a Pseudo Voigt lineshape with unit height at the center.
Simulates discrete points of the continuous Pseudo Voigt profile with unit height at the center. Full-width at half-maximum (FWHM) of the Gaussian and Lorentzian distribution are used as the scale parameter as well as eta, the mixing factor.

Functional Form:

\[ f(x; x0, fwhm, \eta) = (1-\eta) \cdot G(x; x0, fwhm) + \eta \cdot L(x; x0, fwhm) \]

Where:

\[ G(x; x0, fwhm) = \exp\left(-\frac{(x-x0)^2 \cdot 4 \cdot \ln(2)}{fwhm^2}\right) \]
\[ L(x; x0, fwhm) = \frac{(0.5 \cdot fwhm)^2}{(x-x0)^2 + (0.5 \cdot fwhm)^2} \]

**Parameters**

- `x`: ndarray
  - Array of values at which to evaluate the distribution.
- `x0`: float
  - Center of the distribution.
- `fwhm`: float
  - Full-width at half-maximum of the Pseudo Voigt profile.
- `eta`: float
  - Lorentzian/Gaussian mixing parameter.

**Returns**

- `f`: ndarray
  - Distribution evaluated at points in `x`.

---

nmrglue.analysis.lineshapes1d.sim_gauss_sigma

```python
nmrglue.analysis.lineshapes1d.sim_gauss_sigma(x, x0, sigma)
```

Simulate a Gaussian (normal) lineshape with unit height at the center.
Simulate discrete points of a continuous Gaussian (normal) distribution with unit height at the center. Sigma (the standard deviation of the distribution) is used as the distribution scale parameter.

Functional form:

\[ f(x; x0, \sigma) = \exp\left(-\frac{(x - x0)^2}{2 \cdot \sigma^2}\right) \]

**Parameters**

- `x`: ndarray
  - Array of values at which to evaluate the distribution.
- `x0`: float
  - Center (mean) of Gaussian distribution.
- `sigma`: float
  - Scale (variance) of the Gaussian distribution.

**Returns**

- `f`: ndarray
  - Distribution evaluated at points in `x`.
**nmrglue.analysis.lineshapes1d.sim_lorentz_gamma**

Simulate a Lorentzian lineshape with unit height at the center.

Simulates discrete points of the continuous Cauchy-Lorentz (Breit-Wigner) distribution with unit height at the center. Gamma (the half-width at half-maximum, HWHM) is used as the scale parameter.

Functional form:

\[ f(x; x_0, \gamma) = \frac{g^2}{(x-x_0)^2 + g^2} \]

**Parameters**

- `x` : ndarray
  Array of values at which to evaluate distribution.
- `x0` : float
  Center of the distribution.
- `gamma` : float
  Scale parameter, half-width at half-maximum, of distribution.

**Returns**

- `f` : ndarray
  Distribution evaluated at points in x.

**nmrglue.analysis.lineshapes1d.sim_voigt_sigmagamma**

Simulate a Voigt lineshape with unit height at the center.

Simulates discrete points of the continuous Voigt profile with unit height at the center. Sigma and gamma are used as the Gaussian and Lorentzian scaler parameters.

Functional Form:

\[ f(x; x_0, \sigma, \gamma) = \frac{\text{Re}[w(z)]}{\text{Re}[w(z_0)]} \]

Where:

\[ z = \frac{(x - x_0) + 1j \cdot \gamma}{\sigma \sqrt{2}} \]
\[ z_0 = \frac{1j \cdot \gamma}{\sigma \sqrt{2}} \]
\( w(z) \) is the complex error function of \( z \)

**Parameters**

- `x` : ndarray
  Array of values at which to evaluate distribution.
- `x0` : float
  Center of the distribution.
- `sigma` : float
  Gaussian scale component of Voigt profile. Variance of the Gaussian distribution.
- `gamma` : float
  Lorentzian scale component of Voigt profile. Half-width at half-maximum of the Lorentzian component.

**Returns**

- `f` : ndarray
  Distribution evaluated at points in x.
nmrglue.Documentation, Release 0.7-dev

nmrglue.analysis.lineshapes1d.ls_str2class

nmrglue.analysis.lineshapes1d.ls_str2class(l)

nmrglue.analysis.lineshapes1d.center_fwhm

nmrglue.analysis.lineshapes1d.center_fwhm(signal)
   Estimate the center and full-width half max of a signal.

nmrglue.analysis.lineshapes1d.center_fwhm_bymoments

nmrglue.analysis.lineshapes1d.center_fwhm_bymoments(signal)
   Estimate the center and full-width half max of a signal using moments

Developer Classes

<table>
<thead>
<tr>
<th>location_scale()</th>
<th>Base class for building a 2 parameter location scale lineshape class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>gauss_sigma()</td>
<td>Gaussian (normal) lineshape class with unit height at the mean and sigma scale parameter.</td>
</tr>
<tr>
<td>gauss_fwhm()</td>
<td>Gaussian (normal) lineshape class with unit height at the mean and fwhm scale parameter.</td>
</tr>
<tr>
<td>lorentz_gamma()</td>
<td>Lorentzian lineshape class with unit height at the center and gamma scale parameter.</td>
</tr>
<tr>
<td>lorentz_fwhm()</td>
<td>Lorentzian lineshape class with unit height at the center and gamma scale parameter.</td>
</tr>
<tr>
<td>location_2params()</td>
<td>Base Class for building a 3 parameter location, scale, other lineshape classes.</td>
</tr>
<tr>
<td>voigt_fwhm()</td>
<td>Voigt lineshape class with unit height at the center and full-width half-maximum scale parameters.</td>
</tr>
<tr>
<td>voigt_sigmagamma()</td>
<td>Voigt lineshape class with unit height at the center and sigma, gamma scale parameters.</td>
</tr>
<tr>
<td>pvoigt_fwhm()</td>
<td>Pseudo-Voigt lineshape class with unit height at the center and full-width half-maximum scale parameter.</td>
</tr>
<tr>
<td>scale()</td>
<td>Scale lineshape class</td>
</tr>
</tbody>
</table>

nmrglue.analysis.lineshapes1d.location_scale

class nmrglue.analysis.lineshapes1d.location_scale
   Base class for building a 2 parameter location scale lineshape class.
   __init__()
nmrglue.analysis.lineshapes1d.gauss_sigma

class nmrglue.analysis.lineshapes1d.gauss_sigma
Gaussian (normal) lineshape class with unit height at the mean and sigma scale parameter. See `sim_gauss_sigma()` for functional form and parameters.

    __init__()

Methods

    __init__()
    add_edge(p, limits)
    guessp(sig)
    nparam(M)
    pnames(M)
    remove_edge(p, limits)
    sim(M, p)

Attributes

    name

nmrglue.analysis.lineshapes1d.gauss_fwhm

class nmrglue.analysis.lineshapes1d.gauss_fwhm
Gaussian (normal) lineshape class with unit height at the mean and fwhm scale parameter. See `sim_gauss_fwhm()` for functional form and parameters.

    __init__()

Methods

    __init__()
    add_edge(p, limits)
    guessp(sig)
    nparam(M)
    pnames(M)
    remove_edge(p, limits)
    sim(M, p)

Attributes

3.3. analysis modules
nmrglue Documentation, Release 0.7-dev

nmrglue.analysis.lineshapes1d.lorentz_gamma

class nmrglue.analysis.lineshapes1d.lorentz_gamma
    Lorentzian lineshape class with unit height at the center and gamma scale parameter. See sim_lorentz_gamma() for functional form and parameters.
    __init__(self)

Methods

__init__()
add_edge(p, limits)
guessp(sig)
nparam(M)
pnames(M)
remove_edge(p, limits)
sim(M, p)

Attributes

name

nmrglue.analysis.lineshapes1d.lorentz_fwhm

class nmrglue.analysis.lineshapes1d.lorentz_fwhm
    Lorentzian lineshape class with unit height at the center and gamma scale parameter. See sim_lorentz_fwhm() for functional form and parameters.
    __init__(self)

Methods

__init__()
add_edge(p, limits)
guessp(sig)
nparam(M)
pnames(M)
remove_edge(p, limits)
sim(M, p)

Attributes

name
nmrglue.analysis.lineshapes1d.location_2params

class nmrglue.analysis.lineshapes1d.location_2params
    Base Class for building a 3 parameter location, scale, other lineshape classes.
    __init__()

    Methods

    __init__()
    add_edge(p, limits)
    npnum(M)
    remove_edge(p, limits)

nmrglue.analysis.lineshapes1d.voigt_fwhm

class nmrglue.analysis.lineshapes1d.voigt_fwhm
    Voigt lineshape class with unit height at the center and full-width half-maximum scale parameters. See
    sim_voigt_fwhm() for functional form and parameters.
    __init__()

    Methods

    __init__()
    add_edge(p, limits)
    guessp(sig)
    npnum(M)
    pnames(M)
    remove_edge(p, limits)
    sim(M, p)

    Attributes

    name

nmrglue.analysis.lineshapes1d.voigt_sigmagamma

class nmrglue.analysis.lineshapes1d.voigt_sigmagamma
    Voigt lineshape class with unit height at the center and sigma, gamma scale parameters. See
    sim_voigt_sigmagamma() for functional form and parameters.
    __init__()
Methods

```
__init__()
add_edge(p, limits)
guessp(sig)
nparam(M)
pnames(M)
remove_edge(p, limits)
sim(M, p)
```

Attributes

```
name
```

**nmrglue.analysis.lineshapes1d.pvoigt_fwhm**

class nmrglue.analysis.lineshapes1d.pvoigt_fwhm

    Pseudo-Voigt lineshape class with unit height at the center and full-width half-maximum scale parameter. See sim_pvoigt_fwhm() for functional form and parameters.

    __init__()

Methods

```
__init__()
add_edge(p, limits)
guessp(sig)
nparam(M)
pnames(M)
remove_edge(p, limits)
sim(M, p)
```

Attributes

```
name
```

**nmrglue.analysis.lineshapes1d.scale**

class nmrglue.analysis.lineshapes1d.scale

    Scale lineshape class

    Simulates a lineshape with functional form: 1.0, a0, a1, a2, ... Where a0, a1, ... are the parameters provided.

    __init__()
Methods

__init__()
add_edge(p, limits)
guessp(sig)
nparam(M)
pnames(M)
remove_edge(p, limits)
sim(M, p)

Attributes

name

3.3.6 nmrglue.linesh

Functions for fitting and simulating arbitrary dimensional lineshapes commonly found in NMR experiments
This module is imported as nmrglue.linesh and can be called as such

User Functions

These are the functions the majority of users will use from the linesh module.

fit_spectrum(spectrum, lineshapes, params, ...) Fit a NMR spectrum by regions which contain one or more peaks.

fit_NDregion(region, lineshapes, params, amps) Fit a N-dimensional region.

sim_NDregion(shape, lineshapes, params, amps) Simulate an N-dimensional region with one or more peaks.

add_to_table(rec, columns, column_names) Add (append) multiple columns to a records array.

pack_table(pbest, abest, iers, rec, ...[, ...]) Pack fitting parameters into table

unpack_table(rec, param_columns, amp_column) Unpack initial fitting parameters from a table.

estimate_scales(spectrum, centers, box_width) Estimate scale parameter for peaks in a spectrum.

nmrglue.analysis.linesh.fit_spectrum

nmrglue.analysis.linesh.fit_spectrum(spectrum, lineshapes, params, amps, bounds, ampbounds, centers, rIDs, box_width, error_flag, verb=True, **kw)

Fit a NMR spectrum by regions which contain one or more peaks.

Parameters

spectrum : array_like
NMR data. ndarray or emulated type, must be slicable.

lineshape : list
List of lineshapes by label (str) or a lineshape class. See fit_NDregion() for details.

params : list
P-length list (P is the number of peaks in region) of N-length lists of tuples where each each tuple is the optimization starting parameters for a given peak and dimension lineshape.

**amps** : list

P-length list of amplitudes.

**bounds** : list

List of bounds for parameter of same shape as params. If none of the parameters in a given dimension have limits None can be used, otherwise each dimension should have a list or tuple of (min,max) or None for each parameter. min or max may be None when there is no bounds in a given direction.

**ampbounds** : list

P-length list of bounds for the amplitude with format similar to bounds.

**centers** : list

List of N-tuples indicating peak centers.

**rIDs** : list

P-length list of region numbers. Peak with the same region number are fit together.

**box_width** : tuple

Tuple of length N indicating box width to add and subtract from peak centers to form regions around peak to fit.

**error_flag** : bool

True to estimate errors for each lineshape parameter and amplitude.

**verb** : bool, optional

True to print a summary of each region fit, False (the default) supresses all printing.

**kw** : optional

Additional keywords passed to the scipy.optimize.leastsq function.

**Returns**

**params_best** : list

Optimal values for lineshape parameters with same format as params input parameter.

**amp_best** : list

List of optimal peak amplitudes.

**param_err** : list, only returned when error_flag is True

Estimated lineshape parameter errors with same format as params.

**amp_err** : list, only returned when error_flag is True

Estimated peak amplitude errors.

**iers** : list

List of integer flag from scipy.optimize.leastsq indicating if the solution was found for a given peak. 1,2,3,4 indicates that a solution was found. Other indicate an error.
nmrglue.analyis.linesh.fit_NDregion

nmrglue.analyis.linesh.fit_NDregion(region, lineshapes, params, amps, bounds=None, ampbounds=None, wmask=None, error_flag=False, **kw)

Fit a N-dimensional region.

Parameters

region : ndarray
Region of a NMR data to fit.

lineshape : list:
List of lineshapes by label (str) or a lineshape class. See Notes for details.

params : list
P-length list (P is the number of peaks in region) of N-length lists of tuples where each each tuple is the optimization starting parameters for a given peak and dimension lineshape.

amps : list
P-length list of amplitudes.

bounds : list
List of bounds for parameter of same shape as params. If none of the parameters in a given dimension have limits None can be used, otherwise each dimension should have a list or tuple of (min,max) or None for each parameter. min or max may be None when there is no bounds in a given direction.

ampbounds : list
P-length list of bounds for the amplitude with format similar to bounds.

wmask : ndarray, optional
Array with same shape as region which is used to weight points in the error calculation, typically a boolean array is used to exclude certain points in the region. Default of None will include all points in the region equally in the error calculation.

centers : list
List of N-tuples indicating peak centers.

error_flag : bool
True to estimate errors for each lineshape parameter and amplitude.

**kw : optional
Additional keywords passed to the scipy.optimize.leastsq function.

Returns

params_best : list
Optimal values for lineshape parameters with same format as params input parameter.

amp_best : list
List of optimal peak amplitudes.

param_err : list, only returned when error_flag is True
Estimated lineshape parameter errors with same format as params.

amp_err : list, only returned when error_flag is True
Estimated peak amplitude errors.
iers : list

List of integer flag from scipy.optimize.leastsq indicating if the solution was found for
a given peak. 1,2,3,4 indicates that a solution was found. Other indicate an error.

Notes

The lineshape parameter:

Elements of the lineshape parameter list can be string indicating the lineshape of given dimension or an instance
of a lineshape class which provide a sim method which takes two arguments, the first being the length of the
lineshape the second being a list of lineshape parameters, and returns a simulated lineshape as well as an nparam
method which when given the length of lineshape returns the number of parameters needed to describe the
lineshape. Currently the following strings are allowed:

• ‘g’ or ‘gauss’ Gaussian (normal) lineshape.
• ‘l’ or ‘lorentz’ Lorentzian lineshape.
• ‘v’ or ‘voigt’ Voigt lineshape.
• ‘pv’ or ‘pvoigt’ Pseudo Voigt lineshape
• ‘s’ or ‘scale’ Scaled lineshape.

The first four lineshapes (Gaussian, Lorentzian, Voigt and Pseudo Voigt) all take a FWHM scale parameter.
The following are all valid lineshapes parameters for a 2D Gaussian peak:

• ['g','g']
• ['gauss','gauss']
• [ng.lineshapes1d.gauss(),ng.lineshapes1d.gauss()]

nmrglue.analysis.linesh.sim_NDregion

nmrglue.analysis.linesh.sim_NDregion(shape, lineshapes, params, amps)
Simulate an N-dimensional region with one or more peaks.

Parameters

shape : tuple of ints
Shape of region.

lineshapes : list
List of lineshapes by label (str) or a lineshape class. See fit_NDregion() for additional documentation.

params : list
P-length list (P is the number of peaks in region) of N-length lists of tuples where each
each tuple is lineshape parameters for a given peak and dimension.

amps : list
P-length of peak amplitudes.

Returns

sim : ndarray with shape, shape.
Simulated region.
nmrglue.analysis.linesh.add_to_table

nmrglue.analysis.linesh.add_to_table(rec, columns, column_names)
Add (append) multiple columns to a records array.

**Parameters**
- **rec**: recarray
  Records array (table).
- **columns**: list of ndarrays
  List of columns data to append to table.
- **column_names**: list of str
  List of names of columns.

**Returns**
- **nrec**: recarray
  Records array with columns added

nmrglue.analysis.linesh.pack_table

nmrglue.analysis.linesh.pack_table(pbest, abest, iers, rec, param_columns, amp_column, ier_column=None)
Pack fitting parameters into table

**Parameters**
- **pbest**: list
  List of best-fit parameters. See `fit_NDregion()` for format.
- **abest**: list
  List of best-fit amplitudes.
- **iers**: list
  List of fitting error return values.
- **rec**: recarray
  Records array (table) to save fitting parameters into. Updated with fitting parameter in place.
- **param_columns**: list
  List of parameter columns in rec. Format is the same as pbest.
- **amp_columns**: str
  Name of amplitude column in rec.
- **ier_column**: str or None, optional
  Name of column in rec to save iers to. None will not record this in the table.

nmrglue.analysis.linesh.unpack_table

nmrglue.analysis.linesh.unpack_table(rec, param_columns, amp_column)
Unpack initial fitting parameters from a table.

**Parameters**
- **rec**: recarray
  Records array (table) holding parameters.
**param_columns**: list

List of column names which hold lineshape parameters. See `fit_NDregion()` for format.

**amp_column**: str

Name of columns in rec holding initial amplitudes.

**Returns**

**params**: list

List of initial parameter in the format required for `fit_NDregion()`.

**amps**: list

List of initial peak amplitudes.

**nmrglue.analysis.linesh.estimate_scales**

```
```

Estimate scale parameter for peaks in a spectrum.

**Parameters**

**spectrum**: array_like

NMR spectral data. ndarray or emulated type which can be sliced.

**centers**: list

List of N-tuples indicating peak centers.

**box_width**: tuple

N-tuple indicating box width to add and subtract from peak centers to form region around peak to fit.

**scale_axis**: int

Axis number to estimate scale parameters for.

**Returns**

**scales**: list

List of estimated scale parameters.

**Developer Functions**

These functions are typically not used directly by users. Developers who want fine control over lineshape fitting may be interested in these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>make_slist(l, t_sizes)</code></td>
<td>Create a list of tuples of given sizes from a list</td>
</tr>
<tr>
<td><code>split_list(l, N)</code></td>
<td>Split list l into N sublists of equal size</td>
</tr>
<tr>
<td><code>calc_errors(region, ls_classes, p, cov, ...)</code></td>
<td>Calculate the parameter errors from the standard errors of the estimate.</td>
</tr>
<tr>
<td><code>s_NDregion(p, shape, ls_classes, n_peaks)</code></td>
<td>Simulate an N-dimensional region with one or more peaks.</td>
</tr>
<tr>
<td><code>s_single_NDregion(p, shape, ls_classes)</code></td>
<td>Simulate an N-dimensional region with a single peak.</td>
</tr>
<tr>
<td><code>err_NDregion(p, region, shape, ls_classes, ...)</code></td>
<td>Error function for an N-dimensional region, called by <code>f_NDregion()</code>.</td>
</tr>
<tr>
<td><code>f_NDregion(region, ls_classes, p0, p_bounds, ...)</code></td>
<td>Fit an N-dimensional regions containing one or more peaks.</td>
</tr>
</tbody>
</table>
nmrglue.analysis.linesh.make_slist

nmrglue.analysis.linesh.make_slist(l, t_sizes)
Create a list of tuples of given sizes from a list

Parameters
- l : list or ndarray
  List or array to pack into shaped list.
- t_sizes : list of ints
  List of tuple sizes.
Returns
- slist : list of tuples
  List of tuples of lengths given by t_sizes.

nmrglue.analysis.linesh.split_list

nmrglue.analysis.linesh.split_list(l, N)
Split list l into N sublists of equal size

nmrglue.analysis.linesh.calc_errors

nmrglue.analysis.linesh.calc_errors(region, ls_classes, p, cov, n_peaks, wmask)
Calculate the parameter errors from the standard errors of the estimate.

Parameters
- region : ndarray
  Region which was fit.
- ls_classes : list
  List of lineshape classes.
- p : ndarray
  Fit parameters.
- cov : ndarray
  Covariance matrix from least squares fitting.
- n_peaks : int
  Number of peaks in the region.
Returns
- errors : ndarray
  Array of standard errors of parameters in p.

nmrglue.analysis.linesh.s_NDregion

nmrglue.analysis.linesh.s_NDregion(p, shape, ls_classes, n_peaks)
Simulate an N-dimensional region with one or more peaks.

Parameters
- p : list
  List of parameters, must be a list, modified by function.
- shape : tuple of ints
Shape of region.

**ls_classes** : list
List of lineshape classes.

**n_peaks** : int
Number of peaks in region.

**Returns**
**r** : ndarray
Simulated region.

**nmrglue.analysis.linesh.s_single_NDregion**

*nmrglue.analysis.linesh.s_single_NDregion(p, shape, ls_classes)*

Simulate an N-dimensional region with a single peak.

This function is called repeatedly by *s_NDregion* to build up a full simulated region.

**Parameters**
**p** : list
List of parameters, must be a list.

**shape** : tuple
Shape of region.

**ls_classes** : list
List of lineshape classes.

**Returns**
**r** : ndarray
Simulated region.

**nmrglue.analysis.linesh.err_NDregion**

*nmrglue.analysis.linesh.err_NDregion(p, region, shape, ls_classes, n_peaks, wmask)*

Error function for an N-dimensional region, called by *f_NDregion()*

**nmrglue.analysis.linesh.f_NDregion**

*nmrglue.analysis.linesh.f_NDregion(region, ls_classes, p0, p_bounds, n_peaks, wmask, **kw)*

Fit an N-dimensional regions containing one or more peaks.

Region is fit using a constrained Levenberg-Marquardt optimization algorithm. See *fit_NDregion()* for additional documentation.

**Parameters**
**region** : ndarray
Region to fit.

**ls_classes** : list
List of lineshape classes.

**p0** : ndarray
Initial parameters.
**p_bounds** : list of tuples

List of (min, max) bounds for each element of p0.

**n_peaks** : int

Number of peaks in the simulated region.

**wmask** : ndarray

Array with same shape as region which is used to weight points in the error calculation, typically a boolean array is used to exclude certain points in the region.

**kw** : optional

Additional keywords passed to the scipy.optimize.leastsq function.

See also:

*fit_NDregion* Fit N-dimensional region with user friendly parameter.

### 3.3.7 nmrglue.peakpick

Peak picking routines, lineshape parameter guessing, and related functions.

This modules is imported as nmrglue.peakpick and can be called as such.

**User Functions**

```python
pick(data, pthres[, nthres, msep, ...])
```

Pick (find) peaks in a region of a NMR spectrum.

#### nmrglue.analysis.peakpick.pick

```python
nmrglue.analysis.peakpick.pick(data, pthres=None, nthres=None, msep=None, algorithm='connected', est_params=True, lineshapes=None, edge=None, diag=False, c_struc=None, c_ndil=0, cluster=True, table=True, axis_names=['A', 'Z', 'Y', 'X'])
```

Pick (find) peaks in a region of a NMR spectrum.

**Parameters**

**data** : ndarray

Region of NMR spectrum to pick peaks from.

**pthres** : float

Minimum peak height for positive peaks. None to not detect positive peaks.

**nthres** : float

Minimum peak height for negative peaks (typically a negative value). None to not detect negative peaks.

**msep** : tuple of ints, optional

N-tuple of minimum peak separations along each axis. Must be provided if algorithm is ‘thresh’ or ‘thresh-fast’.

**algorithm** : {'thres', thresh-fast', 'downward', 'connected'}, optional

Peak picking algorithm to use. Default is ‘connected’.

