J Biomol NMR (2013) 55:355–367
DOI 10.1007/s10858-013-9718-x

ARTICLE

Nmrglue: an open source Python package for the analysis of multidimensional NMR data

Jonathan J. Helmus · Christopher P. Jaroniec

Received: 18 December 2012 / Accepted: 19 February 2013 / Published online: 2 March 2013

Abstract Nmrglue, an open source Python package for working with multidimensional NMR data, is described. When used in combination with other Python scientific libraries, nmrglue provides a highly flexible and robust environment for spectral processing, analysis and visualization and includes a number of common utilities such as linear prediction, peak picking and lineshape fitting. The package also enables existing NMR software programs to be readily tied together, currently facilitating the reading, writing and conversion of data stored in Bruker, Agilent/Varian, NMRPipe, Sparky, SIMPSON, and Rowland NMR Toolkit file formats. In addition to standard applications, the versatility offered by nmrglue makes the package particularly suitable for tasks that include manipulating raw spectrometer data files, automated quantitative analysis of multidimensional NMR spectra with irregular lineshapes such as those frequently encountered in the context of biomacromolecular solid-state NMR, and rapid implementation and development of unconventional data processing methods such as covariance NMR and other non-Fourier approaches. Detailed documentation, install files and source code for nmrglue are freely available at http://nmrglue.com. The source code can be redistributed and modified under the New BSD license.

Keywords Nuclear magnetic resonance · Solid-state NMR · Data processing · Data analysis · Data visualization · Python · Open source

Introduction

Nuclear magnetic resonance (NMR) spectroscopy has become an indispensable tool for the detailed analysis of biological macromolecules (Wüthrich 2003) and has also been applied toward imaging (Lauterbur 2005), drug discovery (Shuker et al. 1996; Pellecchia et al. 2008), and metabolomics (Nicholson et al. 1999). NMR spectra contain a tremendous amount of information on the structure and dynamics of the molecule under investigation, but oftentimes the extraction of this information can be a complicated and, at times, highly computationally demanding process. To address these challenges, a growing collection of software, aided by the increasing computational power of personal computers, has been developed to record, process, analyze, and visualize NMR data (Smith et al. 1994; Delaglio et al. 1995; Pons et al. 1996; Hoch and Stern 1996; Günther et al. 2000; Bak et al. 2000; Blanton 2003; Keller 2004; Veshtort and Griffin 2006; van Beek 2007; Goddard and Kneller 2008; Lewis et al. 2009; Short et al. 2011; Nowling et al. 2011; Stevens et al. 2011). This proliferation of NMR software can act as both a benefit and a burden to the practitioners of the technique. The benefit comes from the fact that users can often locate existing software that will adequately address a

Electronic supplementary material The online version of this article (doi:10.1007/s10858-013-9718-x) contains supplementary material, which is available to authorized users.

J. J. Helmus (✉) · C. P. Jaroniec (✉)
Department of Chemistry and Biochemistry, The Ohio State University, 100 West 18th Avenue, Columbus, OH 43210, USA
e-mail: jjhelmus@gmail.com

C. P. Jaroniec
e-mail: jaroniec@chemistry.ohio-state.edu

Present Address:
J. J. Helmus
Environmental Science Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, USA

specific task they wish to accomplish. The downside is that, as the number of programs used to work with data generated in a typical experiment or set of experiments grows, new methods must be devised to allow all of these programs to effectively interact and exchange information; this problem is amplified by the lack of a standard format for storing NMR data.

For example, when performing an NMR structural analysis of a protein, it is not uncommon to collect the multidimensional time-domain NMR data using software provided by the spectrometer vendor, process the data using a second program, assign the various spectra using a third, extract structural restraints from peak positions and intensities with a fourth, and, finally, perform molecular dynamics simulations subject to these restraints using a fifth piece of software. Between each of these steps, the peak lists, restraint tables, and other spectral data must be converted from the format which was generated by the program or programs used in the preceding step into the format required for the next. Depending on the types of software used for the tasks above, this conversion must frequently be accomplished by auxiliary scripts and short software programs written within individual research groups and seldom shared with the scientific community at large. Although recent software packages such as the CCPN NMR FormatConverter (Vranken et al. 2005), the WeNMR portal (Wassenaar et al. 2012) and the CONNJUR Spectrum Translator (Nowling et al. 2011) provide capabilities for converting between a number of NMR formats they do not offer a complete solution to this conversion problem. The FormatConverter and the WeNMR portal are able to handle peak lists, assignment tables and other similar data but do not provide functionality to convert the raw numerical spectral data between different file formats. The CONNJUR Spectrum Translator introduces this functionality but is currently limited to relatively few file formats.

Here we describe nmrglue, an open source package for working with NMR data in the Python programming language. Nmrglue is designed to enable the various software programs that make up an NMR experiment data workflow to be seamlessly connected together. In addition to providing resources for handling raw spectrometer data files, the package furnishes a flexible and robust environment for rapidly implementing as well as developing new methods to process, analyze and visualize multidimensional NMR data sets, including spectra with irregular lineshapes such as those frequently encountered in the context of biomacromolecular solid-state NMR. The design of nmrglue and how it compares to existing NMR software is discussed, followed by a detailed description of the various features of the package. Finally, a number of examples where nmrglue is used to manipulate NMR data are presented.