---

**3.3. analysis modules**

229
est_params : bool, optional

True to perform an estimate of linewidths and amplitude for all peaks picked. False, the default, will return only the peak locations.

lineshapes : list, optional

A list of lineshape classes or string shortcuts for each dimension. If not specified Gaussian type lineshapes with a FWHM linewidth parameter is assumed in each dimension. This parameter if only used if est_params is True.

diag : bool, optional

Yes to consider diagonal points to be touching in peak finding algorithm and clustering.

c_struc : ndarray, optional

Structure element to use when applying dilation on segments before applying clustering algorithm. None will apply a default square structure with connectivity one will be applied.

c_ndil : int, optional

Number of dilations to perform on segments before applying clustering algorithm.

cluster : bool, optional

True to cluster touching peaks. False does not apply clustering.

table : bool, optional

True to return a table. False to return lists.

axis_names : list, optional

List of axis names, the last n will be used for column name prefixes in table where n is the number of dimensions.

Returns:

locations : list, returned when table is False

Peak locations.

cluster_ids : list, returned when table is False and cluster is True

Cluster numbers for peaks.

scales : list, returned when table is False and est_params is True

Estimated peak scales (linewidths).

amps : list, returned when table is False and est_params is True

Estimated peak amplitudes.

table : recarray, returned when table is True

Table of request peak parameters.
Developer Functions

These functions are typically not used directly by users. Developers who want fine control over peak picking will be interested in these functions.
### nmrglue Documentation, Release 0.7-dev

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_edge(locations, edge)</code></td>
<td>Add edge to list of locations, returning a list of edge-added locations</td>
</tr>
<tr>
<td><code>clusters(data, locations, pthres, nthres[, ...])</code></td>
<td>Perform cluster analysis of peak locations.</td>
</tr>
<tr>
<td><code>pack_table(locations[, cluster_ids, scales, ...])</code></td>
<td>Create a table from peak information.</td>
</tr>
<tr>
<td><code>guess_params_slice(data, location, ...)</code></td>
<td>Guess the parameter of a peak in a segment.</td>
</tr>
<tr>
<td><code>extract_1d(data, location, axis)</code></td>
<td>Extract a 1D slice from data along axis at location</td>
</tr>
<tr>
<td><code>find_all_thres(data, thres, msep[, find_segs])</code></td>
<td>Peak pick a spectrum using a threshold-minimum distance algorithm.</td>
</tr>
<tr>
<td><code>find_all_nthres(data, thres, msep[, find_segs])</code></td>
<td>Peak pick a spectrum using a threshold-minimum distance algorithm.</td>
</tr>
<tr>
<td><code>find_all_thres_fast(data, thres, msep[, ...])</code></td>
<td>Fast version of <code>find_all_thres</code>.</td>
</tr>
<tr>
<td><code>find_all_nthres_fast(data, thres, msep[, ...])</code></td>
<td>Fast version of <code>find_all_nthres</code>.</td>
</tr>
<tr>
<td><code>find_pseg_slice(data, location, thres)</code></td>
<td>Find slices which define a segment in data above thres.</td>
</tr>
<tr>
<td><code>find_nseg_slice(data, location, thres)</code></td>
<td>Find slices which define a segment in data below thres.</td>
</tr>
</tbody>
</table>

### nmrglue.analysis.peakpick.add_edge

```python
nmrglue.analysis.peakpick.add_edge(locations, edge)
```

Add edge to list of locations, returning a list of edge-added locations

### nmrglue.analysis.peakpick.clusters

```python
nmrglue.analysis.peakpick.clusters(data, locations, pthres, nthres, d_struc=None, l_struc=None, ndil=0)
```

Perform cluster analysis of peak locations.

**Parameters**

- `data` : ndarray
  Array of data which has been peak picked.

- `locations` : list
  List of peak locations.

- `pthres` : float
  Postive peak threshold. None for no positive peaks.

- `nthres` : float
  Negative peak threshold. None for no negative peaks.

- `d_struc` : ndarray, optional
  Structure of binary dilation to apply on segments before clustering. None uses a square structure with connectivity of one.

- `l_struc` : ndarray, optional
  Structure to use for determining segment connectivity in clustering. None uses square structure with connectivity of one.

- `ndil` : int, optional
  Number of dilation to apply on segments before determining clusters.

**Returns**

- `cluster_ids` : list
  List of cluster number corresponding to peak locations.
**nmrglue.analysis.peakpick.pack_table**

Create a table from peak information.

**Parameters**

- **locations**: list
  - List of peak locations.

- **cluster_ids**: list, optional
  - List of cluster numbers. None will not include cluster number in the table.

- **scales**: list, optional
  - List of peak scales (lineweights). None will not include peak scales in the table.

- **amps**: list, optional
  - List of peak amplitudes. None will not include peak amplitudes in the table.

- **axis_names**: list, optional
  - List of axis names, the last n will be used for column name prefixes where n is the number of dimensions.

**Returns**

- **table**: recarray
  - nmrglue table with column representing peak parameters. Peak locations are given column names like ‘X_AXIS’, ‘Y_AXIS’, etc. Cluster_ids are given a column name of ‘cID’. Peak scales (lineweights) are given column names like ‘X_LW’, ‘Y_LW’. Peak amplitudes are given a column name of ‘VOL’.

**nmrglue.analysis.peakpick.guess_params_slice**

Guess the parameter of a peak in a segment.

**Parameters**

- **data**: ndarray
  - NMR data.

- **location**: tuple
  - Peak locations.

- **seg_slice**: list of slices
  - List slices which slice data to give the desired segment.

- **lineshapes**: list
  - List of lineshape classes.

**Returns**

- **location**: list
  - Peak locations.

- **scale**: list
  - Peak scales (lineweights).

- **amp**: list
  - Peak amplitudes.
nmrglue.analysis.peakpick.extract_1d

```
Extract a 1D slice from data along axis at location
```

nmrglue.analysis.peakpick.find_all_thres

```
Peak pick a spectrum using a threshold-minimum distance algorithm.

Find peaks (local maxima) in an arbitrary dimensional NMR spectra above a set threshold with a minimal distance between peaks. When the spectrum is small and multiple copies can fit into RAM use the _fast version of this function. Segments are found by finding the first point in each direction along each dimension which is below the threshold.

```
Parameters data : ndarray
    NMR data.

    thres : float
        Threshold value for minimum peak height

    msep : tuple
        Tuple of minimum peak seperations along each axis.

    find_segs : bool, optional
        True to find segments and return a list of slices which select that segment. False performs no segmentation discovery.

Returns locations : list
    List of peak locations

    seg_slices : list, optional
        List of slices which extract a region around each peak. Only returned when find_segs is True.
```

nmrglue.analysis.peakpick.find_all_nthres

```
Identical to find_all_thres except local minima are found below the given threshold. See find_all_thres() for a description of the algorithm and documentation.
```

nmrglue.analysis.peakpick.find_all_thres_fast

```
Fast version of find_all_thres. See find_all_thres().
```
nmrglue.analysis.peakpick.find_all_nthres_fast

nmrglue.analysis.peakpick.find_all_nthres_fast (data, thres, msep, find_segs=False)
Fast version of find_all_nthres_fast. See find_all_thres().

nmrglue.analysis.peakpick.find_pseg_slice

nmrglue.analysis.peakpick.find_pseg_slice (data, location, thres)
Find slices which define a segment in data above thres.

nmrglue.analysis.peakpick.find_nseg_slice

nmrglue.analysis.peakpick.find_nseg_slice (data, location, thres)
Find slices which define a segment in data below thres.

3.3.8 nmrglue.segmentation

Functions to perform segmentation of NMR spectrum.
This module is imported as nmrglue.segmentation and can be called as such.

Developer Functions

These functions are typically not used directly by users. Developers who want to segment spectra will be interested in these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find_all_connected</td>
<td>Find all connected segments</td>
</tr>
<tr>
<td>find_all_nconnected</td>
<td>Find all negatively connected segments in data</td>
</tr>
<tr>
<td>find_all_upward</td>
<td>Find all upward connected segments in data</td>
</tr>
<tr>
<td>label_connected</td>
<td>Label connected features in data</td>
</tr>
<tr>
<td>label_downward</td>
<td>Label downward-connected features in data</td>
</tr>
<tr>
<td>label_upward</td>
<td>Label upward connected features in data</td>
</tr>
<tr>
<td>find_downward</td>
<td>Find points downward-connected to a point in data</td>
</tr>
<tr>
<td>find_connected</td>
<td>Find points connected to pt in data below threshold.</td>
</tr>
<tr>
<td>find_nconnected</td>
<td>Find points upward-connected to pt in data</td>
</tr>
<tr>
<td>mark_dseg</td>
<td>Mark downward-connected region on segment map starting at node pt</td>
</tr>
<tr>
<td>label_downward_seg</td>
<td>Label a segment which is downward connected</td>
</tr>
<tr>
<td>mark_useg</td>
<td>Mark upward-connected region on segment map starting at node pt</td>
</tr>
<tr>
<td>label_upward_seg</td>
<td>Label a segment which is upward connected</td>
</tr>
</tbody>
</table>

3.3. analysis modules 235
**nmrglue.analysis.segmentation.find_all_connected**

Find all connected segments.

**Parameters**
- `data`: ndarray
  Data to perform segmentation on.
- `thres`: float
  Threshold, below this nodes are considered noise.
- `find_segs`: bool, optional
  True to return a list of slices for the segments.
- `diag`: bool
  True to include diagonal neighbors in connection.

**Returns**
- `locations`: list
  List of indicies of local maximum in each segment.
- `seg_slices`: list, optional
  List of slices which extract a given segment from the data. Only returned when `fig_segs` is True.

**nmrglue.analysis.segmentation.find_all_nconnected**

Find all negatively connected segments in data.

**Parameters**
- `data`: ndarray
  Data to perform segmentation on.
- `thres`: float
  Threshold, below this nodes are considered noise.
- `find_segs`: bool, optional
  True to return a list of slices for the segments.
- `diag`: bool
  True to include diagonal neighbors in connection.

**Returns**
- `locations`: list
  List of indicies of local maximum in each segment.
- `seg_slices`: list, optional
  List of slices which extract a given segment from the data. Only returned when `fig_segs` is True.
nmrglue.analysis.segmentation.find_all_downward

nmrglue.analysis.segmentation.find_all_downward(data, thres, find_segs=False, diag=False)

Find all downward connected segments in data

Parameters

- **data**: ndarray
  - Data to perform segmentation on.
- **thres**: float
  - Threshold, below this nodes are considered noise.
- **find_segs**: bool, optional
  - True to return a list of slices for the segments.
- **diag**: bool, optional
  - True to include diagonal neighbors in connection.

Returns

- **locations**: list
  - List of indices of local maximum in each segment.
- **seg_slices**: list, optional
  - List of slices which extract a given segment from the data. Only returned when fig_segs is True.

nmrglue.analysis.segmentation.find_all_upward

nmrglue.analysis.segmentation.find_all_upward(data, thres, find_segs=False, diag=False)

Find all upward connected segments in data

Parameters

- **data**: ndarray
  - Data to perform segmentation on.
- **thres**: float
  - Threshold, below this nodes are considered noise.
- **find_segs**: bool, optional
  - True to return a list of slices for the segments.
- **diag**: bool, optional
  - True to include diagonal neighbors in connection.

Returns

- **locations**: list
  - List of indices of local maximum in each segment.
- **seg_slices**: list, optional
  - List of slices which extract a given segment from the data. Only returned when fig_segs is True.
nmrglue.analysis.segmentation.label_connected

nmrglue.analysis.segmentation.label_connected(data, thres, structure)
Label connected features in data. Returns labeled_array, num_features

nmrglue.analysis.segmentation.label_nconnected

nmrglue.analysis.segmentation.label_nconnected(data, thres, structure)
Label nconnected features in data. Returns labeled_array, num_features

nmrglue.analysis.segmentation.label_downward

nmrglue.analysis.segmentation.label_downward(data, thres, structure)
Label connected features in data. Returns labeled_array, num_features

nmrglue.analysis.segmentation.label_upward

nmrglue.analysis.segmentation.label_upward(data, thres, structure)
Label upward connected features in data. Returns labeled_array, num_features

nmrglue.analysis.segmentation.find_downward

nmrglue.analysis.segmentation.find_downward(data, pt, thres, diag=False)
Find points downward-connected to a point in data.

Parameters
data : ndarray
    Array of data.
pt : tuple of ints
    Starting point of peak.
thres : float
    Threshold, below this nodes are considered noise.
diag : bool, optional
    True to include diagonal neighbors in connection.

Returns
nodes : list
    Indicies of downward-connected nodes.

nmrglue.analysis.segmentation.find_connected

nmrglue.analysis.segmentation.find_connected(data, pt, thres, diag=False)
Find points connected to a point in data.

Parameters
data : ndarray
    Array of data.
pt : tuple of ints
Starting point of peak.

**thres** : float
Threshold, below this nodes are considered noise.

**diag** : bool, optional
True to include diagonal neighbors in connection.

**Returns nodes** : list
Indicies of connected nodes.

**nmrglue.analysis.segmentation.find_nconnected**

**nmrglue.analysis.segmentation.find_nconnected**(data, pt, thres, diag=False)
Find points connected to pt in data below threshold.

**Parameters**

*data* : ndarray
Array of data.

*pt* : tuple of ints
Starting point of peak.

*thres* : float
Threshold, above this nodes are considered noise.

*diag* : bool, optional
True to include diagonal neighbors in connection.

**Returns nodes** : list
Indicies of connected nodes.

**nmrglue.analysis.segmentation.find_upward**

**nmrglue.analysis.segmentation.find_upward**(data, pt, thres, diag=False)
Find points upward-connected to pt in data.

**Parameters**

*data* : ndarray
Array of data.

*pt* : tuple of ints
Starting point of peak.

*thres* : float
Threshold, below this nodes are considered noise.

*diag* : bool, optional
True to include diagonal neighbors in connection.

**Returns nodes** : list
Indicies of upward-connected nodes.
nmrglue.analysis.segmentation.mark_dseg

nmrglue.analysis.segmentation.mark_dseg(mdata, map, pt, mark, structure)
Mark downward-connected region on segment map starting at node pt.
Modifies mdata mask and map.

Parameters
mdata : masked ndarray
   Masked data array.
map :
   Array mapping out segments.
pt : tuple of ints
   Index of starting node
mark : int
   Integer to mark map with.

nmrglue.analysis.segmentation.label_downward_seg

nmrglue.analysis.segmentation.label_downward_seg(data, labels, seg_slice, seg_index, max_index, structure)
Label a segment which is downward connected

nmrglue.analysis.segmentation.mark_useg

nmrglue.analysis.segmentation.mark_useg(mdata, map, pt, mark, structure)
Mark upward-connected region on segment map starting at node pt.
Modifies mdata mask and map.

Parameters
mdata : masked ndarray
   Masked data array.
map :
   Array mapping out segments.
pt : tuple of ints
   Index of starting node
mark : int
   Integer to mark map with.

nmrglue.analysis.segmentation.label_upward_seg

nmrglue.analysis.segmentation.label_upward_seg(data, labels, seg_slice, seg_index, max_index, structure)
Label a segment which is upward connected
3.4 util modules

3.4.1 nmrglue.misc

Misc. functions

This module is imported as nmrglue.misc and can be called as such.

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pair_similar</td>
<td>Check a dic, data pair against a second dic, data pair for differences.</td>
</tr>
<tr>
<td>isdatasimilar</td>
<td>Check that two sets of NMR data are equal within a tolerance.</td>
</tr>
<tr>
<td>isdicsimilar</td>
<td>Compare two dictionaries for differences</td>
</tr>
<tr>
<td>islistsimilar</td>
<td>Compare two lists (or iterable) for differences</td>
</tr>
<tr>
<td>isitemsimilar</td>
<td>Compare two values for differences</td>
</tr>
</tbody>
</table>

**nmrglue.util.misc.pair_similar**

**nmrglue.util.misc.pair_similar** *(dic1, data1, dic2, data2, verb=False, atol=0.1001, rtol=0.1001, dtol=0.5001, ignore_pipe_display=False)*

Check a dic, data pair against a second dic, data pair for differences.

**Parameters**

- **dic1**: dict
  - First dictionary of NMR parameters.
- **data1**: ndarray
  - First array of NMR data
- **dic2**: dict
  - Second dictionary of NMR parameters
- **data2**: ndarray
  - Second array of NMR data
- **verb**: bool, optional
  - Set True for verbose reporting.
- **atol**: float, optional
  - The absolute tolerance parameter to pass to numpy.allclose.
- **rtol**: float, optional
  - The relative tolerance parameter to pass to numpy.allclose.

**Returns**

- **r1**: bool
  - True is data1 and data2 are similar, False if they differ.
- **r2**: bool
  - True is dic1 and dic2 are similar, False if they differ.
nmrglue.util.misc.isdatasimilar

```
mrmglue.util.misc.isdatasimilar(data1, data2, verb=False, atol=0.1001, rtol=0.1001)
```

Check that two sets of NMR data are equal within a tolerance.

**Parameters**
- `data1`: ndarray
  - First array of NMR data
- `data2`: ndarray
  - Second array of NMR data
- `verb`: bool, optional
  - Set True for verbose reporting.
- `atol`: float, optional
  - The absolute tolerance parameter to pass to numpy.allclose.
- `rtol`: float, optional
  - The relative tolerance parameter to pass to numpy.allclose.

**Returns**
- `r1`: bool
  - True is data1 and data2 are similar, False if they differ.

nmrglue.util.misc.isdicsimilar

```
mrmglue.util.misc.isdicsimilar(dic1, dic2, verb=False, dtol=0.5001, ignore_pipe_display=False)
```

Compare two dictionaries for differences.

Float and int types compared within `dtol`. Lists and dictionaries are checked recursively all other checked by simple equivalence.

**Parameters**
- `dic1`: dict
  - First dictionary of NMR parameters.
- `dic2`: dict
  - Second dictionary of NMR parameters
- `verb`: bool, optional
  - Set True for verbose reporting.
- `dtol`: float, optional
  - Maximum allowable difference between int and float elements if dic1 and dic2.

**Returns**
- `r1`: bool
  - True is dic1 and dic2 are similar, False if they differ.

nmrglue.util.misc.islistsimilar

```
mrmglue.util.misc.islistsimilar(l1, l2, verb=False, dtol=0.5001)
```

Compare two lists (or iterable) for differences.

See `isdicsimilar()` for Parameters.
**nmrglue.util.misc.isitemsimilar**

```
nmrglue.util.misc.isitemsimilar(v1, v2, verb=False, dtol=0.5001)
```

Compare two values for differences

See `isdicsimilar()` for Parameters.
4.1 Application Examples

4.1.1 application example: apod_viewer_1win

This is a sample GUI application showing how nmrglue can be used with additional python modules like matplotlib and wxPython to create full fledged NMR applications.

In this application users can examine graphically the apodization windows produced by the various window functions supported by NMRPipe. In this example the canvas in which the apodization windows are drawn and the location to input the apodization parameter are contained in the same window. The application example: apod Viewer_2win example has the canvas and input area in separate windows.

[source code]

```python
#!/usr/bin/env python

""
An example of using wxPython to build a GUI application using nmrglue
This application displays NMRPipe apodization windows.
""

import wx
import numpy as np
import nmrglue as ng
import matplotlib

# uncomment the following to use wx rather than wxagg
#matplotlib.use('WX')
# from matplotlib.backends.backend_wx import FigureCanvasWx as FigureCanvas

# comment out the following to use wx rather than wxagg
matplotlib.use('WXAgg')
```
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
from matplotlib.backends.backend_wx import NavigationToolbar2Wx
from matplotlib.figure import Figure

apod_list = ["SP", "EM", "GM", "GMB", "TM", "TRI", "JMOD"]

class ParameterPanel(wx.Panel):
    """ WX panel where apodization parameters can be set by the user """
    def __init__(self, parent):
        """ Initialize the frame """
        wx.Panel.__init__(self, parent, -1)

        self.parent = parent

        # create the various elements in the panel
        self.qName1 = wx.StaticText(self, -1, "Type:")
        self.qName2 = wx.Choice(self, -1, choices=apod_list)
        self.Bind(wx.EVT_CHOICE, self.ApodChoose, self.qName2)

        self.q1_1 = wx.StaticText(self, -1, "q1:" )
        self.q1_2 = wx.TextCtrl(self, -1, "0.0")

        self.q2_1 = wx.StaticText(self, -1, "q2:" )
        self.q2_2 = wx.TextCtrl(self, -1, "1.0")

        self.q3_1 = wx.StaticText(self, -1, "q3:" )
        self.q3_2 = wx.TextCtrl(self, -1, "1.0")

        self.c1 = wx.StaticText(self, -1, "c")
        self.c2 = wx.TextCtrl(self, -1, "1.0")

        self.start_1 = wx.StaticText(self, -1, "Start")
        self.start_2 = wx.TextCtrl(self, -1, "1.0")

        self.size_1 = wx.StaticText(self, -1, "Size")
        self.size_1.Enable(False)
        self.size_2 = wx.TextCtrl(self, -1, "1.0")
        self.size_2.Enable(False)

        self.inv = wx.CheckBox(self, -1, "Invert")
        self.use_size = wx.CheckBox(self, -1, "Custom Size")
        self.Bind(wx.EVT_CHECKBOX, self.OnLimitCheck, self.use_size)

        self.points_1 = wx.StaticText(self, -1, "Number of Points:" )
        self.points_2 = wx.TextCtrl(self, -1, "1000")

        self.sw_1 = wx.StaticText(self, -1, "Spectral Width:" )
        self.sw_2 = wx.TextCtrl(self, -1, "50000.")

        self.b1 = wx.Button(self, 10, "Draw")
        self.Bind(wx.EVT_BUTTON, self.OnDraw, self.b1)
        self.b1.SetDefault()

        self.b2 = wx.Button(self, 20, "Clear")
        self.Bind(wx.EVT_BUTTON, self.OnClear, self.b2)
```python
self.b2.SetDefault()
self.InitApod("SP")

# layout of the panel
apod_grid = wx.GridSizer(8, 2)
apod_grid.AddMany([self.qName1, self.qName2,
                    self.q1_1, self.q1_2,
                    self.q2_1, self.q2_2,
                    self.q3_1, self.q3_2,
                    self.c1, self.c2,
                    self.start_1, self.start_2,
                    self.size_1, self.size_2,
                    self.inv, self.use_size])
data_grid = wx.GridSizer(2, 2)
data_grid.AddMany([self.points_1, self.points_2,
                    self.sw_1, self.sw_2])
apod_box = wx.StaticBoxSizer(wx.StaticBox(self, -1, "Apodization Parameters"))
apod_box.Add(apod_grid)
data_box = wx.StaticBoxSizer(wx.StaticBox(self, -1, "Data Parameters"))
data_box.Add(data_grid)
button_box = wx.GridSizer(1, 2)
button_box.AddMany([self.b1, self.b2])
mainbox = wx.BoxSizer(wx.VERTICAL)
mainbox.Add(apod_box)
mainbox.Add(data_box)
mainbox.Add(button_box)
self.SetSizer(mainbox)

def OnLimitCheck(self, event):
    """ Set the custom size when the check box is checked. """
    k = event.IsChecked()
    self.size_1.Enable(k)
    self.size_2.Enable(k)
    points = float(self.points_2.GetValue())
    self.size_2.SetValue(str(points))

def ApodChoose(self, event):
    """ Set the choosen apodization type """
    self.InitApod(apod_list[self.qName2.GetCurrentSelection()])

def InitApod(self, qName):
    """ Set the default parameter for a choosen apodization window """
    if qName == "SP":
        self.q1_1.Enable(True)
        self.q1_1.SetLabel("off")
        self.q1_2.Enable(True)
        self.q1_2.SetValue("0.0")

4.1. Application Examples 247
```
```python
self.q2_1.Enable(True)
self.q2_1.SetLabel("end")
self.q2_2.Enable(True)
self.q2_2.SetValue("1.0")

self.q3_1.Enable(True)
self.q3_1.SetLabel("pow")
self.q3_2.Enable(True)
self.q3_2.SetValue("1.0")

elif qName == "EM":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("lb (Hz)")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")

    self.q2_1.Enable(False)
    self.q2_2.Enable(False)

    self.q3_1.Enable(False)
    self.q3_2.Enable(False)

elif qName == "GM":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("g1 (Hz)")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")

    self.q2_1.Enable(True)
    self.q2_1.SetLabel("g2 (Hz)")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")

    self.q3_1.Enable(True)
    self.q3_1.SetLabel("g3")
    self.q3_2.Enable(True)
    self.q3_2.SetValue("0.0")

elif qName == "GMB":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("lb")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")

    self.q2_1.Enable(True)
    self.q2_1.SetLabel("gb")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")

    self.q3_1.Enable(False)
    self.q3_2.Enable(False)

elif qName == "TM":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("t1")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")
```

248 Chapter 4. Examples
self.q2_1.Enable(True)
self.q2_1.SetLabel("t2")
self.q2_2.Enable(True)
self.q2_2.SetValue("0.0")

self.q3_1.Enable(False)
self.q3_2.Enable(False)

elif qName == "TRI":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("loc")
    self.q1_2.Enable(True)
    points = float(self.points_2.GetValue())
    self.q1_2.SetValue(str(points / 2.))

    self.q2_1.Enable(True)
    self.q2_1.SetLabel("lHi")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")

    self.q3_1.Enable(True)
    self.q3_1.SetLabel("rHi")
    self.q3_2.Enable(True)
    self.q3_2.SetValue("0.0")

elif qName == "JMOD":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("off")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")

    self.q2_1.Enable(True)
    self.q2_1.SetLabel("j (Hz)")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")

    self.q3_1.Enable(True)
    self.q3_1.SetLabel("lb (Hz)")
    self.q3_2.Enable(True)
    self.q3_2.SetValue("0.0")

def OnDraw(self, event):
    """ Draw the apodization window ""
    qName = apod_list[self.qName2.GetCurrentSelection()]
    q1 = float(self.q1_2.GetValue())
    q2 = float(self.q2_2.GetValue())
    q3 = float(self.q3_2.GetValue())
    c = float(self.c2.GetValue())
    start = float(self.start_2.GetValue())
    size = float(self.size_2.GetValue())

    inv = self.inv.GetValue()
    use_size = self.use_size.GetValue()

    points = float(self.points_2.GetValue())
    sw = float(self.sw_2.GetValue())

    self.parent.ApplyApod(qName, q1, q2, q3, c, start, size, inv,
```python
use_size, points, sw)

def OnClear(self, event):
    """ Clear all apodization windows previously drawn """
    self.parent.ClearFigure()

class CanvasFrame(wx.Frame):
    """
    WX frame containing a matplotlib canvas where the apodization windows are
drawn and the parameter panel where apodization parameter can be set.
    """
    def __init__(self):
        """ Initialize the frame """
        wx.Frame.__init__(self, None, -1, 'Apodization Viewer')

        self.SetBackgroundColour(wx.NamedColor("WHITE"))

        self.figure = Figure()
        self.axes = self.figure.add_subplot(111)
        self.canvas = FigureCanvas(self, -1, self.figure)
        self.params = ParameterPanel(self)
        self.toolbar = NavigationToolbar2Wx(self.canvas)
        self.toolbar.Realize()

        # layout
        fsizer = wx.BoxSizer(wx.VERTICAL)
        fsizer.Add(self.canvas, 0, wx.EXPAND)
        fsizer.Add(self.toolbar, 0, wx.EXPAND)

        self.sizer = wx.BoxSizer(wx.HORIZONTAL)
        self.sizer.Add(self.params, 0, wx.EXPAND)
        self.sizer.Add(fsizer, 0, wx.EXPAND)

        self.SetSizer(self.sizer)
        self.Fit()

def OnPaint(self, event):
    """ Draw the apodization window """
    self.canvas.draw()

def ClearFigure(self):
    """ Clear all previously drawn apodization windows """
    self.axes.cla()
    self.OnPaint(-1)

def ApplyApod(self, qName, q1, q2, q3, c, start, size, inv, use_size,
           points, sw):
    """ Apply the selected apodization, draw the windows on the canvas """

    # create the dictionary
    dic = ng.fileiobase.create_blank_udic(1)
    dic[0]["sw"] = sw
    dic[0]["size"] = points

    # create the data
    data = np.ones(points, dtype="complex")

    # convert to NMRPipe format
```
4.1.2 application example: apod_viewer_2win

This is a sample GUI application showing how nmrglue can be used with additional python modules like matplotlib and wxPython to create full fledged NMR applications.

In this application users can examine graphically the apodization windows produced by the various window functions supported by NMRPipe. In this example the canvas in which the apodization windows are drawn and the location to input the apodization parameter are contained in two separate same window. The application example: apod_viewer_1win example has the canvas and input area in a single window.

[source code]

```python
#!/usr/bin/env python

***
An example of using wxPython to build a GUI application using nmrglue
This application displays NMRPipe apodization windows.
***

import wx
import numpy as np
import nmrglue as ng
import matplotlib

# uncomment the following to use wx rather than wxagg
#matplotlib.use('WX')
# from matplotlib.backends.backend_wx import FigureCanvasWx as FigureCanvas

# comment out the following to use wx rather than wxagg
matplotlib.use('WXAgg')
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
```
from matplotlib.backends.backend_wx import NavigationToolbar2Wx
from matplotlib.figure import Figure

apod_list = ['SP', 'EM', 'GM', 'GMB', 'TM', 'TRI', 'JMOD']

class ParameterFrame(wx.Frame):
    """ WX frame where apodization parameters can be set by the user """

    def __init__(self, parent, id):
        """ Initialize the frame """
        wx.Frame.__init__(self, parent, -1, "Parameters",
                          style=wx.DEFAULT_DIALOG_STYLE | wx.RESIZE_BORDER)
        self.parent = parent

        # create the various elements in the panel
        self.qName1 = wx.StaticText(self, -1, "Type:")
        self.qName2 = wx.Choice(self, -1, choices=apod_list)
        self.Bind(wx.EVT_CHOICE, self.ApodChoose, self.qName2)
        self.q1_1 = wx.StaticText(self, -1, "q1:")
        self.q1_2 = wx.TextCtrl(self, -1, "0.0")
        self.q2_1 = wx.StaticText(self, -1, "q2:")
        self.q2_2 = wx.TextCtrl(self, -1, "1.0")
        self.q3_1 = wx.StaticText(self, -1, "q3:")
        self.q3_2 = wx.TextCtrl(self, -1, "1.0")
        self.c1 = wx.StaticText(self, -1, "c")
        self.c2 = wx.TextCtrl(self, -1, "1.0")
        self.start_1 = wx.StaticText(self, -1, "Start")
        self.start_2 = wx.TextCtrl(self, -1, "1.0")
        self.size_1.Enable(False)
        self.size_2 = wx.TextCtrl(self, -1, "1.0")
        self.size_2.Enable(False)
        self.inv = wx.CheckBox(self, -1, "Invert")
        self.use_size = wx.CheckBox(self, -1, "Custom Size")
        self.Bind(wx.EVT_CHECKBOX, self.OnLimitCheck, self.use_size)
        self.points_1 = wx.StaticText(self, -1, "Number of Points:")
        self.points_2 = wx.TextCtrl(self, -1, "1000")
        self.sw_1 = wx.StaticText(self, -1, "Spectral Width:")
        self.sw_2 = wx.TextCtrl(self, -1, "50000.")
        self.b1 = wx.Button(self, 10, "Draw")
        self.Bind(wx.EVT_BUTTON, self.OnDraw, self.b1)
        self.b1.SetDefault()
        self.b2 = wx.Button(self, 20, "Clear")
        self.Bind(wx.EVT_BUTTON, self.OnClear, self.b2)
self.b2.SetDefault()

self.InitApod("SP")

# layout of the panel
apod_grid = wx.GridSizer(8, 2)

apod_grid.AddMany([self.qName1, self.qName2,
                   self.q1_1, self.q1_2,
                   self.q2_1, self.q2_2,
                   self.q3_1, self.q3_2,
                   self.c1, self.c2,
                   self.start_1, self.start_2,
                   self.size_1, self.size_2,
                   self.inv, self.use_size])

data_grid = wx.GridSizer(2, 2)
data_grid.AddMany([self.points_1, self.points_2,
                   self.sw_1, self.sw_2])

apod_box = wx.StaticBoxSizer(wx.StaticBox(self, -1, "Apodization Parameters"))
apod_box.Add(apod_grid)

data_box = wx.StaticBoxSizer(wx.StaticBox(self, -1, "Data Parameters"))
data_box.Add(data_grid)

button_box = wx.GridSizer(1, 2)
button_box.AddMany([self.b1, self.b2])

mainbox = wx.BoxSizer(wx.VERTICAL)
mainbox.Add(apod_box)
mainbox.Add(data_box)
mainbox.Add(button_box)
self.SetSizer(mainbox)

self.Fit()
self.SetMinSize(self.GetSize())

def OnLimitCheck(self, event):
    
    
    k = event.IsChecked()
    self.size_1.Enable(k)
    self.size_2.Enable(k)
    points = float(self.points_2.GetValue())
    self.size_2.SetValue(str(points))

def ApodChoose(self, event):
    
    
    self.InitApod(apod_list[self.qName2.GetCurrentSelection()])

def InitApod(self, qName):
    
    
    if qName == "SP":
        self.q1_1.Enable(True)
        self.q1_1.SetLabel("off")

4.1. Application Examples
self.q1_2.Enable(True)
self.q1_2.SetValue("0.0")

self.q2_1.Enable(True)
self.q2_1.SetLabel("end")
self.q2_2.Enable(True)
self.q2_2.SetValue("1.0")

self.q3_1.Enable(True)
self.q3_1.SetLabel("pow")
self.q3_2.Enable(True)
self.q3_2.SetValue("1.0")

elif qName == "EM":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("lb (Hz)")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")
    self.q2_1.Enable(False)
    self.q2_2.Enable(False)
    self.q3_1.Enable(False)
    self.q3_2.Enable(False)

elif qName == "GM":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("g1 (Hz)")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")
    self.q2_1.Enable(True)
    self.q2_1.SetLabel("g2 (Hz)")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")
    self.q3_1.Enable(True)
    self.q3_1.SetLabel("g3")
    self.q3_2.Enable(True)
    self.q3_2.SetValue("0.0")

elif qName == "GMB":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("lb")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")
    self.q2_1.Enable(True)
    self.q2_1.SetLabel("gb")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")
    self.q3_1.Enable(False)
    self.q3_2.Enable(False)

elif qName == "TM":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("t1")
self.q1_2.Enable(True)
self.q1_2.SetValue("0.0")

self.q2_1.Enable(True)
self.q2_1.SetLabel("t2")
self.q2_2.Enable(True)
self.q2_2.SetValue("0.0")

self.q3_1.Enable(False)
self.q3_2.Enable(False)

elif qName == "TRI":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("loc")
    self.q1_2.Enable(True)
    points = points = float(self.points_2.GetValue())
    self.q1_2.SetValue(str(points / 2.))

    self.q2_1.Enable(True)
    self.q2_1.SetLabel("lHi")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")

    self.q3_1.Enable(True)
    self.q3_1.SetLabel("rHi")
    self.q3_2.Enable(True)
    self.q3_2.SetValue("0.0")

elif qName == "JMOD":
    self.q1_1.Enable(True)
    self.q1_1.SetLabel("off")
    self.q1_2.Enable(True)
    self.q1_2.SetValue("0.0")

    self.q2_1.Enable(True)
    self.q2_1.SetLabel("j (Hz)")
    self.q2_2.Enable(True)
    self.q2_2.SetValue("0.0")

    self.q3_1.Enable(True)
    self.q3_1.SetLabel("lb (Hz)")
    self.q3_2.Enable(True)
    self.q3_2.SetValue("0.0")

def OnDraw(self, event):
    
    """ Draw the apodization window ""
    qName = apod_list[self.qName2.GetCurrentSelection()]
    q1 = float(self.q1_2.GetValue())
    q2 = float(self.q2_2.GetValue())
    q3 = float(self.q3_2.GetValue())
    c = float(self.c2.GetValue())
    start = float(self.start_2.GetValue())
    size = float(self.size_2.GetValue())

    inv = self.inv.GetValue()
    use_size = self.use_size.GetValue()

    points = float(self.points_2.GetValue())
sw = float(self.sw_2.GetValue())

self.parent.ApplyApod(qName, q1, q2, q3, c, start, size, inv,
                      use_size, points, sw)

def OnClear(self, event):
    """ Clear all apodization windows previously drawn """
    self.parent.ClearFigure()

class CanvasFrame(wx.Frame):
    """ WX frame containing a matplotlib canvas where the apodization windows are
    drawn. Launches the parameter frame where apodization parameter can be set.
    """
    def __init__(self):
        """ Initializate the frame """
        wx.Frame.__init__(self, None, -1, 'Apodization Viewer')

        self.SetBackgroundColour(wx.NamedColor("WHITE"))

        self.figure = Figure()
        self.axes = self.figure.add_subplot(111)
        self.canvas = FigureCanvas(self, -1, self.figure)
        self.toolbar = NavigationToolbar2Wx(self.canvas)
        self.toolbar.Realize()

        # open parameter window
        win = ParameterFrame(self, -1)
        win.Show(True)

        # layout
        self.sizer = wx.BoxSizer(wx.VERTICAL)
        self.sizer.Add(self.canvas, 1, wx.LEFT| wx.TOP | wx.GROW)
        self.sizer.Add(self.toolbar, 0, wx.LEFT| wx.EXPAND)
        self.SetSizer(self.sizer)
        self.Fit()

    def OnPaint(self, event):
        """ Draw the apodization window """
        self.canvas.draw()

    def ClearFigure(self):
        """ Clear all previously drawn apodization windows """
        self.axes.cla()
        self.OnPaint(-1)

    def ApplyApod(self, qName, q1, q2, q3, c, start, size, inv, use_size,
                  points, sw):
        """ Apply the selected apodization, draw the windows on the canvas """

        # create the dictionary
        dic = ng.fileiobase.create_blank_udic(1)
        dic[0]["sw"] = sw
        dic[0]["size"] = points

        # create the data
        data = np.ones(points, dtype="complex")
4.2 Coadd Examples

4.2.1 coadd example: coadd_1d_pipe

This example shows how to use nmrglue to coadd a series of 1D NMRPipe files. All files matching the test*.fid pattern will be coadded into a file named coadded.fid.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng
import numpy as np
import glob

# create a list of files to coadd
flist = glob.glob("test*.fid")
flist.sort()

# initialize the new data
dic, data = ng.pipe.read(flist[0])
coadd_data = np.zeros_like(data)
coadd_dic = dict(dic)

# loop over the files, adding them to the coadded array
for f in flist:
    dic, data = ng.pipe.read(f)
    coadd_data += data

# convert to NMRPipe format
C = ng.convert.converter()
pdic, pdata = C.to_pipe()

if use_size == True:
tsize = size
else:
tsize = 'default'
nul, apod_data = ng.pipe_proc.apod(pdic, pdata, qName=qName,
    q1=q1, q2=q2, q3=q3, c=c, inv=inv, size=tsize, start=start)

# draw the window
self.axes.cla()
self.axes.plot(apod_data)
self.OnPaint(-1)
```

```python
class App(wx.App):
    """ Wx application """
    def OnInit(self):
        """ Create the main window and insert the custom frame """
        frame = CanvasFrame()
        frame.Show(True)
        return True

app = App(0)
app.MainLoop()
```
print "Coadding file:", f
dic, data = ng.pipe.read(f)
coadd_data += data / len(flist)

# write out the file
print "Writing out coadded data"
ng.pipe.write("coadded.fid", coadd_dic, coadd_data, overwrite=True)

4.2.2 coadd example: coadd_2d_pipe

This example shows how to use nmrglue to coadd a series of 2d NMRPipe files. All files matching the test*.fid pattern will be coadded into a file named coadded.fid.

The data used in this example is available for download.
[source code]

import nmrglue as ng
import numpy as np
import glob

# create a list of files to coadd
flist = glob.glob("test*.fid")
flist.sort()

# initialize the new data
dic, data = ng.pipe.read(flist[0])
coadd_data = np.zeros_like(data)
coadd_dic = dict(dic)

# loop over the files, adding them to the coadded array
for f in flist:
    print "Coadding file:", f
dic, data = ng.pipe.read(f)
coadd_data += data / len(flist)

# write out the file
print "Writing out coadded data"
ng.pipe.write("coadded.fid", coadd_dic, coadd_data, overwrite=True)

4.2.3 coadd example: coadd_pseudo3d_pipe

This example shows how to use nmrglue to coadd a number of collections of 2D NMRPipe files which constitute a pseudo-3D data set. The two (or more) psuedo-3D data sets are assumed to be in directories named run*.dir with subdirectories named *.fid containing a test.fid file. The directory coadded_data.dir is created with the same subdirectory structure containing test.fid files containing data created by coadding each pseudo-3D.

The data used in this example is available for download.
[source code]

import nmrglue as ng
import numpy as np
import glob
import os.path

# create a list of directories to coadd
dlist = glob.glob("run*.dir")
dlist.sort()

# create a list of 2D files in the first directory
flist = glob.glob(os.path.join(dlist[0], "*.fid", "test.fid"))
flist.sort()

# loop over the files
for base_fname in flist:
    # initialize the new data
dic, data = ng.pipe.read(base_fname)
coadd_data = np.zeros_like(data)
coadd_dic = dict(dic)

    # loop over the pseudo-3D directories
    for d in dlist:
        # the file name is found by replace the directory name
        f = base_fname.replace(dlist[0], d, 1)
        print "Coadding file:", f
        dic, data = ng.pipe.read(f)
        coadd_data += data / len(dlist)

    # write out the file
    of = base_fname.replace(dlist[0], "coadded_data.dir", 1)
    print "Writing out:", of
    ng.pipe.write(of, coadd_dic, coadd_data, overwrite=True)

4.3 Convert Examples

4.3.1 convert example: agilent2pipe_1d

This example shows how to use nmrglue to convert a 1D Agilent/Varian data set into a NMRPipe file.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# read in the Agilent data
dic, data = ng.varian.read('agilent_1d')

# Set the spectral parameters.
udic = ng.varian.guess_udic(dic, data)
udic[0]['size'] = 1500
udic[0]['complex'] = True
udic[0]['encoding'] = 'direct'
```
4.3.2 convert example: agilent2pipe_2d

This example shows how to use nmrglue to convert a 2D Agilent/Varian data set into a NMRPipe file.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# read in the Agilent data
dic, data = ng.varian.read("agilent_2d")

# Set the spectral parameters
udic = ng.varian.guess_udic(dic, data)

# Direct dimension		# Indirect dimension
udic[1]['size'] = 1500	; udic[0]['size'] = 332
udic[1]['complex'] = True	; udic[0]['complex'] = True
udic[1]['encoding'] = 'direct'	; udic[0]['encoding'] = 'states'
udic[1]['sw'] = 50000.0	; udic[0]['sw'] = 5555.556
udic[1]['obs'] = 125.691	; udic[0]['obs'] = 50.648
udic[1]['car'] = 55.0 * 125.691; udic[0]['car'] = 120.0 * 50.648
udic[1]['label'] = '13C'	; udic[0]['label'] = '15N'

# create the converter object and initialize with Agilent data
C = ng.convert.converter()
C.from_varian(dic, data, udic)

# create NMRPipe data and then write it out
ng.pipe.write("2d_pipe.fid", *C.to_pipe(), overwrite=True)
```

4.3.3 convert example: agilent2pipe_2d_tppi

This example shows how to use nmrglue to convert between the 2D Agilent/Varian data set collected using TPPI in the indirect dimension into a NMRPipe file.

The data used in this example is available for download.

[source code]
#!/usr/bin/env python

import nmrglue as ng

# read in the Agilent data
dic, data = ng.varian.read("agilent_2d_tppi")

# Set the spectral parameters.
u = ng.varian.guess_udic(dic, data)

# Direct Dimension
u[1]['size'] = 1400 ; u[0]['size'] = 600
u[1]['complex'] = True ; u[0]['complex'] = False
u[1]['encoding'] = 'direct' ; u[0]['encoding'] = 'tppi'
u[1]['sw'] = 50000.0 ; u[0]['sw'] = 33333.333
u[1]['obs'] = 125.681 ; u[0]['obs'] = 125.681
u[1]['car'] = 101.274 * 125.681 ; u[0]['car'] = 101.274 * 125.681
u[1]['label'] = 'C13x' ; u[0]['label'] = 'C13y'

# create the converter object and initialize with Agilent data
C = ng.convert.converter()
C.from_varian(dic, data, u)

# create NMRPipe data and then write it out.
ng.pipe.write("2d_pipe_tppi.fid", *C.to_pipe(), overwrite=True)

4.3.4 convert example: agilent2pipe_3d

This example shows how to use nmrglue to convert a 3D Agilent/Varian data set into a NMRPipe file.
The data used in this example is available for download.
[source code]

#!/usr/bin/env python

import nmrglue as ng

# read in the Agilent data (any of the follow lines will work)
#dic, data=ng.varian.read("agilent_3d")
dic, data=ng.varian.read_lowmem("agilent_3d")

# Set the spectral parameters
udic = ng.varian.guess_udic(dic, data)

# Direct Dimension
udic[2]['size'] = 1250
udic[2]['complex'] = True
udic[2]['encoding'] = 'direct'
udic[2]['sw'] = 50000.0
udic[2]['obs'] = 125.676
udic[2]['car'] = 56.0 * 125.676
udic[2]['label'] = 'CX'

# First indirect dimension
udic[1]['size'] = 88
udic[1]['complex'] = True

4.3. Convert Examples
4.3.5 **convert example: bruker2pipe_1d**

This example shows how to use nmrglue to convert a 1D Bruker data set into a NMRPipe file.

The data used in this example is available for download.

[source code]

```python
#! /usr/bin/env python

import nmrglue as ng

def read_brucker(d, data):
    dic, data = ng.bruker.read("bruker_1d")

    # Set the spectral parameters.
    udic = ng.bruker.guess_udic(dic, data)
    udic[0]['size'] = 2048
    udic[0]['complex'] = True
    udic[0]['encoding'] = 'direct'
    udic[0]['sw'] = 10000.000
    udic[0]['obs'] = 600.133
    udic[0]['car'] = 4.773 * 600.133
    udic[0]['label'] = '1H'

    # create the converter object and initialize with Bruker data
    C = ng.convert.converter()
    C.from_bruker(dic, data, udic)

    # create NMRPipe data and then write it out
    ng.pipe.write("1d_pipe.fid", C.to_pipe(), overwrite=True)
```
4.3.6 convert example: bruker2pipe_2d

This example shows how to use nmrglue to convert a 2D Bruker data set into a NMRPipe file. The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# read in the Bruker data
dic, data = ng.bruker.read("bruker_2d")

# Set the spectral parameters
u = ng.bruker.guess_udic(dic, data)

# Direct Dimension
u[1]['size'] = 768 ; u[0]['size'] = 600
u[1]['complex'] = True ; u[0]['complex'] = True
u[1]['encoding'] = 'direct' ; u[0]['encoding'] = 'states'

# Indirect Dimension
u[1]['sw'] = 11061.947 ; u[0]['sw'] = 4000.000
u[1]['obs'] = 800.134 ; u[0]['obs'] = 201.204
u[1]['car'] = 4.773 * 800.134 ; u[0]['car'] = 58.742 * 201.204
u[1]['label'] = '1H' ; u[0]['label'] = '13C'

# create the converter object and initialize with Bruker data
C = ng.convert.converter()
C.from_bruker(dic, data, u)

# create NMRPipe data and then write it out
ng.pipe.write("2d_pipe.fid", *C.to_pipe(), overwrite=True)
```

4.3.7 convert example: bruker2pipe_3d

This example shows how to use nmrglue to convert a 3D Bruker data set into a NMRPipe file. The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# read in the Bruker data
dic, data = ng.bruker.read("bruker_3d")
#dic, data = ng.bruker.read_lowmem("bruker_3d")

# Set the spectral parameters
udic = ng.bruker.guess_udic(dic, data)

# Direct Dimension
udic[2]['size'] = 768
udic[2]['complex'] = True
udic[2]['encoding'] = 'direct'
udic[2]['sw'] = 11061.947
```

4.3. Convert Examples
udic[2]['obs'] = 800.134
udic[2]['car'] = 4.784 * 800.134
udic[2]['label'] = '1H'

# First indirect dimension
udic[1]['size'] = 128
udic[1]['complex'] = True
udic[1]['encoding'] = 'states'
udic[1]['sw'] = 2500.000
udic[1]['obs'] = 81.086
udic[1]['car'] = 119.787 * 81.086
udic[1]['label'] = '15N'

# Second indirect dimension
udic[0]['size'] = 116
udic[0]['complex'] = True
udic[0]['encoding'] = 'states'
udic[0]['sw'] = 5555.556
udic[0]['obs'] = 201.204
udic[0]['car'] = 55.743 * 201.204
udic[0]['label'] = '13C'

# create the converter object and initialize with Bruker data
C = ng.convert.converter()
C.from_bruker(dic, data, udic)

# create NMRPipe data and then write it out
ng.pipe.write("./data/3d_pipe%03d.fid", *C.to_pipe(), overwrite=True)

4.3.8 convert example: pipe2sparky_2d

This example shows how to use nmrglue to convert a 2D NMRPipe file into a Sparky formatted file.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# read in the NMRPipe data
dic, data = ng.pipe.read("nrmpipe_2d/test.ft2")

# Set the spectral parameters
udic = ng.pipe.guess_udic(dic, data)

# create the converter object and initialize with NMRPipe data
C = ng.convert.converter()
C.from_pipe(dic, data, udic)

# create Sparky data and then write it out
ng.sparky.write("sparky_2d.ucsf", *C.to_sparky(), overwrite=True)
```
4.3.9 convert example: pipe2sparky_3d

This example shows how to use nmrglue to convert a 3D NMRPipe file into a Sparky formatted file. The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# read in the NMRPipe data (either line will work)
dic, data = ng.pipe.read("nmrpipe_3d/ft/test%03d.ft3")
#dic, data = ng.pipe.read_lowmem("nmrpipe_3d/ft/test%03d.ft3")

# Set the spectral parameters
udic = ng.pipe.guess_udic(dic, data)

# create the converter object and initialize with NMRPipe data
C = ng.convert.converter()
C.from_pipe(dic, data, udic)

# create Sparky data and then write it out
ng.sparky.write("sparky_3d.ucsf", *C.to_sparky(), overwrite=True)
```

4.4 Fitting Examples

4.4.1 fitting example: fitting_t1_data

This example shows how to use nmrglue and the SciPy optimize module to fit T1 relaxation trajectories. Three scripts are used in the process. The data used in this example is available for download.

First the `extract_trajs.py` script reads in box limits from `boxes.in` and a list of spectra from `spectra.in`. The script integrates each peak in each spectrum and writes the trajectory for each peak to disk as `traj.npy` in NumPy format.

[extract_trajs.py]

```python
#!/usr/bin/env python
# Script to extract trajectories from a series a 2D spectrum.

import nmrglue as ng
import numpy as np

# read in the integration limits and list of spectra
peak_list = np.recfromtxt("boxes.in")
spectra_list = np.recfromtxt("spectra.in")

# prepare the trajs records array
num_spec = spectra_list.size
num_peaks = peak_list.size
elist = [np.empty(num_spec, dtype="float") for i in xrange(num_peaks)]
trajs = np.rec.array(elist, names=list(peak_list.f0))
```
# loop over the spectra
for sn, spectra in enumerate(spectra_list):

    # read in the data from a NMRPipe file
    print "Extracting from:", spectra
    dic, data = ng.pipe.read(spectra)

    # loop over the integration limits
    for name, x0, y0, x1, y1 in peak_list:

        # integrate the region and save in trajs record array
        if x0 > x1:
            x0, x1 = x1, x0
        if y0 > y1:
            y0, y1 = y1, y0
        trajs[name][sn] = data[y0:y1 + 1, x0:x1 + 1].sum()

# normalize each trajectory
for peak in trajs.dtype.names:
    trajs[peak] = trajs[peak] / trajs[peak].max()

# save the trajectories records array to disk
np.save("traj.npy", trajs)

[boxes.in]

Peak  X0  Y0  X1  Y1
A20  4068  938  4079  913
A24  3992  1013  4000  997
A26  4065  962  4075  940
A34  4009  985  4018  958
A48  4028  1034  4036  1010
C28  4035  1115  4044  1092
D36  3994  987  4003  973
D40  4076  802  4085  774
D46  4155  987  4163  883
D47  4053  967  4062  941
E15  4162  1022  4170  996
E19  4176  902  4185  875
E27  4036  1084  4044  1054
E42  4136  1055  4142  1026
E56  4107  821  4115  794
F30  4013  1060  4023  1031
F52  4097  828  4105  799
G09  4054  1249  4063  1220
G14  4068  1331  4077  1304
G38  4098  1254  4106  1227
G41  4091  1283  4099  1259
I06  4087  903  4096  884

[spectra.in]

data/Ytau_100.fid/test.ft2
data/Ytau_100000.fid/test.ft2
data/Ytau_250000.fid/test.ft2
data/Ytau_500000.fid/test.ft2
data/Ytau_750000.fid/test.ft2
data/Ytau_1000000.fid/test.ft2
The second script `fit_exp_leastsq.py` reads in this `traj.npy` file and the T1 relaxation times associated with the spectra collected from `time.dat`. Each trajectory is fit using the least squares approach. Other optimization algorithms can be substituted with small changes to the code, see the `scipy.optimize` documentation). The resulting fits are saved to a `fits.pickle` file for easy reading into python as well as the human readable `fits.txt` file.

```python
#!/usr/bin/env python
# fit a collection to T1 trajectories to a decaying exponential
import scipy.optimize
import numpy as np
import pickle

# read in the trajectories and times
trajs = np.load("traj.npy")
t1 = np.recfromtxt("time.dat")

# fitting function and residual calculation
def fit_func(p, x):
    A, R2 = p
    # bound A between 0.98 and 1.02 (although fits do not reflect this)
    if A > 1.02:
        A = 1.02
    if A < 0.98:
        A = 0.98
    return A * np.exp(-1.0 * np.array(x) * R2 / 1.0e6)

def residuals(p, y, x):
    err = y - fit_func(p, x)
    return err

p0 = [1.0, 0.05]  # initial guess
fits = {}
# loop over the peak trajectories
for peak in trajs.dtype.names:
    print "Fitting Peak:", peak
    # get the trajectory to fit
    traj = trajs[peak]
    # fit the trajectory using leastsq (fmin, etc can also be used)
    results = scipy.optimize.leastsq(residuals, p0, args=(traj, t1))
    fits[peak] = results

# pickle the fits
f = open("fits.pickle", 'w')
pickle.dump(fits, f)
f.close()
```

4.4. Fitting Examples
# output the fits nicely to file
f = open("fits.txt", 'w')
f.write("#Peak\tA\tR2\tiер\n")
for k, v in fits.iteritems():
    f.write(k + "\t" + str(v[0][0]) + "\t" + str(v[0][1]) + "\t" + str(v[1]) + "\n")
f.close()

The last script pt.py reads in the fits, trajectories and T1 relaxation times and plots the experimental points and best fit to a series of *.plot.png files.

[pt.py]

`#!/usr/bin/env python`
`# Plot trajectories and fitting results`

`import pickle`
import matplotlib.pyplot as plt
import numpy as np

# the same fit_func as in fit_exp_leastsq.py
def fit_func(p, x):
    A, R2 = p

    # bound A between 0.98 and 1.02 (although fits do not reflect this)
    if A > 1.02:
        A = 1.02
    if A < 0.98:
        A = 0.98

    return A * np.exp(-1.0 * np.array(x) * R2 / 1.0e6)

# read in the trajectories, fitting results, and times
fits = pickle.load(open("fits.pickle"))
trajs = np.load("traj.npy")
times = np.recfromtxt("time.dat")

sim_times = np.linspace(times[0], times[-1], 2000)

# loop over the peaks
for peak, params in fits.iteritems():
    print "Plotting:", peak
    exp_traj = trajs[peak]
    sim_traj = fit_func(params[0], sim_times)

    # create the figure
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(times, exp_traj, 'or')
    ax.plot(sim_times, sim_traj, '-k')
    ax.set_title(peak)

    # save the figure
    fig.savefig(peak + "_plot.png")

Results:
[A24_plot.png]
4.5 Integration Examples

4.5.1 integration example: integrate_1d

This example shows how to use nmrglue to integrate a 1D NMRPipe spectra. The script reads in ppm peak limits from limits.in and takes a simple summation integral of each peak using the spectra contained in 1d_data.ft. The integration values are written to area.out and a plot is made showing the integration limits and values overlayed on the spectra to plot.png.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python
# Example script to show integration of a 1D spectrum

import nmrglue as ng
import numpy as np
import matplotlib.pyplot as plt

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("1d_data.ft")
length = data.shape[0]
```
# read in the integration limits
peak_list = np.recfromtxt("limits.in")

# determine the ppm scale
uc = ng.pipe.make_uc(dic, data)
ppm_scale = uc.ppm_scale()

# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(ppm_scale, data, 'k-')

# prepare the output file
f = open("area.out", 'w')
f.write("#Name	Start	Stop	Area
")

# loop over the integration limits
for name, start, end in peak_list:
    min = uc(start, "ppm")
    max = uc(end, "ppm")
    if min > max:
        min, max = max, min

    # extract the peak
    peak = data[min:max + 1]
    peak_scale = ppm_scale[min:max + 1]

    # plot the integration lines, limits and name of peaks
    ax.plot(peak_scale, peak.cumsum() / 100. + peak.max(), 'g-')
    ax.plot(peak_scale, [0] * len(peak_scale), 'r-')
    ax.text(peak_scale[0], 0.5 * peak.sum() / 100. + peak.max(), name,
            fontsize=8)

    # write out the integration info
    tup = (name, peak_scale[0], peak_scale[-1], peak.sum())
    f.write("%s%.3f%.3f%.3f
" % tup)

# close the output file and save the plot
f.close()
ax.set_xlim(ppm_scale[0], ppm_scale[-1])
fig.savefig("plot.png")

[input file]

#Peak Start  Stop
C01 183.40 178.97
C02 178.97 175.33
# Now some more
Ca 65.77 49.46
Cb1 49.46 43.75
Cb2 43.75 39.00
Cg1 37.73 33.86
Cg2 33.86 32.00
Cd1 32.00 29.62
Cd2 29.62 26.98
Ce 26.98 12.10
Results:

[output file]

<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>Stop</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO1</td>
<td>183.395</td>
<td>178.976</td>
<td>2.884854E+08</td>
</tr>
<tr>
<td>CO2</td>
<td>178.976</td>
<td>175.333</td>
<td>1.205766E+08</td>
</tr>
<tr>
<td>Ca</td>
<td>65.774</td>
<td>49.457</td>
<td>4.906673E+08</td>
</tr>
<tr>
<td>Cb1</td>
<td>49.457</td>
<td>43.750</td>
<td>3.062952E+08</td>
</tr>
<tr>
<td>Cb2</td>
<td>43.750</td>
<td>38.991</td>
<td>2.336557E+08</td>
</tr>
<tr>
<td>Cg1</td>
<td>37.729</td>
<td>33.868</td>
<td>3.073099E+08</td>
</tr>
<tr>
<td>Cg2</td>
<td>33.868</td>
<td>31.998</td>
<td>1.470495E+08</td>
</tr>
<tr>
<td>Cd1</td>
<td>31.998</td>
<td>29.618</td>
<td>4.963560E+08</td>
</tr>
<tr>
<td>Cd2</td>
<td>29.618</td>
<td>26.972</td>
<td>3.168956E+08</td>
</tr>
<tr>
<td>Ce</td>
<td>26.972</td>
<td>12.111</td>
<td>2.024605E+08</td>
</tr>
</tbody>
</table>

[figure]

4.5.2 integration example: integrate_2d

This example shows how to use nmrglue to integrate a 2D NMRPipe spectra. The script reads in point limits from limits.in and takes a simple summation integral of all points in each box described. The integrated volumes are writting to volumes.out. For a method to graphically examine these limits see plotting example: plot_2d_boxes. Similarly to check the peak assignments see plotting example: plot_2d_assignments.
The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python
# Example script to show integration of a 2D spectrum

import nmrglue as ng
import numpy as np

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_2d/test.ft2")

# read in the integration limits
peak_list = np.recfromtxt("limits.in")

# prepare the output file
f = open("volumes.out","w")
f.write("# Name	Volume
")

# loop over the integration limits
for name, x0, y0, x1, y1 in peak_list:
    if x0 > x1:
        x0, x1 = x1, x0
    if y0 > y1:
        y0, y1 = y1, y0

    vol = data[y0:y1+1, x0:x1+1].sum()
    f.write("%s	%.3f
"%(name, vol))

# close the output file
f.close()
```

[input file]

```plaintext
#Peak X0 Y0 X1 Y1
# Peak defines 15N resonance in 2D NCO spectra.
# Limits are in term of points aom 0 to length-1.
# These can determined from nmrDraw by subtracting 1 from the X and Y
# values reported.
#Peak X0 Y0 X1 Y1
T49 1992 1334 2003 1316
T11 1996 1302 2008 1284
G14 2032 1314 2044 1293
E15 2077 1025 2087 1004
W43 2008 952 2019 933
```

Results:

[output file]

```plaintext
# Name	Volume
T49 10194682.000
T11 8478199.000
G14 9677602.000
E15 9187130.000
W43 8778141.000
```

4.5. Integration Examples 273
4.6 Interactive Examples

4.6.1 interactive example: interactive_1d

This example shows how to use nmrglue and matplotlib to create a simple interactive environment for examining a 1D spectrum from a NMRPipe file.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python
# Create a 1D interactive plot from NMRPipe data

import nmrglue as ng
import matplotlib.pyplot as plt

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_1d/test.ft")

# create a unit conversion object
uc = ng.pipe.make_uc(dic, data)

# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(uc.ppm_scale(), data, 'k-')

# decorate axes
ax.set_yticklabels([])
ax.set_title("Protein 1D Spectrum")
ax.set_xlabel("13C ppm")
ax.set_xlim(200, 0)
ax.set_ylim(-80000, 2500000)

# start interactive session, script ends when windows closes
plt.show()
```

4.6.2 interactive example: interactive_2d

This example shows how to use nmrglue and matplotlib to create a simple interactive environment for examining a 2D spectrum from a NMRPipe file.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python
# Create a interactive contour plot of a 2D NMRPipe spectrum

import nmrglue as ng
import matplotlib.pyplot as plt
import matplotlib.cm

# plot parameters
cmap = matplotlib.cm.Blues_r  # contour map (colors to use for contours)
contour_start = 30000         # contour level start value
```
contour_num = 20  # number of contour levels
contour_factor = 1.20  # scaling factor between contour levels

# calculate contour levels
cl = [contour_start * contour_factor ** x for x in range(contour_num)]

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_2d/test.ft2")

# make ppm scales
uc_13c = ng.pipe.make_uc(dic, data, dim=1)
ppm_13c_0, ppm_13c_1 = uc_13c.ppm_limits()
uc_15n = ng.pipe.make_uc(dic, data, dim=0)
ppm_15n_0, ppm_15n_1 = uc_15n.ppm_limits()

# create the figure
fig = plt.figure()
ax = fig.add_subplot(111)

# plot the contours
ax.contour(data, cl, cmap=cmap,
            extent=(ppm_13c_0, ppm_13c_1, ppm_15n_0, ppm_15n_1))

# decorate the axes
ax.set_ylabel("15N (ppm)"
ax.set_xlabel("13C (ppm)"
ax.set_title("Protein 2D NCa Spectrum")
ax.set_xlim(70, 40)
ax.set_ylim(135, 100)

# start interactive session, script ends when window is closed
plt.show()

4.7 Plotting Examples

4.7.1 plotting example: plot_1d_time

This example shows how to use nmrglue and matplotlib to create figures for examining data or publication. In this example the fid from a 1D NMRPipe file is plotted.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python
# Create a 1D plot of NMRPipe data

import nmrglue as ng
import matplotlib.pyplot as plt
import numpy as np

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_1d/test.fid")

# make a unit conversion object
uc = ng.pipe.make_uc(dic, data)
```
# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(uc.ms_scale(), data, 'k-')

# decorate axes
ax.set_yticklabels([])
ax.set_title("Protein 1D FID")
ax.set_xlabel("Time (ms)")
ax.set_ylim(-100000, 100000)

# save the figure
fig.savefig("fid.png") # this can be to .pdf, .ps, etc

Results:

[figure]

4.7.2 plotting example: plot_1d_freq

This example shows how to use nmrglue and matplotlib to create figures for examining data or publication. In this example the spectrum from a 1D NMRPipe file is plotted.

The data used in this example is available for download.
```python
#!/usr/bin/env python
# Create a 1D plot of NMRPipe data

import nmrglue as ng
import matplotlib.pyplot as plt
import numpy as np

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_1d/test.ft")

# create a unit conversion object for the axis
uc = ng.pipe.make_uc(dic, data)

# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(uc.ppm_scale(), data, 'k-')

# annotate the figure
ax.annotate('CO region', xy=(173, 2.15e6), xycoords='data', xytext=(30,20),
textcoords='offset points', arrowprops=dict(arrowstyle="->"))
ax.text(59, 1.55e6, "alphatic region")
ax.annotate('', xy=(70,1.2e6), xycoords='data', xytext=(10, 1.2e6),
textcoords='data',
arrowprops=dict(arrowstyle="<->", connectionstyle="bar", ec="k", shrinkA=5, shrinkB=5,))

# decorate axes
ax.set_yticklabels([])
ax.set_title("Protein 1D Spectrum")
ax.set_xlabel("13C ppm")
ax.set_xlim(200, 0)
ax.set_ylim(-80000, 2500000)

# save the figure
fig.savefig("spectrum.png") # this can be .pdf, .ps, etc
```

Results:

[figure]
4.7.3 plotting example: plot_2d_spectrum

This example shows how to use nmrglue and matplotlib to create figures for examining data or publication. In this example a contour plot of the spectrum from a 2D NMRPipe file is created. Slices are added in the 15N and 13C dimension as well as sample peak labels. *plotting example: plot_2d_spectrum_pts* is similar to this example but plotted on a point scale.

The data used in this example is available for [download](#).

[source code]

```python
#!/usr/bin/env python
# Create a contour plot of a 2D NMRPipe spectrum

import nmrglue as ng
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm

# plot parameters

# contour map (colors to use for contours)
cmap = matplotlib.cm.Blues_r
contour_start = 30000  # contour level start value
contour_num = 20       # number of contour levels
contour_factor = 1.20  # scaling factor between contour levels
```
```python
# calculate contour levels
cl = contour_start * contour_factor ** np.arange(contour_num)

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_2d/test.ft2")

# make ppm scales
uc_13c = ng.pipe.make_uc(dic, data, dim=1)
ppm_13c = uc_13c.ppm_scale()
ppm_13c_0, ppm_13c_1 = uc_13c.ppm_limits()
uc_15n = ng.pipe.make_uc(dic, data, dim=0)
ppm_15n = uc_15n.ppm_scale()
ppm_15n_0, ppm_15n_1 = uc_15n.ppm_limits()

# create the figure
fig = plt.figure()
ax = fig.add_subplot(111)

# plot the contours
ax.contour(data, cl, cmap=cmap,
            extent=(ppm_13c_0, ppm_13c_1, ppm_15n_0, ppm_15n_1))

# add some labels
ax.text(59.25, 104.0, "T49", size=8, color='r')
ax.text(58.75, 106, "T11", size=8, color='k')

# plot slices in each direction
xslice = data[uc_15n("111.27 ppm"), :]
ax.plot(ppm_13c, -xslice / 4.e4 + 111.27)
yslice = data[:, uc_13c("62.0 ppm")]
ax.plot(yslice / 2.e4 + 62.0, ppm_15n)

# decorate the axes
ax.set_ylabel("15N (ppm)")
ax.set_xlabel("13C (ppm)")
ax.set_title("Protein 2D NCa Spectrum")
ax.set_xlim(70, 40)
ax.set_ylim(135, 100)

# save the figure
fig.savefig("spectrum.png") # this can be .pdf, .ps, etc
```

Result:

[spectrum.png]
4.7.4 plotting example: plot_2d_spectrum_pts

This example shows how to use nmrglue and matplotlib to create figures for examining data or publication. In this example a contour plot of the spectrum from a 2D NMRPipe file is created. Slices are added in the 15N and 13C dimension as well as sample peak labels. *plotting example: plot_2d_spectrum* is similar to this example but plotted on a ppm scale.

[source code]

```python
#!/usr/bin/env python
# Create a contour plot of a 2D NMRPipe spectrum
import nmrglue as ng
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm

# plot parameters

cmap = matplotlib.cm.Blues_r      # contour map (colors to use for contours)
contour_start = 30000           # contour level start value
contour_num = 20                  # number of contour levels
contour_factor = 1.20          # scaling factor between contour levels

# calculate contour levels
```
```python
cl = contour_start * contour_factor ** np.arange(contour_num)

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_2d/test.ft2")

# create the figure
fig = plt.figure()
ax = fig.add_subplot(111)

# plot the contours
ax.contour(data, cl, cmap=cmap,
            extent=(0, data.shape[1] - 1, 0, data.shape[0] - 1))

# add some labels
ax.text(2006, 1322, "T49", size=8, color='r')
ax.text(2010, 1290, "T11", size=8, color='k')

# plot slices in each direction
xslice = data[1187, :]
ax.plot(xrange(data.shape[1]), xslice / 3.e3 + 1187)
yslice = data[:, 1976]
ax.plot(-yslice / 3.e3 + 1976, xrange(data.shape[0]))

# decorate the axes
ax.set_ylabel("15N (points)")
ax.set_xlabel("13C (points)")
ax.set_title("Protein 2D NCA Spectrum")
ax.set_xlim(1900, 2200)
ax.set_ylim(750, 1400)

# save the figure
fig.savefig("spectrum_pts.png") # this can be .pdf, .ps, etc
```

Result:
```
[spectrum.png]
```
4.7.5 plotting example: plot_2d_boxes

This example shows how to use nmrglue and matplotlib to create figures for examining data or publication. In this example the box limits used in integration example: integrate_2d are graphically examined. A contour plot of each peak is plotted with the box limits indicated by the dark dashed line. To check peak assignments see plotting example: plot_2d_assignments.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python
# Create a contour plots of each peak defined in limits.in file

import nmrglue as ng
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm

# plot parameters
xpad = 5  # padding around peak box on x-axis
ypad = 5  # padding around peak box on y-axis
cmap = matplotlib.cm.Blues_r  # contour map (colors to use for contours)
contour_start = 30000  # contour level start value
contour_num = 20  # number of contour levels
```
contour_factor = 1.20  # scaling factor between contour levels

# calculate contour levels
cl = contour_start * contour_factor ** np.arange(contour_num)

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_2d/test.ft2")

# read in the integration limits
peak_list = np.recfromtxt("limits.in")

# loop over the peaks
for name, x0, y0, x1, y1 in peak_list:
  if x0 > x1:
    x0, x1 = x1, x0
  if y0 > y1:
    y0, y1 = y1, y0

  # slice the data around the peak
  slice = data[y0 - ypad:y1 + 1 + ypad, x0 - xpad:x1 + 1 + xpad]

  # create the figure
  fig = plt.figure()
  ax = fig.add_subplot(111)

  # plot the contours
  print "Plotting: ", name
  etup = (x0 - xpad + 1, x1 + xpad - 1, y0 - ypad + 1, y1 + ypad - 1)
  ax.contour(slice, cl, cmap=cmap, extent=etup)

  # draw a box around the peak
  ax.plot([x0, x1, x1, x0, x0], [y0, y0, y1, y1, y0], 'k--')

  # draw light boxes at +/- one point
  ax.plot([x0 - 1, x1 + 1, x1 + 1, x0 - 1, x0 - 1],
          [y0 - 1, y0 - 1, y1 + 1, y1 + 1, y0 - 1], 'k--', alpha=0.35)
  ax.plot([x0 + 1, x1 - 1, x1 - 1, x0 + 1, x0 + 1],
          [y0 + 1, y0 + 1, y1 - 1, y1 - 1, y0 + 1], 'k--', alpha=0.35)

  # set the title
  ax.set_title(name)

  # save the figure
  fig.savefig(name + ".png")
  del(fig)

[input file]

# Peak X0 Y0 X1 Y1
# Peak defines 15N resonance in 2D NCO spectra.
# Limits are in terms of points from 0 to length-1.
# These can determined from nmrDraw by subtracting 1 from the X and Y
# values reported.
# Peak X0 Y0 X1 Y1
T49 1992 1334 2003 1316
T11 1996 1302 2008 1284
# comments can appear anywhere in this file just start the line with #
4.7.6 plotting example: plot_2d_assignments

This example shows how to use nmrglue and matplotlib to create figures for examining data or publication. In this example the assignments used in integration example: integrate_2d are graphically examined. A contour plot of the spectrum with the boxes and assignments is created. To examine the box limit more closely see plotting example: plot_2d_boxes.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python
# Create a contour plots of a spectrum with each peak in limits.in labeled

import nmrglue as ng
import numpy as np
import matplotlib.pyplot as plt
```

[T11.png]
```python
import matplotlib.cm

# plot parameters

cmap = matplotlib.cm.Blues_r  # contour map (colors to use for contours)
contour_start = 30000          # contour level start value
contour_num = 20               # number of contour levels
contour_factor = 1.20          # scaling factor between contour levels
textsize = 6                  # text size of labels

# calculate contour levels
cl = contour_start * contour_factor ** np.arange(contour_num)

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("nmrpipe_2d/test.ft2")

# read in the integration limits
peak_list = np.recfromtxt("limits.in")

# create the figure
fig = plt.figure()
ax = fig.add_subplot(111)

# plot the contours
ax.contour(data, cl, cmap=cmap,
            extent=(0, data.shape[1] - 1, 0, data.shape[0] - 1))

# loop over the peaks
for name, x0, y0, x1, y1 in peak_list:
    if x0 > x1:
        x0, x1 = x1, x0
    if y0 > y1:
        y0, y1 = y1, y0

    # plot a box around each peak and label
    ax.plot([x0, x1, x1, x0, x0], [y0, y0, y1, y1, y0], 'k')
    ax.text(x1 + 1, y0, name, size=textsize, color='r')

# set limits
ax.set_xlim(1900, 2200)
ax.set_ylim(750, 1400)

# save the figure
fig.savefig("assignments.png")
```

[input file]
```
#Peak  X0     Y0     X1     Y1
# Peak defines 15N resonance in 2D NCO spectra.
# Limits are in term of points from 0 to length-1.
# These can determined from nmrDraw by subtracting 1 from the X and Y
# values reported.

#Peak  X0     Y0     X1     Y1
T49    1992    1334    2003    1316
T11    1996    1302    2008    1284

# comments can appear anywhere in this file just start the line with #
G14    2032    1314    2044    1293
E15    2077    1025    2087    1004
```

4.7. Plotting Examples
4.8 Processing Examples

4.8.1 Processing 1D Bruker Data

This example shows how nmrglue can be used to process and display one dimensional Bruker data.

Raw Bruker data from modern spectrometers contains a group delay artifact which must be removed during processing. There has been much speculation as to the origins of this artifact and many methods for removing the artifact have been suggested [1], [2], [3], [4], [5].

Nmrglue provides an algorithm for removing this artifact based on the protocol presented in “DMX DIGITAL FILTERS AND NON-BRUKER OFFLINE PROCESSING III” by W. M. Westler and F. Abildgaard. This method is available for use through `nmrglue.fileio.bruker.remove_digital_filter()`. Nmrglue users can use this included function to remove the artifact or implement their own method if they are unsatisfied with the results.

In this example a 1D NMR spectrum of 1,3 diaminopropane is processed and plotted using nmrglue. The results can be compared with the spectrum produced from NMRPipe which provides a different artifact removal algorithm. Note
that no apodization or baseline corrections are performed on these spectra.

**Instructions**

- Download the 1D proton spectrum of 1,3 diaminopropane and unpack in this directory. This raw data is available from the Madison Metabolomics Consortium Database as expnmr_00001_1.tar.
- Execute `process_and_plot_nmrglue.py` to process and plot the 1D spectrum. This creates the file `figure_nmrglue.png`.
- Optionally, the data can be processed with NMRPipe using the script `nmrpipe_proc.com`. Then `plot_nmrpipe.py` can be used to plot the resulting spectrum. This creates the file `figure_nmrpipe.png`.

```python
#!/usr/bin/env python

import nmrglue as ng
import matplotlib.pyplot as plt

# read in the bruker formatted data
dic, data = ng.bruker.read('expnmr_00001_1')

# remove the digital filter
data = ng.bruker.remove_digital_filter(dic, data)

# process the spectrum
data = ng.proc_base.zf_size(data, 32768)  # zero fill to 32768 points
data = ng.proc_base.fft(data)            # Fourier transform
data = ng.proc_base.ps(data, p0=-50.0)   # phase correction
data = ng.proc_base.di(data)             # discard the imaginaries
data = ng.proc_base.rev(data)            # reverse the data

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(data[20000:25000])
fig.savefig('figure_nmrglue.png')
```

Output:

[figure_nmrglue.png]
```bash
#!/bin/csh

bruk2pipe -in ./expnmr_00001_1/fid \
  -bad 0.0 -noaswap -DMX -decim 32 -dspfv 12 -grpdly 0 \
  -xN 32768 \n  -xT 16384 \n  -xMODE DQD \n  -xSW 4807.692 \n  -xOBS 400.132 \n  -xCAR 4.697 \n  -xLAB 1H \n  -ndim 1 \n  -out ./test.fid -verb -ov

nmrPipe -in test.fid \
  | nmrPipe -fn ZF -auto \
  | nmrPipe -fn FT \
  | nmrPipe -fn PS -p0 -22.0 -p1 0.0 -di \
  -out test.ft2 -verb -ov

plot_nmrpipe.py
```
import nmrglue as ng
import matplotlib.pyplot as plt

# read in the data
dic, data = ng.pipe.read('test.ft2')

# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
av.plot(data[20000:25000])
fig.savefig('figure_nmrpipe.png')

Output:

[figure_nmrpipe.png]

4.8.2 process example: process_pipe_1d

This example shows how nmrglue can be used to process NMR data. In this script a 1D time domain NMRPipe file is processing into a 1D NMRPipe frequency domain file.

The data used in this example is available for download.

[source code]
import nmrglue as ng

dic, data = ng.pipe.read("nmrpipe_1d/test.fid")

dic, data = ng.pipe_proc.sp(dic, data, off=0.35, end=0.98, pow=2, c=1.0)
dic, data = ng.pipe_proc.zf(dic, data, auto=True)
dic, data = ng.pipe_proc.ft(dic, data, auto=True)
dic, data = ng.pipe_proc.ps(dic, data, p0=-17.7, p1=-36.0)
dic, data = ng.pipe_proc.di(dic, data)

dic, data = ng.pipe.write("1d_pipe.ft", dic, data, overwrite=True)

4.8.3 process example: process_pipe_2d

This example shows how nmrglue can be used to process NMR data. In this script a 2D time domain NMRPipe file is processing into a 2D NMRPipe frequency domain file.

The data used in this example is available for download.

[source code]
4.8.4 process example: process_pipe_2d_tppi

This example shows how nmrglue can be used to process NMR data. In this script a 2D time domain NMRPipe file is processing into a 2D NMRPipe frequency domain file.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# read in the file
dic, data = ng.pipe.read("nmrpipe_2d_tppi/test.fid")

# process the direct dimension
dic, data = ng.pipe_proc.sp(dic, data, off=0.35, end=0.98, pow=2, c=1.0)
dic, data = ng.pipe_proc.zf(dic, data, auto=True)
dic, data = ng.pipe_proc.ft(dic, data, auto=True)
dic, data = ng.pipe_proc.ps(dic, data, p0=151.0, p1=0.0)
dic, data = ng.pipe_proc.di(dic, data)

# process the indirect dimension
dic, data = ng.pipe_proc.tp(dic, data)
dic, data = ng.pipe_proc.sp(dic, data, off=0.35, end=0.98, pow=2, c=0.5)
dic, data = ng.pipe_proc.zf(dic, data, auto=True)
dic, data = ng.pipe_proc.ft(dic, data, auto=True)
dic, data = ng.pipe_proc.ps(dic, data, p0=0.0, p1=0.0)
dic, data = ng.pipe_proc.di(dic, data)
dic, data = ng.pipe_proc.rev(dic, data, sw=True)
dic, data = ng.pipe_proc.tp(dic, data)

# write out processed data
ng.pipe.write("2d_pipe_tppi.ft2", dic, data, overwrite=True)
```

4.8.5 process example: process_pipe_3d

This example shows how nmrglue can be used to process NMR data. In this script a 3D time domain NMRPipe file is processing into a 3D NMRPipe frequency domain file. For 3D processing the iter3D object is used to loop over XY and ZX planes. Detail on this object can be found in the `nmrglue.varian` documentation.

The data used in this example is available for download.

[source code]

```python
#!/usr/bin/env python

import nmrglue as ng

# create the iterator (xiter) which will loop over YX planes
xiter = ng.pipe.iter3D("nmrpipe_3d/data/test%03d.fid", 'x', 'x')

print "Processing XY planes..."
for i, (dic, plane) in enumerate(xiter):

    # process the direct dimention (x)
    dic, plane = ng.pipe_proc.zf(dic, plane, auto=True)
```

4.8. Processing Examples
dic, plane = ng.pipe_proc.ft(dic, plane, auto=True)
dic, plane = ng.pipe_proc.ps(dic, plane, p0=0.0, p1=0.0)
dic, plane = ng.pipe_proc.di(dic, plane)

def process_first_indirect_y(dic, plane):
    # process the first indirect (y) dimension
    dic, plane = ng.pipe_proc.tp(dic, plane)
dic, plane = ng.pipe_proc.zf(dic, plane, auto=True)
dic, plane = ng.pipe_proc.ft(dic, plane, auto=True)
dic, plane = ng.pipe_proc.ps(dic, plane, p0=-92.0, p1=65.0)
dic, plane = ng.pipe_proc.di(dic, plane)

    # write out the plane
    xiter.write("./ft/test%03d.ft2", plane, dic)

print "Processing ZY planes..."

for i, (dic, plane) in enumerate(ziter):
    # process the z-dim
    dic, plane = ng.pipe_proc.zf(dic, plane, auto=True)
dic, plane = ng.pipe_proc.ft(dic, plane, auto=True)
dic, plane = ng.pipe_proc.ps(dic, plane, p0=-92.0, p1=65.0)
dic, plane = ng.pipe_proc.di(dic, plane)

    # write out the plane
    ziter.write("./ft/test%03d.ft3", plane, dic)

4.9 Separate Examples

4.9.1 separate example: separate_1d_varian

This example shows how to use nmrglue to separate Agilent/Varian data collected with an innermost parameter interleaved. The full experimental data in the arrayed_data.dir directory is unpacked into a series of directories with names tHX_* .fid which can be converted with nmrglue or NMRPipe. The name and values of the interleaved parameter is determined from the procpar file in the arrayed_data.dir directory.

The data used in this example is available for download.
[source code]

```python
#! /usr/bin/env python
# Separate 1D Agilent/Varian data creating directories based on the
# array parameter found in the procpar file.
import nmrglue as ng

dic, data = ng.varian.read('arrayed_data.dir')
```
4.9.2 separate example: separate_2d_varian

This example shows how to use nmrglue to separate Agilent/Varian data collected with an innermost parameter interleaved. The full experimental data in the `arrayed_data.dir` directory is unpacked into a series of directories with names `tXmix_* . fid` which can be converted with nmrglue or NMRPipe. The name and values of the interleaved parameter is determined from the `procpar` file in the `arrayed_data.dir` directory.

The data used in this example is available for download.
[source code]

```python
#!/usr/bin/env python
# Unarray 2D Agilent/Varian data creating directories based on array parameter
# found in procpar file.

import nmrglue as ng

# read in the NMR data
dic, data = ng.varian.read('arrayed_data.dir')

# set the new size of the separated data
dic['nblocks'] = data.shape[0]

arrayed_param = dic['procpar']['array']['values'][0]

# loop over the echo times, separating and saving each 1D
for i, array_val in enumerate(dic['procpar'][arrayed_param]['values']):
    dir_name = arrayed_param + '_' + array_val + '.fid'
    print "Creating directory: ", dir_name
    ng.varian.write(dir_name, dic, data[:, i, :], overwrite=True)
```

4.9.3 separate example: separate_2d_bruker

This example shows how to use nmrglue to separate Bruker data collected with an innermost parameter interleaved. The full experimental data in the `arrayed_data.dir` directory is unpacked into a series of directories named `1, 2, 3, ... 23` which can be converted with nmrglue or NMRPipe. The data shape, array size and additional files to copy to the new directories must be determined by the user.

The data used in this example is available for download.
[source code]

```python
#!/usr/bin/env python
# Separate 2D data sets from an arrayed data set, directories will be names
# 1, 2, 3, ... 23
```
import nmrglue as ng

def read_in_the_NMR_data:
    dic, data = ng.bruker.read('arrayed_data.dir', shape=(7360, 640), cplex=True)

array_size = 23

for i in range(array_size):
    dir_name = str(i+1)
    print "Creating directory:", dir_name
    ng.bruker.write(dir_name, dic, data[i::array_size], overwrite=True)

4.10 Simulation Examples

4.10.1 Simulate a ucsf (Sparky) file

This example shows how to use the nmrglue.analysis.linesh.sim_NDregion() function to simulate a HSQC spectrum with peak locations provided by a text file. The simulated spectrum is saved as a ucsf (Sparky) file named test.ucsf.

[make_ucsf.py]

```python
#!/usr/bin/env python

import nmrglue as ng
import numpy as np

# create a sparky dictionary
# A dictionary from a existing Sparky ucsf file can be found using:
# ng.sparky.guess_udic(*ng.sparky.read('filename.ucsf'))
udic = {
    'ndim': 2,
    0: {'car': 7329.0,
        'complex': False,
        'encoding': 'states',
        'freq': True,
        'label': '15N',
        'obs': 60.8,
        'size': 512,
        'sw': 1523.43,
        'time': False},
    1: {'car': 5403.570418865944,
        'complex': False,
        'encoding': 'direct',
        'freq': True,
        'label': '1H',
        'obs': 600.0,
        'size': 1024,
        'sw': 3606.5,
        'time': False}
}

dic = ng.sparky.create_dic(udic)
```
```python
data = np.empty((512, 1024), dtype='float32')

# read in the peak list
peak_list = np.recfromtxt('peaks.txt', names=True)
npeaks = len(peak_list)

# convert the peak list from PPM to points
uc_15N = ng.sparky.make_uc(dic, None, 0)
uc_1H = ng.sparky.make_uc(dic, None, 1)

lw_15N = 5.0  # 15N dimension linewidth in points
lw_1H = 5.0   # 1H dimension linewidth in points

params = []
for ppm_15N, ppm_1H in peak_list:
    pts_15N = uc_15N.f(ppm_15N, 'ppm')
    pts_1H = uc_1H.f(ppm_1H, 'ppm')
    params.append([(pts_15N, lw_15N), (pts_1H, lw_1H)])

# simulate the spectrum
shape = (512, 1024)  # size should match the dictionary size
lineshapes = ('g', 'g')  # gaussian in both dimensions
amps = [100.0] * npeaks
data = ng.linesh.sim_NDregion(shape, lineshapes, params, amps)

# save the spectrum
ng.sparky.write("test.ucsf", dic, data.astype('float32'), overwrite=True)
```

```
[peaks.txt]

#15N  1H
112.7  7.45
117.8  8.57
123.0  9.02
```
These are all the examples from the Journal of Biomolecular NMR article on nmrglue. The full reference for this article is:

5.1 Sparky to NMRPipe example

5.1.1 Introduction

This example is taken from Listing S1 in the 2013 JBNMR nmrglue paper. In this example a 2D Sparky file (data.ucsf) is converted to a NMRPipe file (data.ft2) by the Python script sparky2pipe_convert.py.

5.1.2 Instructions

Execute python sparky2pipe_convert.py to perform the conversion. The file data.ft2 will be created.

The data used in this example is available for download.

Listing S1

```
[sparky2pipe_convert.py]

import nmrglue as ng

# read in the Sparky file
sdic, sdata = ng.sparky.read('data.ucsf')

# convert to NMRPipe format
C = ng.convert.converter()
C.from_sparky(sdic, sdata)
pdic, pdata = C.to_pipe()
```
5.2 1D visualization examples

5.2.1 Introduction

These two examples are taken from Listing S2 and S3 in the 2013 JBNMR nmrglue paper. In the first example a 1D time domain signal from a NMRPipe file (test.fid) is visualized using the plot_1d_pipe_time.py script. In the second example a portion of a 1D 13C CP MAS NMR spectrum is visualized from the NMRPipe file test.ft using the script plot_1d_pipe_freq.py

5.2.2 Instructions

Execute `python plot_1d_pipe_time.py` to visualize the data in the file `test.fid`. The resulting file `fid.png` is presented as Figure 2 in the paper.

Execute `python plot_1d_pipe_spectrum.py` to visualize the data in the file `test.ft`. The resulting file `spectrum.png` is presented as Figure 3 in the paper.

The data used in this example is available for download.

Listing S2

```python
import nmrglue as ng
import matplotlib.pyplot as plt

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("test.fid")

# make a unit conversion object for the axis
uc = ng.pipe.make_uc(dic, data)

# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(uc.ms_scale(), data.real, 'k-')

# decorate axes
ax.set_yticklabels([])
ax.set_xlabel("Time (ms)")
ax.set_ylim(-100000, 100000)

# save the figure
fig.savefig("fid.png")
```
Listing S3
[plot_1d_pipe_spectrum.py]

```python
import nmrglue as ng
import matplotlib.pyplot as plt

# read in the data from a NMRPipe file
dic, data = ng.pipe.read("test.ft")

# create a unit conversion object for the axis
uc = ng.pipe.make_uc(dic, data)

# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(uc.ppm_scale(), data, 'k-')

# decorate axes
ax.set_yticklabels([])
ax.set_xlabel("13C ppm")
ax.set_xlim(200, 0)
ax.set_ylim(-80000, 250000)

# save the figure
fig.savefig("spectrum.png")
```

5.2. 1D visualization examples
5.3 2D visualization example

5.3.1 Introduction

This example is taken from Listing S4 from the 2013 JBNMR nmrglue paper. In this example a 2D SSNMR spectrum is visualized using the script `plot_2d_pipe_spectrum.py`.

5.3.2 Instructions

Execute `python plot_2d_pipe_spectrum.py` to visualize the data in the file `test.ft`. The resulting file `spectrum_2d.png` is presented as Figure 4 in the paper.

The data used in this example is available for download.

Listing S4

[plot_2d_pipe_spectrum.py]

```python
import nmrglue as ng
import matplotlib.pyplot as plt

# read in data
```

![2D SSNMR spectrum](image)
dic, data = ng.pipe.read("test.ft2")

# find PPM limits along each axis
uc_15n = ng.pipe.make_uc(dic, data, 0)
uc_13c = ng.pipe.make_uc(dic, data, 1)
x0, x1 = uc_13c.ppm_limits()
y0, y1 = uc_15n.ppm_limits()

# plot the spectrum
fig = plt.figure(figsize=(10, 10))
fig = plt.figure()
ax = fig.add_subplot(111)
cl = [8.5e4 * 1.30 ** x for x in range(20)]
ax.contour(data, cl, colors='blue', extent=(x0, x1, y0, y1), linewidths=0.5)

# add 1D slices
x = uc_13c.ppm_scale()
s1 = data[uc_15n("105.52ppm"), :]
s2 = data[uc_15n("115.85ppm"), :]
s3 = data[uc_15n("130.07ppm"), :]
ax.plot(x, -s1 / 8e4 + 105.52, 'k-')
ax.plot(x, -s2 / 8e4 + 115.85, 'k-')
ax.plot(x, -s3 / 8e4 + 130.07, 'k-')

# label the axis and save
ax.set_xlabel("13C ppm", size=20)
ax.set_xlim(183.5, 167.5)
ax.set_ylabel("15N ppm", size=20)
ax.set_ylim(139.5, 95.5)
fig.savefig("spectrum_2d.png")
5.4 Separated interleaved example

5.4.1 Introduction

This example is taken from Listing S5 in the 2013 JBNMR nmrglue paper. In this example a pseudo-3D NMRPipe data set is separated into 2D data sets using the script separate.py.

5.4.2 Instructions

Execute `python separate.py` to separate the pseudo-3D data set into 2D data sets. Six directories with names `techo_XXXX.fid` will be created.

The data used in this example is available for download.

Listing S5

```python
import nmrglue as ng

# read in the NMR data
dic, data = ng.varian.read('arrayed_data.fid')
```
5.5 Seperated inner phase example

5.5.1 Introduction

This example is taken from Listing S6 in the 2013 JBNMR nmrglue paper. In this example a pseudo-3D NMRPipe data set is seperated into 2D data sets using the script `seperate.py` where there is an innermost quadrature phase loop.

5.5.2 Instructions

Execute `python seperate.py` to seperate the pseudo-3D data set into 2D data sets. Ten directories with names `nredor_XX.fid` will be created.

The data used in this example is available for download.

Listing S6

```python
import nmrglue as ng
import numpy as np

# read the NMR data, forcing the data to be two dimensional
dic, data = ng.varian.read('arrayed_data.fid', as_2d=True)

# set the new size of the separated data
array_size = len(dic['procpar']['nredor']['values'])
out_shape = int(data.shape[0] / array_size), data.shape[1]
dic['nblocks'] = out_shape[0]

# loop over the redor multiples, separating and saving each 2D
for i, nredor in enumerate(dic['procpar']['nredor']['values']):
    dir_name = 'nredor_' + nredor + '.fid'
    print "Creating directory:", dir_name
    sdata = np.empty(out_shape, dtype=data.dtype)
    sdata[::2] = data[2 * i::2 * array_size]
    sdata[1::2] = data[2 * i + 1::2 * array_size]
    ng.varian.write(dir_name, dic, sdata, overwrite=True)
```
5.6 Processing S3E filtered data example

5.6.1 Introduction

This example is taken from Listing S7, S8 and S9 in the 2013 JBNMR nmrglue paper. In this example a 2D Agilent/Varian data set collect using a S3E filter is separated (seperate_s3e.py), converted to NMRPipe format (Sparky file ('data.ucsf')) is converted to a NMRPipe file ('convert.py') and finally processed (xy_s3e.py).

5.6.2 Instructions

Execute `python seperate_s3e.py` to separate the S3E sum and difference spectra from data set in the Agilent/Varian `fid` file. This creates the files `fid_dif` and `fid_sum`.

Execute `python convert.py` to convert these two files to NMRPipe format. This step creates the files `test_sum.fid` and `test_dif.fid`.

Execute `python xy_s3e.py` to process and combine the sum and different spectra. This step creates the `test.ft2` file.

The data used in this example is available for download.

Listing S7

```
import nmrglue as ng

dic, data = ng.varian.read('.', as_2d=True)
dic['nblocks'] /= 2
A = data[::2]
B = data[1::2]
ng.varian.write_fid('fid_sum', dic, A + B, overwrite=True)
ng.varian.write_fid('fid_dif', dic, A - B, overwrite=True)
```

Listing S8

```
import nmrglue as ng

# read in the sum data set
dic, data = ng.varian.read('.', fid_file='fid_sum', as_2d=True)

# set the spectral parameters
udic = ng.varian.guess_udic(dic, data)
udic[1]['size'] = 1500 ; udic[0]['size'] = 256
udic[1]['complex'] = True ; udic[0]['complex'] = True
udic[1]['encoding'] = 'direct' ; udic[0]['encoding'] = 'states'
udic[1]['sw'] = 50000.000 ; udic[0]['sw'] = 5000.0
udic[1]['obs'] = 125.690 ; udic[0]['obs'] = 50.648
udic[1]['car'] = 174.538 * 125.690 ; udic[0]['car'] = 119.727 * 50.648
udic[1]['label'] = 'C13' ; udic[0]['label'] = 'N15'

# convert to NMRPipe format
C = ng.convert.converter()
C.from_varian(dic, data, udic)
pdic, pdata = C.to_pipe()

# write out the NMRPipe file
ng.pipe.write("test_sum.fid", pdic, pdata, overwrite=True)
```
# repeat for the difference data set
dic, data = ng.varian.read('.', fid_file='fid_dif', as_2d=True)
C = ng.convert.converter()
C.from_varian(dic, data, udic)
pdic, pdata = C.to_pipe()
ng.pipe.write("test_dif.fid", pdic, pdata, overwrite=True)

Listing S9

import nmrglue as ng

# process the direct dimension of the sum data set
sdic, sdata = ng.pipe.read('test_sum.fid')
sdic, sdata = ng.pipe_proc.sp(sdic, sdata, off=0.45, end=0.95, pow=1, c=1.0)
sdic, sdata = ng.pipe_proc.zf(sdic, sdata, size=8192)
sdic, sdata = ng.pipe_proc.ft(sdic, sdata)
uc = ng.pipe.make_uc(sdic, sdata, dim=1)
pts = uc.f('27.5 Hz') - uc.f('0 Hz')
sdic, sdata = ng.pipe_proc.fsh(sdic, sdata, dir='ls', pts=pts)
sdic, sdata = ng.pipe_proc.ps(sdic, sdata, p0=-79.0, p1=0.0)
sdic, sdata = ng.pipe_proc.di(sdic, sdata)

# process the direct dimension of the difference data set
ddic, ddata = ng.pipe.read('test_dif.fid')
ddic, ddata = ng.pipe_proc.sp(ddic, ddata, off=0.45, end=0.95, pow=1, c=1.0)
ddic, ddata = ng.pipe_proc.zf(ddic, ddata, size=8192)
ddic, ddata = ng.pipe_proc.ft(ddic, ddata)
uc = ng.pipe.make_uc(ddic, ddata, dim=1)
pts = uc.f('27.5 Hz') - uc.f('0 Hz')
ddic, ddata = ng.pipe_proc.fsh(ddic, ddata, dir='rs', pts=pts)
ddic, ddata = ng.pipe_proc.ps(ddic, ddata, p0=-90.0, pl=0.0)
ddic, ddata = ng.pipe_proc.di(ddic, ddata)

# sum the different and sum data sets
data = sdata + ddata
dic = ddic

# process the indirect dimension
dic, data = ng.pipe_proc.tp(dic, data)
dic, data = ng.pipe_proc.sp(dic, data, off=0.45, end=0.95, pow=1, c=1.0)
dic, data = ng.pipe_proc.zf(dic, data, size=2048)
dic, data = ng.pipe_proc.ft(dic, data, neg=True)
dic, data = ng.pipe_proc.ps(dic, data, p0=0.0, pl=0.0)
dic, data = ng.pipe_proc.di(dic, data)
dic, data = ng.pipe_proc.tp(dic, data)

# write out the results
ng.pipe.write('test.ft2', dic, data, overwrite=True)
5.7 Covariance Processing example

5.7.1 Introduction

This example is taken from Listing S10 in the 2013 JBNMR nmrglue paper. In this example covariance processing is performed on a 2D NMRPipe file.

5.7.2 Instructions

The test.ft file is provided in the archive. To create this from time domain data use the NMRPipe script x.com

Execute python cov_process.py to perform the covariance processing on the test.ft file. The file test.ft2 is created.

Execute python cov_plot.py to perform the covariance processing and plot the results. The output, covariance_figure.png is presented as Figure 5 in the article.

The data used in this example is available for download.

Listing S10

[cov_process.py]

```python
import nmrglue as ng
import numpy as np

# open the data
dic, data = ng.pipe.read("test.ft")

# compute the covariance
C = np.cov(data.T).astype('float32')

# update the spectral parameter of the indirect dimension
dic['FDF1FTFLAG'] = dic['FDF2FTFLAG']
dic['FDF1ORIG'] = dic['FDF2ORIG']
dic['FDF1SW'] = dic['FDF2SW']
dic['FDSPECNUM'] = C.shape[1]

# write out the covariance spectrum
ng.pipe.write("test.ft2", dic, C, overwrite=True)
```

[cov_plot.py]

```python
import nmrglue as ng
import matplotlib.pyplot as plt
import numpy as np

# open the data
dic, data = ng.pipe.read("test.ft")
uc = ng.pipe.make_uc(dic, data, 1)
x0, x1 = uc.ppm_limits()

# compute the covariance
C = np.cov(data.T)

# plot the spectrum
fig = plt.figure()
ax = fig.add_subplot(111)
cl = [1e8 * 1.30 ** x for x in range(20)]
```
5.8 3D strip plot example

5.8.1 Introduction

This example is taken from Listing S11 from the 2013 JBNMR nmrglue paper. In this example a three 3D solid state NMR spectra are visualized as strip plots.

5.8.2 Instructions

Process each of the three 3D spectra by decending into the appopiate directory and executing the *fid.com* followed by the *xy.com* NMRPipe scripts.

Execute `python make_strip_plots.py` to create the strip plots. The resulting file *strip_plots.png* is presented as Figure 6 in the paper.
The data used in this example is available for download part1, part2.

Listing S11

```
[# make strip plots.py]

#!/usr/bin/env python

import nmrglue as ng
import numpy as np
import matplotlib.pyplot as plt

# NMRPipe files of spectra to create strip plots from
spectrum_1 = 'GB3-CN-3DCONCA-041007.fid/ft/test%03d.ft3'
spectrum_2 = 'GB3-CN-3DNACAX-040507.fid/ft/test%03d.ft3'
spectrum_3 = 'GB3-CN-3DNOCX-040807.fid/ft/test%03d.ft3'

# contour parameters
contour_start_s1 = 1.0e5
contour_step_s1 = 1.15

contour_start_s2 = 2.3e5
contour_step_s2 = 1.15

contour_start_s3 = 3.0e5
contour_step_s3 = 1.20

colors_s1 = 'blue'
colors_s2 = 'green'
colors_s3 = 'red'

cl_s1 = contour_start_s1 * contour_step_s1 ** np.arange(20)
cl_s2 = contour_start_s2 * contour_step_s2 ** np.arange(20)
cl_s3 = contour_start_s3 * contour_step_s3 ** np.arange(20)

# open the three data sets
dic_1, data_1 = ng.pipe.read_lowmem(spectrum_1)
dic_2, data_2 = ng.pipe.read_lowmem(spectrum_2)
dic_3, data_3 = ng.pipe.read_lowmem(spectrum_3)

# make unit conversion objects for each axis of each spectrum
uc_s1_a0 = ng.pipe.make_uc(dic_1, data_1, 0) # N
uc_s1_a1 = ng.pipe.make_uc(dic_1, data_1, 1) # CO
uc_s1_a2 = ng.pipe.make_uc(dic_1, data_1, 2) # CA

uc_s2_a0 = ng.pipe.make_uc(dic_2, data_2, 0) # CA
uc_s2_a1 = ng.pipe.make_uc(dic_2, data_2, 1) # N
uc_s2_a2 = ng.pipe.make_uc(dic_2, data_2, 2) # CX

uc_s3_a0 = ng.pipe.make_uc(dic_3, data_3, 0) # CO
uc_s3_a1 = ng.pipe.make_uc(dic_3, data_3, 1) # N
uc_s3_a2 = ng.pipe.make_uc(dic_3, data_3, 2) # CX

# read in assignments
table_filename = 'ass.tab'
table = ng.pipe.read_table(table_filename)[2]
assignments = table['ASS'][1:]

# set strip locations and limits
```
```python
x_center_s1 = table['N_PPM'][1:] # center of strip x axis in ppm, spectrum 1
x_center_s2 = table['N_PPM'][1:] # center of strip x axis in ppm, spectrum 2
x_center_s3 = table['N_PPM'][2:] # center in strip x axis in ppm, spectrum 3
x_width = 1.8 # width in ppm (+/-) of x axis for all strips
y_min = 40.0 # y axis minimum in ppm
y_max = 65.0 # y axis minimum in ppm
z_plane_s1 = table['CO_PPM'][:,] # strip plane in ppm, spectrum 1
z_plane_s2 = table['CA_PPM'][1:,:] # strip plane in ppm, spectrum 2
z_plane_s3 = table['CO_PPM'][1:,:] # strip plane in ppm, spectrum 3

fig = plt.figure()
for i in xrange(7):
    ### spectral 1, CONCA
    # find limits in units of points
    idx_s1_a1 = uc_s1_a1(z_plane_s1[i], "ppm")
    min_s1_a0 = uc_s1_a0(x_center_s1[i] + x_width, "ppm")
    max_s1_a0 = uc_s1_a0(x_center_s1[i] - x_width, "ppm")
    min_s1_a2 = uc_s1_a2(y_min, "ppm")
    max_s1_a2 = uc_s1_a2(y_max, "ppm")

    if min_s1_a2 > max_s1_a2:
        min_s1_a2, max_s1_a2 = max_s1_a2, min_s1_a2

    # extract strip
    strip_s1 = data_1[min_s1_a0:max_s1_a0+1, idx_s1_a1, min_s1_a2:max_s1_a2+1]

    # determine ppm limits of contour plot
    strip_ppm_x = uc_s1_a0.ppm_scale()[min_s1_a0:max_s1_a0+1]
    strip_ppm_y = uc_s1_a2.ppm_scale()[min_s1_a2:max_s1_a2+1]
    strip_x, strip_y = np.meshgrid(strip_ppm_x, strip_ppm_y)

    # add contour plot of strip to figure
    ax1 = fig.add_subplot(1, 21, 3 * i + 1)
    ax1.contour(strip_x, strip_y, strip_s1.transpose(), cl_s1,
                colors=colors_s1, linewidths=0.5)
    ax1.invert_yaxis() # flip axes since ppm indexed
    ax1.invert_xaxis()
    ax1.tick_params(axis='both', labelbottom=False, bottom=False, top=False,
                    labelleft=False, left=False, right=False,
                    direction='out')
    ax1.set_xlabel("%.1f"%(x_center_s1[i]), size=6)
    ax1.text(0.1, 0.975, "%.1f"%(z_plane_s1[i]), size=6,
             transform=ax1.transAxes)

    # turn off tick and labels, add labels
    ax1.tick_params(axis='both', labelbottom=False, bottom=False, top=False,
                    labelleft=False, left=False, right=False)
    ax1.set_xlabel("%.1f"%(x_center_s1[i]), size=6)
    ax1.text(0.1, 0.975, "%.1f"%(z_plane_s1[i]), size=6,
             transform=ax1.transAxes)

    # label and put ticks on first strip plot
    if i == 0:
        ax1.set_ylabel("13C (ppm)"
ax1.tick_params(axis='y', labelleft=True, left=True, direction='out')

    # label and put ticks on first strip plot
    if i == 0:
        ax1.set_ylabel("13C (ppm)"
ax1.tick_params(axis='y', labelleft=True, left=True, direction='out')

### spectra 2, NCACX
# find limits in units of points
idx_s2_a0 = uc_s2_a0(z_plane_s2[i], "ppm")
min_s2_a1 = uc_s2_a1(x_center_s2[i] + x_width, "ppm")
```

5.8. 3D strip plot example
```python
max_s2_a1 = uc_s2_a1(x_center_s2[i] - x_width, "ppm")
min_s2_a2 = uc_s2_a2(y_min, "ppm")
max_s2_a2 = uc_s2_a2(y_max, "ppm")

if min_s2_a2 > max_s2_a2:
    min_s2_a2, max_s2_a2 = max_s2_a2, min_s2_a2

# extract strip
strip_s2 = data_2[idx_s2_a0, min_s2_a1:max_s2_a1+1, min_s2_a2:max_s2_a2+1]

# add contour plot of strip to figure
ax2 = fig.add_subplot(1, 21, 3 * i + 2)
ax2.contour(strip_s2.transpose(), cl_s2, colors=colors_s2, linewidths=0.5)
ax2.tick_params(axis='both', labelbottom=False, bottom=False, top=False,
                left=False, left=False, right=False)
ax2.set_xlabel("%.1f"%(x_center_s2[i]), size=6)
ax2.text(0.2, 0.975, "%.1f"%(z_plane_s2[i]), size=6,
         transform=ax2.transAxes)
ax2.set_title(assignments[i])

### spectral 3, NCOCX
# find limits in units of points
idx_s3_a0 = uc_s3_a0(z_plane_s3[i], "ppm")
min_s3_a1 = uc_s3_a1(x_center_s3[i] + x_width, "ppm")
max_s3_a1 = uc_s3_a1(x_center_s3[i] - x_width, "ppm")
min_s3_a2 = uc_s3_a2(y_min, "ppm")
max_s3_a2 = uc_s3_a2(y_max, "ppm")

if min_s3_a2 > max_s3_a2:
    min_s3_a2, max_s3_a2 = max_s3_a2, min_s3_a2

# extract strip
strip_s3 = data_3[idx_s3_a0, min_s3_a1:max_s3_a1+1, min_s3_a2:max_s3_a2+1]

# add contour plot of strip to figure
ax3 = fig.add_subplot(1, 21, 3 * i + 3)
ax3.contour(strip_s3.transpose(), cl_s3, colors=colors_s3, linewidths=0.5)
ax3.tick_params(axis='both', labelbottom=False, bottom=False, top=False,
                labelleft=False, left=False, right=False)
ax3.set_xlabel("%.1f"%(x_center_s3[i]), size=6)
ax3.text(0.1, 0.975, "%.1f"%(z_plane_s3[i]), size=6,
         transform=ax3.transAxes)

# add X axis label, save figure
fig.text(0.45, 0.05, "15N (ppm)"
fig.savefig('strip_plots.png')
```
5.9 Relaxation trajectory analysis example

5.9.1 Introduction

This example is taken from Listing S12 - S15 in the 2013 JBNMR nmrglue paper. In this example a series of 3D NMRPipe files containing relaxation trajectories for a solid state NMR experiment and analyzed.

5.9.2 Instructions

Execute `python extract_traj.py` to extract the relaxation trajectories from the data set. ‘XXX.dat’ files are created for the peaks defined in the `boxes.in` file. The `spectra.in` file defines which spectra the trajectories will be extracted from.

Execute `python plot_boxes.py` to create plots showing the peak and integration limits for all peaks defined in `boxes.in`. `peak_XXX_spectrum_X.png` files are created for all peaks and spectra. A example plot is provided as Figure 7 of the paper, which corresponds to `peak_D40_spectrum_0.png`.

Execute `python fit_exp.py` to fit all relaxation trajectories. The fitting results are provided in the `fits.txt` file. The `relaxation_times.in` file defines the relaxation times for each spectra.

Execute `python plot_trajectories` to create plots of all experimental and fit relaxation trajectories. This script creates a series of `XXX_plot.png` files. An example plot is provided as Figure 8 of the paper, which corresponds to `D40_plot.png`.

The data used in this example is available for download part1, part2, part3, part4.
Listing S12

```
import nmrglue as ng
import numpy as np

# read the integration limits and list of spectra
peak_list = np.recfromtxt("boxes.in", names=True)
spectra_list = np.recfromtxt("spectra.in")

# create an array to hold the trajectories
trajectories = np.empty((peak_list.size, spectra_list.size), dtype='float')

# loop over the spectra
for sn, spectra in enumerate(spectra_list):
    # read in the spectra data
    print "Extracting peak intensities from:"
    dic, data = ng.pipe.read(spectra)
    # loop over the integration limits
    for i, (name, x0, y0, x1, y1) in enumerate(peak_list):
        if x0 > x1:
            x0, x1 = x1, x0
        if y0 > y1:
            y0, y1 = y1, y0

        # integrate the region and save in trajectories array
        trajectories[i][sn] = data[y0:y1 + 1, x0:x1 + 1].sum()

# write out the trajectories for each peak
for itraj, peak_traj in enumerate(trajectories):
    peak_traj /= peak_traj.max()  # normalize the peak's trajectory
    fname = peak_list.peak_label[itraj] + '.dat'
    f = open(fname, 'w')
    for v in peak_traj:
        f.write(str(v) + 'n')
    f.close()
```

Listing S13

```
import numpy as np
import nmrglue as ng
import matplotlib.pyplot as plt
import matplotlib.cm

# plot parameters
xpad = 5  # padding around peak box on x-axis
ypad = 5  # padding around peak box on y-axis
cmap = matplotlib.cm.Blues_r  # contour map (colors to use for contours)

# contour levels
cl = 30000 * 1.20 ** np.arange(20)

# read in the box limits and list of spectra
```
peak_list = np.recfromtxt("boxes.in", names=True)
spectra_list = np.recfromtxt("spectra.in")

# loop over the spectra
for spec_number, spectra in enumerate(spectra_list):
    # read in the spectral data
dic, data = ng.pipe.read(spectra)

    # loop over the peaks
    for peak, x0, y0, x1, y1 in peak_list:
        if x0 > x1:
            x0, x1 = x1, x0
        if y0 > y1:
            y0, y1 = y1, y0

        # slice the data around the peak
        slice = data[y0 - ypad:y1 + 1 + ypad, x0 - xpad:x1 + 1 + xpad]

        # create the figure
        fig = plt.figure()
        ax = fig.add_subplot(111)

        # plot the contours
        print "Plotting:", peak, spec_number
        extent = (x0 - xpad + 1, x1 + xpad - 1, y0 - ypad + 1, y1 + ypad - 1)
        ax.contour(slice, cl, cmap=cmap, extent=extent)

        # draw a box around the peak
        ax.plot([x0, x1, x1, x0, x0], [y0, y0, y1, y1, y0], 'k--')

        # draw lighter boxes at +/- 1 point
        ax.plot([x0 - 1, x1 + 1, x1 + 1, x0 - 1], [y0 - 1, y0 - 1, y1 + 1, y1 + 1], 'k--', alpha=0.35)
        ax.plot([x0 + 1, x1 - 1, x1 - 1, x0 + 1], [y0 + 1, y0 + 1, y1 - 1, y1 - 1], 'k--', alpha=0.35)

        # set the title, save the figure
        ax.set_title('Peak: %s Spectrum: %i' % (peak, spec_number))
        fig.savefig('peak_%s_spectrum_%i' % (peak, spec_number))
        del(fig)

Listing S14

[fit_exp.py]

import glob
import numpy as np
from nmrglue.analysis.leastsqbound import leastsqbound

# exponential function to fit data to.
def fit_func(p, x):
    A, R2 = p
    return A * np.exp(-1.0 * np.array(x) * R2 / 1.0e6)

# residuals between fit and experimental data.

5.9. Relaxation trajectory analysis example
```python
def residuals(p, y, x):
    err = y - fit_func(p, x)
    return err

# prepare fitting parameters
relaxation_times = np.loadtxt("relaxation_times.in")
x0 = [1.0, 0.10]  # initial fitting parameter
bounds = [(0.98, 1.02), (None, None)]  # fitting constraints

# create an output file to record the fitting results
output = open('fits.txt', 'w')
output.write("#Peak	A	R2	ier
")

# loop over the trajectory files
for filename in glob.glob('*.dat'):
    peak = filename[:3]
    print "Fitting Peak:", peak
    trajectory = np.loadtxt(filename)
    x, ier = leastsqbound(residuals, x0, bounds=bounds,
                          args=(trajectory, relaxation_times))
    output.write("%s	%.6f	%.6f	%i
" % (peak, x[0], x[1], ier))
output.close()  # close the output file
```

Listing S15

[plot_trajectories.py]

```python
import numpy as np
import matplotlib.pyplot as plt

# exponential function used to fit the data
def fit_func(p, x):
    A, R2 = p
    return A * np.exp(-1.0 * np.array(x) * R2 / 1.0e6)

fitting_results = np.recfromtxt('fits.txt')
experimental_relaxation_times = np.loadtxt("relaxation_times.in")
simulated_relaxation_times = np.linspace(0,4000000,2000)

# loop over the fitting results
for peak, A, R2, ier in fitting_results:
    print "Plotting:", peak
    # load the experimental and simulated relaxation trajectories
    experimental_trajectory = np.loadtxt(peak + '.dat')
simulated_trajectory = fit_func((A, R2), simulated_relaxation_times)

    # create the figure
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(experimental_trajectory, 'or')
```

314 Chapter 5. Examples from JBNMR article
```python
ax.plot(simulated_relaxation_times, simulated_trajectory, '-k')
ax.set_title(peak)
fig.savefig(peak+'_plot.png')
```

Example input files

[spectra.in]

data/Ytau_100.fid/test.ft2
data/Ytau_100000.fid/test.ft2
data/Ytau_250000.fid/test.ft2
data/Ytau_500000.fid/test.ft2
data/Ytau_750000.fid/test.ft2
data/Ytau_1000000.fid/test.ft2
data/Ytau_1500000.fid/test.ft2
data/Ytau_2000000.fid/test.ft2
data/Ytau_3000000.fid/test.ft2
data/Ytau_4000000.fid/test.ft2

[boxes.in]

<table>
<thead>
<tr>
<th>peak_label</th>
<th>X0</th>
<th>Y0</th>
<th>X1</th>
<th>Y1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A20</td>
<td>4068</td>
<td>938</td>
<td>4079</td>
<td>913</td>
</tr>
<tr>
<td>A24</td>
<td>3992</td>
<td>1013</td>
<td>4000</td>
<td>997</td>
</tr>
<tr>
<td>A26</td>
<td>4065</td>
<td>962</td>
<td>4075</td>
<td>940</td>
</tr>
<tr>
<td>A34</td>
<td>4009</td>
<td>985</td>
<td>4018</td>
<td>958</td>
</tr>
<tr>
<td>A48</td>
<td>4028</td>
<td>1034</td>
<td>4036</td>
<td>1010</td>
</tr>
<tr>
<td>C28</td>
<td>4035</td>
<td>1115</td>
<td>4044</td>
<td>1092</td>
</tr>
<tr>
<td>D36</td>
<td>3994</td>
<td>987</td>
<td>4003</td>
<td>973</td>
</tr>
<tr>
<td>D40</td>
<td>4076</td>
<td>802</td>
<td>4085</td>
<td>774</td>
</tr>
<tr>
<td>D46</td>
<td>4155</td>
<td>899</td>
<td>4163</td>
<td>883</td>
</tr>
<tr>
<td>D47</td>
<td>4053</td>
<td>967</td>
<td>4062</td>
<td>941</td>
</tr>
<tr>
<td>E15</td>
<td>4162</td>
<td>1022</td>
<td>4170</td>
<td>996</td>
</tr>
<tr>
<td>E19</td>
<td>4176</td>
<td>902</td>
<td>4185</td>
<td>875</td>
</tr>
<tr>
<td>E27</td>
<td>4036</td>
<td>1084</td>
<td>4044</td>
<td>1054</td>
</tr>
<tr>
<td>E42</td>
<td>4136</td>
<td>1055</td>
<td>4142</td>
<td>1026</td>
</tr>
<tr>
<td>E56</td>
<td>4107</td>
<td>821</td>
<td>4115</td>
<td>794</td>
</tr>
<tr>
<td>F30</td>
<td>4013</td>
<td>1060</td>
<td>4023</td>
<td>1031</td>
</tr>
<tr>
<td>F52</td>
<td>4097</td>
<td>828</td>
<td>4105</td>
<td>799</td>
</tr>
<tr>
<td>G09</td>
<td>4054</td>
<td>1249</td>
<td>4063</td>
<td>1220</td>
</tr>
<tr>
<td>G14</td>
<td>4068</td>
<td>1331</td>
<td>4077</td>
<td>1304</td>
</tr>
<tr>
<td>G38</td>
<td>4098</td>
<td>1254</td>
<td>4106</td>
<td>1227</td>
</tr>
<tr>
<td>G41</td>
<td>4091</td>
<td>1283</td>
<td>4099</td>
<td>1259</td>
</tr>
<tr>
<td>I06</td>
<td>4087</td>
<td>903</td>
<td>4096</td>
<td>884</td>
</tr>
</tbody>
</table>

[relaxation_times.in]

<table>
<thead>
<tr>
<th># time in us</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>100000</td>
</tr>
<tr>
<td>250000</td>
</tr>
<tr>
<td>500000</td>
</tr>
<tr>
<td>750000</td>
</tr>
<tr>
<td>1000000</td>
</tr>
<tr>
<td>1500000</td>
</tr>
<tr>
<td>2000000</td>
</tr>
<tr>
<td>3000000</td>
</tr>
<tr>
<td>4000000</td>
</tr>
</tbody>
</table>

5.9. Relaxation trajectory analysis example
Example output

[fits.txt]

<table>
<thead>
<tr>
<th>Peak</th>
<th>A</th>
<th>R2</th>
<th>ier</th>
</tr>
</thead>
<tbody>
<tr>
<td>F30</td>
<td>0.981006</td>
<td>0.259152</td>
<td>1</td>
</tr>
<tr>
<td>D40</td>
<td>0.980906</td>
<td>0.328635</td>
<td>1</td>
</tr>
<tr>
<td>G14</td>
<td>0.994899</td>
<td>0.108697</td>
<td>1</td>
</tr>
<tr>
<td>A48</td>
<td>0.984907</td>
<td>0.125793</td>
<td>1</td>
</tr>
<tr>
<td>E15</td>
<td>0.996023</td>
<td>0.088139</td>
<td>1</td>
</tr>
<tr>
<td>D47</td>
<td>0.993891</td>
<td>0.101194</td>
<td>1</td>
</tr>
<tr>
<td>G38</td>
<td>1.005751</td>
<td>0.127826</td>
<td>1</td>
</tr>
<tr>
<td>F52</td>
<td>1.014378</td>
<td>0.063087</td>
<td>1</td>
</tr>
<tr>
<td>G09</td>
<td>0.993838</td>
<td>0.097431</td>
<td>1</td>
</tr>
<tr>
<td>A34</td>
<td>1.002384</td>
<td>0.152056</td>
<td>1</td>
</tr>
<tr>
<td>I06</td>
<td>0.997428</td>
<td>0.060497</td>
<td>1</td>
</tr>
<tr>
<td>E56</td>
<td>0.987760</td>
<td>0.157588</td>
<td>1</td>
</tr>
<tr>
<td>D36</td>
<td>0.985922</td>
<td>0.153331</td>
<td>1</td>
</tr>
<tr>
<td>A20</td>
<td>0.999985</td>
<td>0.102821</td>
<td>1</td>
</tr>
<tr>
<td>E27</td>
<td>0.993052</td>
<td>0.298782</td>
<td>1</td>
</tr>
<tr>
<td>D46</td>
<td>0.980000</td>
<td>0.074505</td>
<td>1</td>
</tr>
<tr>
<td>C28</td>
<td>0.980000</td>
<td>0.680741</td>
<td>1</td>
</tr>
<tr>
<td>E42</td>
<td>0.996588</td>
<td>0.210449</td>
<td>1</td>
</tr>
<tr>
<td>A26</td>
<td>0.989291</td>
<td>0.258348</td>
<td>1</td>
</tr>
<tr>
<td>E19</td>
<td>0.992583</td>
<td>0.165457</td>
<td>1</td>
</tr>
<tr>
<td>G41</td>
<td>0.980000</td>
<td>0.528399</td>
<td>1</td>
</tr>
<tr>
<td>A24</td>
<td>0.992186</td>
<td>0.417901</td>
<td>1</td>
</tr>
</tbody>
</table>
5.9. Relaxation trajectory analysis example
Chapter 5. Examples from JBNMR article
This guide provides instructions for setting up an environment for developing nmrglue and an overview of the project layout and contribution process.

### 6.1 Requirements

To create an environment for developing nmrglue the following must be installed and available.

- Numpy
- SciPy
- nose
- Sphinx
- git

In addition the following Python packages are highly recommended.

- matplotlib
- IPython

A easy way of obtaining and installing these packages is to use a Python distribution which provides these packages, such as EPD. Detailed information on installing a Scipy stack is available.

Finally, other NMR software packages must be installed to process and convert the test and example data. These are not required for using nmrglue, but are needed to verify its functionality and to run some of the examples.

- NMRPipe
- Sparky
- The Rowland NMR Toolkit
- SIMPSON
6.2 Source Code

nmrglue uses github for source code hosting. For access to the source code, see the nmrglue github site.

To check out the latest version of nmrglue use git:

```
    git clone git://github.com/jjhelmus/nmrglue.git
```

nmrglue is a pure python module, the root directory can be included in your PYTHONPATH directly, or a symbolic link can be added to the site-packages directory of your Python install. In this way any modifications to the nmrglue source tree will be picked up when nmrglue is imported.

6.3 Test and Example Data

nmrglue uses experimental and simulated NMR data for testing and in many examples, this data is divided into two archives, the test data set and additional data needed for the examples. The nmrglue test data sets must be downloaded and unpacked into a directory (a directory named data under the root directory is recommended but not required). The conversions scripts contained in the archive must be run to convert and process the time domain NMR data. Additional NMR software (NMRPipe, etc) are required for this processing and conversion, see the README file in the test data archive for details. After installing this test data edit the setup.py file in the test directory and the make_links.sh file in the examples directory to correctly point to the location of the test data directory. Additional data required for the nmrglue examples can be downloaded as a single archive. Unpack this archive in the examples directory. Run the make_links.sh shell script to make symbolic links to the test data which reused in a number of example. On operating systems which do not support symbolic links (Windows), the data in the test data directory will need to be copied by hand into the appropriate locations.

6.4 Project Layout

The directory layout of the nmrglue project is as follows.

- **nmrglue**: source code for the project.
- **doc**: contains the setup file and source code for building the nmrglue documentation using Sphinx.
- **tests**: unit tests which use the nose framework to verify the functionality of nmrglue. See the Testing section for details.
- **example**: contains numerous examples in which nmrglue is used to solve many real world NMR problems.

Two additional directories can be created to aid in developements. These are not required but will be ignored by git using the default .gitignore file

- **data**: Suggested location to hold the test data.
- **sandbox**: Suggested location to store code, data, etc not yet ready to be include in nmrglue.

6.5 Suggestions

When working with the nmrglue source code please consider the following when preparing patches.

- **Coding Style**: The nmrglue source code tries to follow the PEP8 style guide. Consider using a tool, such as pep8 or pylint to check your Python code against these conventions.
• Documentation: All public functions and classes should have docstrings which follow the NumPy/SciPy documentation standard. Private functions and classes may have shorter docstrings. The nmrglue documentation is built using Sphinx. Sphinx translates reST formatted documents (including docstring) into HTML. When adding new function, classes or parameter to nmrglue please update the docstring and make any necessary changes to the Sphinx files in the doc directory.

• Testing: Tests are available for verifying the functionality of nmrglue, please include a tests when adding new functionality to the package.

• Examples: Numerous example showing real world use of nmrglue are provided in the examples directory. Contributions of additional example are welcome and appreciated.

6.6 Testing

Tests for verifying the functionality of nmrglue are available in the test directory. These tests use the nose testing infrastructure.

6.6.1 Requirements

To run these tests NumPy, SciPy, nmrglue, and nose must be installed and in the Python search path. NMRPipe must be installed to run the pipe_proc tests.

In addition, the location of the the test data sets must be specified in the setup.py file in the test directory. The nmrglue test data is available for download.

In order to run all nmrglue unit tests, the tests data sets must be downloaded, unpacked, and the all conversions scripts contained in the archive must be run. Many of these scripts require additional NMR software (NMRPipe, etc), see the README file in the test data achive for additional details. A subset of the full test suite can be run without installing any additional software.

6.6.2 Running the unit tests

After ensuring that all required packages are installed and setup.py correctly points to the location of the test data directory, the unit tests can be run using the following:

```bash
nosetest
```

Unit tests for a specific module can be run using:

```bash
nosetest tests/test_pipe.py
```

Additional information on the usage of the nosetest command is available.

6.7 Reporting Bugs

The preferred location for submitting feature requests and bugs reported is the github issue tracker. Reports are also welcomed on the nmrglue mailing list or by contacting Jonathan Helmus directly.
6.8 Contributions

Contribution of source code or examples to nmrglue is welcomed provided the contents can be distributed under the New BSD License. The preferred method for contributing is by creating a feature branch on a github fork of nmrglue and submitting a pull request, although patches are also accepted. Refer to the Numpy/SciPy git workflow for details on how to prepare a patch or submit a pull request.
<table>
<thead>
<tr>
<th>Module/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmrglue.analysis.analysisbase</td>
<td>199</td>
</tr>
<tr>
<td>nmrglue.analysis.helpers</td>
<td>206</td>
</tr>
<tr>
<td>nmrglue.analysis.integration</td>
<td>207</td>
</tr>
<tr>
<td>nmrglue.analysis.leastsqbound</td>
<td>209</td>
</tr>
<tr>
<td>nmrglue.analysis.linsh</td>
<td>221</td>
</tr>
<tr>
<td>nmrglue.analysis.lineshapes1d</td>
<td>212</td>
</tr>
<tr>
<td>nmrglue.analysis.peakpick</td>
<td>229</td>
</tr>
<tr>
<td>nmrglue.analysis.segmentation</td>
<td>235</td>
</tr>
<tr>
<td>nmrglue.fileio.bruker</td>
<td>15</td>
</tr>
<tr>
<td>nmrglue.fileio.convert</td>
<td>30</td>
</tr>
<tr>
<td>nmrglue.fileio.fileiobase</td>
<td>34</td>
</tr>
<tr>
<td>nmrglue.fileio.jcampdx</td>
<td>38</td>
</tr>
<tr>
<td>nmrglue.fileio.nmrml</td>
<td>39</td>
</tr>
<tr>
<td>nmrglue.fileio.pipe</td>
<td>40</td>
</tr>
<tr>
<td>nmrglue.fileio.rnmrtk</td>
<td>57</td>
</tr>
<tr>
<td>nmrglue.fileio.simpson</td>
<td>66</td>
</tr>
<tr>
<td>nmrglue.fileio.sparky</td>
<td>68</td>
</tr>
<tr>
<td>nmrglue.fileio.table</td>
<td>80</td>
</tr>
<tr>
<td>nmrglue.fileio.tecmag</td>
<td>85</td>
</tr>
<tr>
<td>nmrglue.fileio.varian</td>
<td>86</td>
</tr>
<tr>
<td>nmrglue.process.pipe_proc</td>
<td>161</td>
</tr>
<tr>
<td>nmrglue.process.proc_autophase</td>
<td>105</td>
</tr>
<tr>
<td>nmrglue.process.proc_base</td>
<td>107</td>
</tr>
<tr>
<td>nmrglue.process.proc_bl</td>
<td>146</td>
</tr>
<tr>
<td>nmrglue.process.proc_lp</td>
<td>150</td>
</tr>
<tr>
<td>nmrglue.util.misc</td>
<td>241</td>
</tr>
</tbody>
</table>
Symbols

__init__() (nmrglue.analysis.analysisbase.ndwindow method), 204
__init__() (nmrglue.analysis.analysisbase.ndwindow_index method), 204
__init__() (nmrglue.analysis.analysisbase.ndwindow_inside method), 205
__init__() (nmrglue.analysis.analysisbase.ndwindow_inside_index method), 205
__init__() (nmrglue.analysis.lineshapes1d.gauss_fwhm method), 217
__init__() (nmrglue.analysis.lineshapes1d.gauss_sigma method), 217
__init__() (nmrglue.analysis.lineshapes1d.location_2params method), 219
__init__() (nmrglue.analysis.lineshapes1d.location_scale method), 216
__init__() (nmrglue.analysis.lineshapes1d.lorentz_fwhm method), 218
__init__() (nmrglue.analysis.lineshapes1d.lorentz_gamma method), 218
__init__() (nmrglue.analysis.lineshapes1d.pvoigt_fwhm method), 220
__init__() (nmrglue.analysis.lineshapes1d.scale method), 220
__init__() (nmrglue.analysis.lineshapes1d.voigt_fwhm method), 219
__init__() (nmrglue.analysis.lineshapes1d.voigt_sigmagamma method), 219
__init__() (nmrglue.fileio.bruker.bruker_nd method), 29
__init__() (nmrglue.fileio.bruker.bruker_base.data_nd_method), 29
__init__() (nmrglue.fileio.fileiobase.data_nd_method), 38
__init__() (nmrglue.fileio.fileiobase.unit_conversion method), 37
__init__() (nmrglue.fileio.pipe.iter3D method), 45
__init__() (nmrglue.fileio.pipe.pipe_2d method), 54
__init__() (nmrglue.fileio.pipe.pipe_3d method), 55
__init__() (nmrglue.fileio.pipe.pipe_4d method), 56
__init__() (nmrglue.fileio.pipe.pipestream_3d method), 55
__init__() (nmrglue.fileio.pipe.pipestream_4d method), 56
__init__() (nmrglue.fileio.rnmrtk.rnmrtk_nd method), 64
__init__() (nmrglue.fileio.sparky.sparky_2d method), 79
__init__() (nmrglue.fileio.sparky.sparky_3d method), 79
__init__() (nmrglue.fileio.sparky.sparky_4d method), 79
__init__() (nmrglue.fileio.varian.fid_nd method), 104
get_nmrml_data() (in module nmrglue.fileio.nmrml), 40
_get_apcme_score() (in module nmrglue.process.proc_autophase), 106
_get_peak_minima_score() (in module nmrglue.process.proc_autophase), 106
_get_ps() (in module nmrglue.process.proc_autophase), 106

A

abs() (in module nmrglue.process.proc_base), 132
add() (in module nmrglue.process.pipe_proc), 181
add() (in module nmrglue.process.proc_base), 124
add_axis_to_dic() (in module nmrglue.process.pipe_proc), 124
add_axis_to_udic() (in module nmrglue.process.pipe_proc), 124
add_edge() (in module nmrglue.analysis.peakpick), 232
add_ri() (in module nmrglue.process.proc_base), 124
add_to_table() (in module nmrglue.process.proc_base), 124
append_column() (in module nmrglue.fileio.table), 83
append_data() (in module nmrglue.fileio.pipeline), 52
append_imag() (in module nmrglue.process.proc_base), 130
append_row() (in module nmrglue.fileio.table), 81
autops() (in module nmrglue.process.proc_autophase), 105
avg_flt() (in module nmrglue.process.proc_base), 145
axisheader2dic() (in module nmrglue.fileio.sparky), 78

B

base() (in module nmrglue.process.pipe_proc), 178
base() (in module nmrglue.process.proc_bl), 146
baseline_corrector() (in module nmrglue.process.proc_bl), 148
Index

C

bin2int() (in module nmrglue.process.proc_base), 145
c2ri() (in module nmrglue.process.proc_base), 129
cadzow() (in module nmrglue.process.proc_lp), 153
cadzow_single() (in module nmrglue.process.proc_lp), 157
calc_bl_linear() (in module nmrglue.process.proc_bl), 149
calc_bl_med() (in module nmrglue.process.proc_bl), 149
calc_errors() (in module nmrglue.analysis.linesh), 227
calc_tshape() (in module nmrglue.fileio.sparky), 73
cbf() (in module nmrglue.process.pipe_proc), 179
cbf() (in module nmrglue.process.proc_bl), 147
cbf_explicit() (in module nmrglue.process.proc_bl), 147
center() (in module nmrglue.process.proc_base), 142
center_fwhm() (in module nmrglue.analysis.lineshapes1d), 216
center_fwhm_bymoments() (in module nmrglue.analysis.lineshapes1d), 216
cfs2bytes() (in module nmrglue.fileio.simpson), 68
collectors() (in module nmrglue.analysis.peakpick), 232
coad() (in module nmrglue.process.pipe_proc), 189
coad() (in module nmrglue.process.proc_base), 133
coaddd() (in module nmrglue.process.pipe_proc), 188
coadd() (in module nmrglue.process.proc_base), 133
cof2amp() (in module nmrglue.process.proc_lp), 157
cof2phase() (in module nmrglue.process.proc_lp), 157
complexify_data() (in module nmrglue.fileio.bruker), 27
conv() (in module nmrglue.process.proc_base), 134
converter (class in nmrglue.fileio.convert), 30
covolutet() (in module nmrglue.process.proc_base), 134
corr() (in module nmrglue.process.proc_base), 135
corrlet() (in module nmrglue.process.proc_base), 135
create_acous_dic() (in module nmrglue.fileio.bruker), 26
create_axisdic() (in module nmrglue.fileio.sparky), 73
create_blank_udic() (in module nmrglue.fileio.fileiobase), 34
create_data() (in module nmrglue.fileio.bruker), 26
create_data() (in module nmrglue.fileio.pipe), 47
create_data() (in module nmrglue.fileio.rnmrtk), 62
create_data() (in module nmrglue.fileio.sparky), 73
create_dic() (in module nmrglue.fileio.bruker), 95
create_dic() (in module nmrglue.fileio.bruker), 25
create_dic() (in module nmrglue.fileio.bruker), 44
create_dic() (in module nmrglue.fileio.rnmrtk), 61
create_dic() (in module nmrglue.fileio.sparky), 71
create_dic() (in module nmrglue.fileio.varian), 93
create_empty_dic() (in module nmrglue.fileio.pipe), 47
create_dic_param() (in module nmrglue.fileio.varian), 95
cs() (in module nmrglue.process.proc_base), 113
cs() (in module nmrglue.process.pipe_proc), 171
datab() (in module nmrglue.process.proc_base), 113
date_time2dic() (in module nmrglue.fileio.pipe), 45
date_time2dic() (in module nmrglue.fileio.sparky), 72
delete_column() (in module nmrglue.fileio.pipe), 83
delete_row() (in module nmrglue.fileio.pipe), 82
data_nd (class in nmrglue.fileio.fileiobase), 38
datetime2dic() (in module nmrglue.fileio.pipe), 45
datetime2dic() (in module nmrglue.fileio.sparky), 72
decode_States() (in module nmrglue.process.proc_base), 129
dia() (in module nmrglue.process.proc_base), 176
dia() (in module nmrglue.process.proc_base), 118
dic2axisheader() (in module nmrglue.fileio.sparky), 78
dic2blockheader() (in module nmrglue.fileio.varian), 102
dic2datetime() (in module nmrglue.fileio.pipe), 45
dic2datetime() (in module nmrglue.fileio.sparky), 72
dic2fdata() (in module nmrglue.fileio.pipe), 52
dic2fheader() (in module nmrglue.fileio.pipe), 78
dic2fheader() (in module nmrglue.fileio.varian), 103
dic2header() (in module nmrglue.fileio.varian), 102
dx() (in module nmrglue.process.proc_base), 182
index() (in module nmrglue.process.proc_base), 124
E

exlr() (in module nmrglue.process.proc_base), 130
extrapolate() (in module nmrglue.process.proc_base), 125
extrapolate_2d() (in module nmrglue.process.proc_base), 161
extrapolate_2d() (in module nmrglue.process.proc_base), 157
f_NDregion() (in module nmrglue.analysis.linesh), 228
fft() (in module nmrglue.process.proc_base), 114
fft() (in module nmrglue.process.proc_base), 114
fft() (in module nmrglue.process.proc_base), 115
f_NDregion() (in module nmrglue.analysis.linesh), 228
fdata2dic() (in module nmrglue.fileio.pipe), 52
data2dic() (in module nmrglue.fileio.pipe), 52
fft_positive() (in module nmrglue.process.proc_base), 116
fid_nd (class in nmrglue.fileio.varian), 104
fileheader2dic() (in module nmrglue.fileio.varian), 102
filter_amax() (in module nmrglue.process.proc_base), 138
filter_amin() (in module nmrglue.process.proc_base), 138
filter_avg() (in module nmrglue.process.proc_base), 139
filter_dev() (in module nmrglue.process.proc_base), 139
filter_generic() (in module nmrglue.process.proc_base), 140
filter_max() (in module nmrglue.process.proc_base), 136
filter_median() (in module nmrglue.process.proc_base), 136
filter_min() (in module nmrglue.process.proc_base), 136
filter_percentile() (in module nmrglue.process.proc_base), 137
filter_range() (in module nmrglue.process.proc_base), 139
filter_rank() (in module nmrglue.process.proc_base), 139
filter_sum() (in module nmrglue.process.proc_base), 139
find_all_connected() (in module nmrglue.analysis.segmentation), 236
find_all_downward() (in module nmrglue.analysis.segmentation), 237
find_all_nconnected() (in module nmrglue.analysis.segmentation), 236
find_all_nthres() (in module nmrglue.analysis.peakpick), 234
find_all_nthres_fast() (in module nmrglue.analysis.peakpick), 234
find_all_thres() (in module nmrglue.analysis.peakpick), 234
find_all_thres_fast() (in module nmrglue.analysis.peakpick), 234
find_all_upward() (in module nmrglue.analysis.segmentation), 239
fit_NDregion() (in module nmrglue.analysis.linesh), 223
fit_spectrum() (in module nmrglue.analysis.linesh), 221
fix_roots() (in module nmrglue.process.proc_lp), 161
from_bruker() (nmrglue.fileio.convert.converter method), 30
from_pipe() (nmrglue.fileio.convert.converter method), 31
from_rnmrtk() (nmrglue.fileio.convert.converter method), 31
from_sparky() (nmrglue.fileio.convert.converter method), 31
from_universal() (nmrglue.fileio.convert.converter method), 31
from_varian() (nmrglue.fileio.convert.converter method), 31
fsh() (in module nmrglue.process.pipe_proc), 172
fsh() (in module nmrglue.process.proc_base), 173

G

gauss_fwhm (class in nmrglue.analysis.lineshapes1d), 217
gauss_sigma (class in nmrglue.analysis.lineshapes1d), 217
get_axisheader() (in module nmrglue.fileio.varian), 78
get_block() (in module nmrglue.fileio.varian), 97
get_block_ntraces() (in module nmrglue.fileio.varian), 98
get_blockheader() (in module nmrglue.fileio.varian), 99
get_data() (in module nmrglue.fileio.bruker), 27
get_data() (in module nmrglue.fileio.pipe), 53
get_data() (in module nmrglue.fileio.rnmrtk), 62
get_data() (in module nmrglue.fileio.sparky), 75
get_fdata() (in module nmrglue.fileio.pipe), 53
get_fileheader() (in module nmrglue.fileio.varian), 77
### nmrglue Documentation, Release 0.7-dev

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_fileheader()</td>
<td>nmrglue.fileio.varian</td>
<td>99</td>
</tr>
<tr>
<td>get_hyperheader()</td>
<td>nmrglue.fileio.varian</td>
<td>100</td>
</tr>
<tr>
<td>get_nblocks()</td>
<td>nmrglue.fileio.varian</td>
<td>97</td>
</tr>
<tr>
<td>get_nblocks_ntraces()</td>
<td>nmrglue.fileio.varian</td>
<td>98</td>
</tr>
<tr>
<td>get_parameter()</td>
<td>nmrglue.fileio.varian</td>
<td>104</td>
</tr>
<tr>
<td>get_tile()</td>
<td>nmrglue.fileio.sparky</td>
<td>75</td>
</tr>
<tr>
<td>get_tilen()</td>
<td>nmrglue.fileio.sparky</td>
<td>74</td>
</tr>
<tr>
<td>get_trace()</td>
<td>nmrglue.fileio.pipe</td>
<td>53</td>
</tr>
<tr>
<td>get_trace()</td>
<td>nmrglue.fileio.rnmrtk</td>
<td>62</td>
</tr>
<tr>
<td>get_trace()</td>
<td>nmrglue.fileio.varian</td>
<td>99</td>
</tr>
<tr>
<td>glue2pipe()</td>
<td>nmrglue.fileio.table</td>
<td>84</td>
</tr>
<tr>
<td>gm()</td>
<td>nmrglue.process.pipe_proc</td>
<td>164</td>
</tr>
<tr>
<td>gm()</td>
<td>nmrglue.process.proc_base</td>
<td>108</td>
</tr>
<tr>
<td>gmb()</td>
<td>nmrglue.process.pipe_proc</td>
<td>165</td>
</tr>
<tr>
<td>gmb()</td>
<td>nmrglue.process.proc_base</td>
<td>108</td>
</tr>
<tr>
<td>gram_schmidt()</td>
<td>nmrglue.process.proc_base</td>
<td>141</td>
</tr>
<tr>
<td>gray()</td>
<td>nmrglue.process.proc_base</td>
<td>145</td>
</tr>
<tr>
<td>guess_ftype()</td>
<td>nmrglue.fileio.simpson</td>
<td>68</td>
</tr>
<tr>
<td>guess_pformat()</td>
<td>nmrglue.fileio.table</td>
<td>85</td>
</tr>
<tr>
<td>guess_shape()</td>
<td>nmrglue.fileio.bruker</td>
<td>26</td>
</tr>
<tr>
<td>guess_shape_and_submatrix_shape()</td>
<td>nmrglue.fileio.bruker</td>
<td>26</td>
</tr>
<tr>
<td>guess_udic()</td>
<td>nmrglue.fileio.bruker</td>
<td>24</td>
</tr>
<tr>
<td>guess_udic()</td>
<td>nmrglue.fileio.jcampdxt</td>
<td>39</td>
</tr>
<tr>
<td>guess_udic()</td>
<td>nmrglue.fileio.pipe</td>
<td>44</td>
</tr>
<tr>
<td>guess_udic()</td>
<td>nmrglue.fileio.rnmrtk</td>
<td>61</td>
</tr>
<tr>
<td>guess_udic()</td>
<td>nmrglue.fileio.sparky</td>
<td>71</td>
</tr>
<tr>
<td>guess_udic()</td>
<td>nmrglue.fileio.tecmag</td>
<td>85</td>
</tr>
<tr>
<td>guess_udic()</td>
<td>nmrglue.fileio.varian</td>
<td>93</td>
</tr>
<tr>
<td>ha()</td>
<td>nmrglue.process.pipe_proc</td>
<td>174</td>
</tr>
<tr>
<td>ha()</td>
<td>nmrglue.process.proc_base</td>
<td>117</td>
</tr>
<tr>
<td>ht()</td>
<td>nmrglue.process.pipe_proc</td>
<td>175</td>
</tr>
<tr>
<td>ht()</td>
<td>nmrglue.process.proc_base</td>
<td>118</td>
</tr>
<tr>
<td>hyperheader2dic()</td>
<td>nmrglue.fileio.varian</td>
<td>102</td>
</tr>
<tr>
<td>ifft()</td>
<td>nmrglue.process.proc_base</td>
<td>116</td>
</tr>
<tr>
<td>ifft_norm()</td>
<td>nmrglue.process.proc_base</td>
<td>116</td>
</tr>
<tr>
<td>ifft_positive()</td>
<td>nmrglue.process.proc_base</td>
<td>116</td>
</tr>
<tr>
<td>img()</td>
<td>nmrglue.process.pipe_proc</td>
<td>117</td>
</tr>
<tr>
<td>index2trace_flat()</td>
<td>nmrglue.fileio.pipe</td>
<td>36</td>
</tr>
<tr>
<td>index2trace_opp()</td>
<td>nmrglue.fileio.pipe</td>
<td>36</td>
</tr>
<tr>
<td>index2trace_reg()</td>
<td>nmrglue.fileio.pipe</td>
<td>36</td>
</tr>
<tr>
<td>insert_column()</td>
<td>nmrglue.fileio.table</td>
<td>82</td>
</tr>
<tr>
<td>insert_row()</td>
<td>nmrglue.fileio.table</td>
<td>81</td>
</tr>
<tr>
<td>int2bin()</td>
<td>nmrglue.process.proc_base</td>
<td>145</td>
</tr>
<tr>
<td>integ()</td>
<td>nmrglue.process.pipe_proc</td>
<td>183</td>
</tr>
<tr>
<td>integ()</td>
<td>nmrglue.process.proc_base</td>
<td>126</td>
</tr>
<tr>
<td>integrate()</td>
<td>nmrglue.analysis.integration</td>
<td>207</td>
</tr>
<tr>
<td>interleave_complex()</td>
<td>nmrglue.process.proc_base</td>
<td>129</td>
</tr>
<tr>
<td>interleave_data()</td>
<td>nmrglue.fileio.rnmrtk</td>
<td>63</td>
</tr>
<tr>
<td>interleavedata()</td>
<td>nmrglue.fileio.varian</td>
<td>103</td>
</tr>
<tr>
<td>irft()</td>
<td>nmrglue.process.proc_base</td>
<td>114</td>
</tr>
<tr>
<td>isdatasimilar()</td>
<td>nmrglue.util.misc</td>
<td>242</td>
</tr>
<tr>
<td>isdicsimilar()</td>
<td>nmrglue.util.misc</td>
<td>242</td>
</tr>
<tr>
<td>isitemsimilar()</td>
<td>nmrglue.util.misc</td>
<td>242</td>
</tr>
<tr>
<td>islistsimilar()</td>
<td>nmrglue.util.misc</td>
<td>242</td>
</tr>
<tr>
<td>iter3D()</td>
<td>nmrglue.fileio.pipe</td>
<td>45</td>
</tr>
<tr>
<td>jmod()</td>
<td>nmrglue.process.pipe_proc</td>
<td>166</td>
</tr>
<tr>
<td>jmod()</td>
<td>nmrglue.process.proc_base</td>
<td>109</td>
</tr>
<tr>
<td>label_connected()</td>
<td>nmrglue.analysis.segmentation</td>
<td>238</td>
</tr>
<tr>
<td>label_downward()</td>
<td>nmrglue.analysis.segmentation</td>
<td>238</td>
</tr>
<tr>
<td>label_downward_seg()</td>
<td>nmrglue.analysis.segmentation</td>
<td>240</td>
</tr>
<tr>
<td>label_nconnected()</td>
<td>nmrglue.analysis.segmentation</td>
<td>238</td>
</tr>
<tr>
<td>label_upward()</td>
<td>nmrglue.analysis.segmentation</td>
<td>238</td>
</tr>
<tr>
<td>label_upward_seg()</td>
<td>nmrglue.analysis.segmentation</td>
<td>240</td>
</tr>
<tr>
<td>largest_power_of_2()</td>
<td>nmrglue.process.proc_base</td>
<td>146</td>
</tr>
<tr>
<td>leastsqbound()</td>
<td>nmrglue.analysis.leastsqbound</td>
<td>209</td>
</tr>
<tr>
<td>limit()</td>
<td>nmrglue.analysis.helpers</td>
<td>206</td>
</tr>
<tr>
<td>limits2slice()</td>
<td>nmrglue.analysis.analysisbase</td>
<td>202</td>
</tr>
<tr>
<td>location_2params</td>
<td>nmrglue.analysis.lineshapes1d</td>
<td>219</td>
</tr>
<tr>
<td>location_scale</td>
<td>nmrglue.analysis.lineshapes1d</td>
<td>216</td>
</tr>
<tr>
<td>lorentz_fwhm</td>
<td>nmrglue.analysis.lineshapes1d</td>
<td>218</td>
</tr>
<tr>
<td>lorentz_gamma</td>
<td>nmrglue.analysis.lineshapes1d</td>
<td>218</td>
</tr>
<tr>
<td>lp()</td>
<td>nmrglue.process.pipe_proc</td>
<td>195</td>
</tr>
<tr>
<td>lp()</td>
<td>nmrglue.process.proc_lp</td>
<td>150</td>
</tr>
<tr>
<td>lp2d()</td>
<td>nmrglue.process.pipe_proc</td>
<td>197</td>
</tr>
<tr>
<td>lp2d()</td>
<td>nmrglue.process.proc_lp</td>
<td>152</td>
</tr>
<tr>
<td>lp_1d()</td>
<td>nmrglue.process.proc_lp</td>
<td>156</td>
</tr>
<tr>
<td>lp_cho()</td>
<td>nmrglue.process.proc_lp</td>
<td>152</td>
</tr>
<tr>
<td>Function</td>
<td>Module</td>
<td>Reference</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>order_data()</td>
<td>nmrglue.fileio.varian</td>
<td>96</td>
</tr>
<tr>
<td>pack_complex()</td>
<td>nmrglue.fileio.pipe</td>
<td>51</td>
</tr>
<tr>
<td>pack_complex()</td>
<td>nmrglue.process.proc_base</td>
<td>129</td>
</tr>
<tr>
<td>pack_table()</td>
<td>nmrglue.analysis.linesh</td>
<td>225</td>
</tr>
<tr>
<td>pack_table()</td>
<td>nmrglue.analysis.peakpick</td>
<td>233</td>
</tr>
<tr>
<td>pair_similar()</td>
<td>nmrglue.util.misc</td>
<td>241</td>
</tr>
<tr>
<td>parse_jcamp_line()</td>
<td>nmrglue.fileio.varian</td>
<td>28</td>
</tr>
<tr>
<td>parse_jcamp_value()</td>
<td>nmrglue.fileio.varian</td>
<td>28</td>
</tr>
<tr>
<td>parse_par_line()</td>
<td>nmrglue.fileio.rnmrtk</td>
<td>63</td>
</tr>
<tr>
<td>pick()</td>
<td>nmrglue.analysis.peakpick</td>
<td>229</td>
</tr>
<tr>
<td>pip2glue()</td>
<td>nmrglue.fileio.pipe</td>
<td>84</td>
</tr>
<tr>
<td>pipe_2d (class in nmrglue.fileio.pipe)</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>pipe_3d (class in nmrglue.fileio.pipe)</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>pipe_4d (class in nmrglue.fileio.pipe)</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>pipeline_3d (class in nmrglue.fileio.pipe)</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>pipeline_4d (class in nmrglue.fileio.pipe)</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>poly()</td>
<td>nmrglue.process.pipe_proc</td>
<td>199</td>
</tr>
<tr>
<td>ps()</td>
<td>nmrglue.process.pipe_proc</td>
<td>176</td>
</tr>
<tr>
<td>ps()</td>
<td>nmrglue.process.proc_base</td>
<td>118</td>
</tr>
<tr>
<td>put_axisheader()</td>
<td>nmrglue.fileio.sparky</td>
<td>78</td>
</tr>
<tr>
<td>put_block()</td>
<td>nmrglue.fileio.varian</td>
<td>100</td>
</tr>
<tr>
<td>put_blockheader()</td>
<td>nmrglue.fileio.varian</td>
<td>101</td>
</tr>
<tr>
<td>put_data()</td>
<td>nmrglue.fileio.bruker</td>
<td>27</td>
</tr>
<tr>
<td>put_data()</td>
<td>nmrglue.fileio.pipe</td>
<td>51</td>
</tr>
<tr>
<td>put_data()</td>
<td>nmrglue.fileio.sparky</td>
<td>75</td>
</tr>
<tr>
<td>put_fileheader()</td>
<td>nmrglue.fileio.sparky</td>
<td>50</td>
</tr>
<tr>
<td>put_fileheader()</td>
<td>nmrglue.fileio.varian</td>
<td>78</td>
</tr>
<tr>
<td>put_fileheader()</td>
<td>nmrglue.fileio.varian</td>
<td>101</td>
</tr>
<tr>
<td>put_hyperheader()</td>
<td>nmrglue.fileio.varian</td>
<td>101</td>
</tr>
<tr>
<td>put_tile()</td>
<td>nmrglue.fileio.sparky</td>
<td>75</td>
</tr>
<tr>
<td>put_trace()</td>
<td>nmrglue.fileio.pipe</td>
<td>50</td>
</tr>
<tr>
<td>put_trace()</td>
<td>nmrglue.fileio.rnmrtk</td>
<td>63</td>
</tr>
<tr>
<td>put_trace()</td>
<td>nmrglue.fileio.varian</td>
<td>101</td>
</tr>
<tr>
<td>pvoigt_fwhm (class in nmrglue.analysis.linesh)</td>
<td></td>
<td>220</td>
</tr>
<tr>
<td>qart()</td>
<td>nmrglue.process.pipe_proc</td>
<td>192</td>
</tr>
<tr>
<td>qart()</td>
<td>nmrglue.process.proc_base</td>
<td>141</td>
</tr>
<tr>
<td>qart_auto()</td>
<td>nmrglue.process.proc_base</td>
<td>141</td>
</tr>
<tr>
<td>qmix()</td>
<td>nmrglue.process.pipe_proc</td>
<td>141</td>
</tr>
<tr>
<td>qmix()</td>
<td>nmrglue.process.proc_base</td>
<td>193</td>
</tr>
<tr>
<td>range_flt()</td>
<td>nmrglue.process.proc_base</td>
<td>145</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.bruker</td>
<td>16</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.jcampdx</td>
<td>39</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.nmrml</td>
<td>40</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.pipe</td>
<td>41</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.nmrmtk</td>
<td>57</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.simpson</td>
<td>66</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.sparky</td>
<td>69</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.table</td>
<td>80</td>
</tr>
<tr>
<td>read()</td>
<td>nmrglue.fileio.tecmag</td>
<td>85</td>
</tr>
<tr>
<td>read_1D()</td>
<td>nmrglue.fileio.pipe</td>
<td>47</td>
</tr>
<tr>
<td>read_2D()</td>
<td>nmrglue.fileio.pipe</td>
<td>48</td>
</tr>
<tr>
<td>read_2D()</td>
<td>nmrglue.fileio.sparky</td>
<td>73</td>
</tr>
<tr>
<td>read_3D()</td>
<td>nmrglue.fileio.pipe</td>
<td>48</td>
</tr>
<tr>
<td>read_3D()</td>
<td>nmrglue.fileio.sparky</td>
<td>74</td>
</tr>
<tr>
<td>read_4D()</td>
<td>nmrglue.fileio.pipe</td>
<td>48</td>
</tr>
<tr>
<td>read_acquis_file()</td>
<td>nmrglue.fileio.bruker</td>
<td>28</td>
</tr>
<tr>
<td>read_binary()</td>
<td>nmrglue.fileio.bruker</td>
<td>20</td>
</tr>
<tr>
<td>read_binary()</td>
<td>nmrglue.fileio.sparky</td>
<td>68</td>
</tr>
<tr>
<td>read_binary_lowmem()</td>
<td>nmrglue.fileio.bruker</td>
<td>22</td>
</tr>
<tr>
<td>read_fid()</td>
<td>nmrglue.fileio.varian</td>
<td>90</td>
</tr>
<tr>
<td>read_fid_lowmem()</td>
<td>nmrglue.fileio.varian</td>
<td>91</td>
</tr>
<tr>
<td>read_fid_ntraces()</td>
<td>nmrglue.fileio.varian</td>
<td>92</td>
</tr>
<tr>
<td>read_jcamp()</td>
<td>nmrglue.fileio.bruker</td>
<td>22</td>
</tr>
<tr>
<td>read_lowmem()</td>
<td>nmrglue.fileio.bruker</td>
<td>19</td>
</tr>
<tr>
<td>read_lowmem()</td>
<td>nmrglue.fileio.pipe</td>
<td>42</td>
</tr>
<tr>
<td>read_lowmem()</td>
<td>nmrglue.fileio.sparky</td>
<td>59</td>
</tr>
<tr>
<td>read_lowmem()</td>
<td>nmrglue.fileio.varian</td>
<td>89</td>
</tr>
<tr>
<td>read_lowmem_2D()</td>
<td>nmrglue.fileio.pipe</td>
<td>48</td>
</tr>
<tr>
<td>read_lowmem_2D()</td>
<td>nmrglue.fileio.sparky</td>
<td>74</td>
</tr>
<tr>
<td>read_lowmem_3D()</td>
<td>nmrglue.fileio.pipe</td>
<td>48</td>
</tr>
<tr>
<td>read_lowmem_3D()</td>
<td>nmrglue.fileio.sparky</td>
<td>74</td>
</tr>
<tr>
<td>read_lowmem_4D()</td>
<td>nmrglue.fileio.pipe</td>
<td>49</td>
</tr>
<tr>
<td>read_lowmem_stream()</td>
<td>nmrglue.fileio.pipe</td>
<td>48</td>
</tr>
<tr>
<td>read_par()</td>
<td>nmrglue.fileio.rnmrtk</td>
<td>60</td>
</tr>
<tr>
<td>read_pdata()</td>
<td>nmrglue.fileio.bruker</td>
<td>18</td>
</tr>
<tr>
<td>read_pdata_binary()</td>
<td>nmrglue.fileio.bruker</td>
<td>21</td>
</tr>
<tr>
<td>read_pprog()</td>
<td>nmrglue.fileio.bruker</td>
<td>23</td>
</tr>
<tr>
<td>read_pprog()</td>
<td>nmrglue.fileio.varian</td>
<td>93</td>
</tr>
<tr>
<td>read_procpar()</td>
<td>nmrglue.fileio.varian</td>
<td>29</td>
</tr>
<tr>
<td>read_raw_bin_1D()</td>
<td>nmrglue.fileio.sparky</td>
<td>67</td>
</tr>
<tr>
<td>read_raw_bin_2d()</td>
<td>nmrglue.fileio.sparky</td>
<td>67</td>
</tr>
<tr>
<td>read_sec()</td>
<td>nmrglue.fileio.rnmrtk</td>
<td>59</td>
</tr>
<tr>
<td>read_stream()</td>
<td>nmrglue.fileio.pipe</td>
<td>48</td>
</tr>
<tr>
<td>read_table()</td>
<td>nmrglue.fileio.pipe</td>
<td>43</td>
</tr>
<tr>
<td>read_text()</td>
<td>nmrglue.fileio.sparky</td>
<td>67</td>
</tr>
<tr>
<td>read_xreim()</td>
<td>nmrglue.fileio.sparky</td>
<td>67</td>
</tr>
<tr>
<td>read_xreim()</td>
<td>nmrglue.fileio.varian</td>
<td>67</td>
</tr>
</tbody>
</table>
remove_digital_filter() (in module nmrglue.fileio.bruker), 19
reorder_columns() (in module nmrglue.fileio.table), 83
reorder_data() (in module nmrglue.fileio.bruker), 96
reorder_rows() (in module nmrglue.fileio.table), 82
reorder_submatrix() (in module nmrglue.fileio.varian), 27
repack_blockheader() (in module nmrglue.fileio.varian), 102
repack_fileheader() (in module nmrglue.fileio.varian), 103
repack_hyperheader() (in module nmrglue.fileio.varian), 102
reshape_data() (in module nmrglue.fileio.pipe), 52
rev() (in module nmrglue.process.pipe_proc), 186
reverse_filter() (in module nmrglue.process.proc_lp), 160
rft() (in module nmrglue.process.pipe_proc), 174
rft() (in module nmrglue.process.proc_base), 114
ri2c() (in module nmrglue.process.proc_base), 129
ri2rr() (in module nmrglue.process.proc_base), 130
rm_dig_filter() (in module nmrglue.fileio.bruker), 27
rnmrtk_nd (class in nmrglue.fileio.rnmrtk), 64
roll() (in module nmrglue.process.proc_base), 113
root2damp() (in module nmrglue.process.proc_base), 130
root2freq() (in module nmrglue.process.proc_base), 157
rotate_lr() (in module nmrglue.process.proc_base), 130
rr2ri() (in module nmrglue.process.proc_base), 130
rs() (in module nmrglue.process.proc_base), 171
rs() (in module nmrglue.process.pipe_proc), 171
sim_gauss_fwhm() (in module nmrglue.analysis.lineshapes1d), 212
sim_gauss_sigma() (in module nmrglue.analysis.lineshapes1d), 214
sim_lorentz_fwhm() (in module nmrglue.analysis.lineshapes1d), 213
sim_lorentz_gammat() (in module nmrglue.analysis.lineshapes1d), 215
sim_NDRegion() (in module nmrglue.analysis.linesh), 224
sim_pvoigt_fwhm() (in module nmrglue.analysis.lineshapes1d), 214
sim_voigt_fwhm() (in module nmrglue.analysis.lineshapes1d), 213
sim_voigt_sigmagamma() (in module nmrglue.analysis.lineshapes1d), 215
sine() (in module nmrglue.analysis.lineshapes1d), 168
sine() (in module nmrglue.process.proc_base), 112
size_and_ndtofrom_iter() (in module nmrglue.fileio.fileiobase), 36
slice2limits() (in module nmrglue.analysis.analysisbase), 202
smo() (in module nmrglue.process.proc_pipe), 194
smo() (in module nmrglue.process.proc_base), 142
sol() (in module nmrglue.process.proc_pipe), 180
sol_boxcar() (in module nmrglue.process.proc_bl), 148
sol_gaussian() (in module nmrglue.process.proc_bl), 149
sol_general() (in module nmrglue.process.proc_bl), 148
sol_sine() (in module nmrglue.process.proc_bl), 149
sol_sine2() (in module nmrglue.process.proc_bl), 149
sp() (in module nmrglue.process.proc_pipe), 167
sp() (in module nmrglue.process.proc_base), 109
sparky_2d (class in nmrglue.fileio.sparky), 79
sparky_3d (class in nmrglue.fileio.sparky), 79
split_list() (in module nmrglue.analysis.linesh), 227
sum_flt() (in module nmrglue.process.proc_base), 145
super_zip() (in module nmrglue.analysis.helpers), 207
swap() (in module nmrglue.process.proc_base), 130
swap_Ri() (in module nmrglue.process.proc_base), 131
thres() (in module nmrglue.process.proc_base), 134
tile_data2d() (in module nmrglue.fileio.sparky), 76
tile_data3d() (in module nmrglue.fileio.sparky), 77
tm() (in module nmrglue.process.proc_pipe), 169
tm() (in module nmrglue.process.proc_base), 111
to Bruker() (nmrglue.fileio.conv.converter method), 32
to_pipe() (nmrglue.fileio.conv.converter method), 32
to_rnmrtk() (nmrglue.fileio.conv.converter method), 32
to_sparky() (nmrglue.fileio.conv.converter method), 32
to_universal() (nmrglue.fileio.convert.converter method), 33

to_varian() (nmrglue.fileio.convert.converter method), 33
torder22ti() (in module nmrglue.fileio.varian), 96
torder22ti() (in module nmrglue.fileio.varian), 96
tp() (in module nmrglue.process.pipe_proc), 177
tp() (in module nmrglue.process.proc_base), 119
tp_hyper() (in module nmrglue.process.proc_base), 120
trace2index_flat() (in module nmrglue.fileio.fileiobase), 36
trace2index_op() (in module nmrglue.fileio.fileiobase), 36
trace2index_reg() (in module nmrglue.fileio.fileiobase), 36
transpose_3D() (in module nmrglue.fileio.pipe), 51
tri() (in module nmrglue.process.pipe_proc), 169
tri() (in module nmrglue.process.proc_base), 111

U
uc_from_frequscale() (in module nmrglue.fileio.fileiobase), 36
uc_from_udic() (in module nmrglue.fileio.fileiobase), 35
udata_nd (class in nmrglue.fileio.convert), 33
unappend_data() (in module nmrglue.fileio.sparky), 52
unappend_data() (in module nmrglue.fileio.simpson), 68
unappend_imag() (in module nmrglue.process.proc_base), 130
uncomplexify_data() (in module nmrglue.fileio bruker), 27
uninterleave_data() (in module nmrglue.fileio.rnmrtk), 63
uninterleave_data() (in module nmrglue.fileio.varian), 103
unit_conversion (class in nmrglue.fileio.fileiobase), 37
unpack_complex() (in module nmrglue.process.proc_base), 129
unpack_table() (in module nmrglue.analysis.linesh), 225
unshape_data() (in module nmrglue.fileio.pipe), 52
untile_data2D() (in module nmrglue.fileio.sparky), 76
untile_data3D() (in module nmrglue.fileio.sparky), 77

V
valid_pt() (in module nmrglue.analysis.analysisbase), 201
voigt_fwhm (class in nmrglue.analysis.lineshapes1d), 219
voigt_sigmagamma (class in nmrglue.analysis.lineshapes1d), 219

W
write() (in module nmrglue.fileio.bruker), 17
write() (in module nmrglue.fileio.pipe), 41
write() (in module nmrglue.fileio.rnmrtk), 58
write() (in module nmrglue.fileio.sparky), 69
write() (in module nmrglue.fileio.table), 81
write() (in module nmrglue.fileio.varian), 88
write_2D() (in module nmrglue.fileio.sparky), 74
write_3D() (in module nmrglue.fileio.pipe), 49
write_3D() (in module nmrglue.fileio.sparky), 74
write_4D() (in module nmrglue.fileio.pipe), 49
write_binary() (in module nmrglue.fileio.brucker), 21
write_binary_lowmem() (in module nmrglue.fileio.brucker), 22
write_fid() (in module nmrglue.fileio.varian), 90
write_fid_lowmem() (in module nmrglue.fileio.varian), 92
write_jcamp() (in module nmrglue.fileio.bruker), 23
write_jcamp_pair() (in module nmrglue.fileio.bruker), 28
write_lowmem() (in module nmrglue.fileio.sparky), 70
write_lowmem() (in module nmrglue.fileio.varian), 88
write_lowmem_2D() (in module nmrglue.fileio.pipe), 49
write_lowmem_3D() (in module nmrglue.fileio.pipe), 50
write_lowmem_3Ds() (in module nmrglue.fileio.pipe), 50
write_lowmem_4D() (in module nmrglue.fileio.pipe), 50
write_lowmem_4Ds() (in module nmrglue.fileio.pipe), 50
write_part() (in module nmrglue.fileio.rnmrtk), 60
write_pprog() (in module nmrglue.fileio.bruker), 24
write_procpar() (in module nmrglue.fileio.varian), 93
write_sec() (in module nmrglue.fileio.rnmrtk), 60
write_slice_3D() (in module nmrglue.fileio.pipe), 51
write_table() (in module nmrglue.fileio.pipe), 43

X
xy2yx() (in module nmrglue.process.proc_base), 120
xyz2zyx() (in module nmrglue.process.pipe_proc), 199

Y
ytp() (in module nmrglue.process.proc_base), 120

Z
zd() (in module nmrglue.process.pipe_proc), 194
zd() (in module nmrglue.process.proc_base), 142
zd_boxcar() (in module nmrglue.process.proc_base), 143
zd_gaussian() (in module nmrglue.process.proc_base), 144
zd_sinebell() (in module nmrglue.process.proc_base), 143
zd_triangle() (in module nmrglue.process.proc_base), 143
zf() (in module nmrglue.process.pipe_proc), 177
zf() (in module nmrglue.process.proc_base), 121
zf_auto() (in module nmrglue.process.proc_base), 122
zf_double() (in module nmrglue.process.proc_base), 121
zf_inter() (in module nmrglue.process.proc_base), 120
zf_pad() (in module nmrglue.process.proc_base), 121
zf_size() (in module nmrglue.process.proc_base), 122
ztp() (in module nmrglue.process.pipe_proc), 199