Software design

Nmrglue aims to serve as a powerful, yet easy to install and use, platform for the facile implementation of NMR data processing, analysis and visualization methods, and one that also permits existing NMR software to be readily interconnected. To achieve these goals, nmrglue is based on the following design approach: rather than creating a new environment for handling NMR data the package takes as its input data stored in any one of a number of different file formats and provides these data as a multidimensional array object that can be further manipulated in the Python programming language. This approach is analogous to that taken by matNMR (Van Beek 2007), which works with NMR data within the MATLAB (MathWorks, Natick, MA) environment. The macro language used by NMRPipe (Delaglio et al. 1995) also provides a similar environment in Tcl/Tk, although this software enables only one-dimensional (1D) slices of NMR data to be manipulated one-at-a-time. A number of other NMR software packages, such as CARA (Keller 2004), incorporate the ability to automate tasks through a macro language or other scripting facilities. However, in most cases scripting is not the primary interface but rather a method of automating functions more typically performed through a graphical user interface (GUI). Finally, the capabilities of many of the existing NMR software packages are difficult to extend readily due to their architecture and lack of access to the source code.

Python is a general purpose, high-level, interpreted programming language (Van Rossum 1995), whose clear, readable syntax and remarkable power have resulted in it becoming one of the most widely used scripting languages, especially for scientific applications. Python is open source software, which can be readily downloaded, installed, and used on computers running the Windows, Linux or OS X operating systems free of charge. The language is considered to be relatively straightforward to learn and use with ample documentation available both online (docs.python.org) and in print (Lutz 2011). Python source code can be executed in script format or interactively within a Python shell allowing new scripts to be readily developed and tested. Python's standard library is extensive, providing tools for a number of common computational tasks. Moreover, the language can be extended with custom modules. Of interest to the scientific community in general and the NMR community in particular is the NumPy module (Oliphant 2007), which adds support for the efficient handling of multidimensional arrays and provides a large library of mathematical functions that operate on these arrays at speeds comparable to compiled programming languages. The SciPy (Jones et al. 2001) and matplotlib (Hunter 2007) libraries, respectively, contain additional mathematical tools for scientific computing and creating high quality plots within Python. Collectively, these open source modules give

Python functionality that is similar to MATLAB and other numerical computing environments. Recently, a number of projects have commenced that harness the power and flexibility of these computational tools to address a diverse collection of problems in a wide variety of fields including astronomy (Turk et al. 2011), bioinformatics (Cock et al. 2009), machine learning (Pedregosa et al. 2011), neuroimaging (Gorgolewski et al. 2011) and statistics (Seabold and Perktold 2010). It is also noteworthy that Python, NumPy, SciPy, matplotlib, any many other scientific Python packages are all distributed under open source licenses, which allows other open source software and, in certain cases, non-open source or even commercial software to distribute and reuse code from these packages free of charge as long as appropriate copyright and license requirements are met.

Nmrglue draws on these powerful open source Python libraries to create an environment for working with NMR data. Data stored in a number of common file formats are made available by nmrglue as NumPy multidimensional arrays; this provides a robust method for storing spectral data during processing and analysis. These data can then be output to various file formats for storage and readily manipulated using mathematical routines in the NumPy or SciPy libraries. Nmrglue provides a number of common NMR processing routines built using the fast and efficient routines contained in these libraries. In addition, since the nmrglue source code is freely available under a permissive license the code for these routines can serve as an example and be easily adapted or extended for new uses, allowing novel data processing or analysis methods to be designed and tested by leveraging the linear algebra and numeric algorithms that already exist in the different scientific Python libraries. NMR data accessed in nmrglue can be visualized using matplotlib (or another Python plotting library), which can also be employed to create interactive plots within GUI applications or publication quality figures.

In conjunction with these comprehensive scientific libraries, nmrglue provides an environment for rapid prototyping and testing of new NMR data processing and analysis methods. New or existing C, C++ or Fortran code can also be interfaced as extension modules, which operate on NumPy arrays, using tools such as SWIG (Beazley 2003), F2PY (Peterson 2009) or Cython (Behnel et al. 2011). Finally, nmrglue can be used from within a Python shell, or an enhanced Python shell such as IPython (Perez and Granger 2007), to interactively examine, process and analyze NMR data.

Software features

In this section, we discuss the main features of nmrglue. The major modules which make up nmrglue and their functionalities are listed in Table 1, and a more detailed listing of the various package components is available online at the nmrglue website, http://nmrglue.com. Throughout the paper the names of nmrglue modules are indicated in **bold** font, function names in *italic* font and references to specific sections of source code or commands to be entered into a shell in `fixed width` font.

Table 1 A listing of the major modules making up the nmrglue package

Module	Description
Fileio	
bruker	Reading and writing of Bruker files
pipe	Reading and writing of NMRPipe files
rnmrtk	Reading and writing of Rowland NMR Toolkit files
simpson	Reading of files created by the SIMPSON simulation program
sparky	Reading and writing of Sparky files
varian	Reading and writing of Agilent/Varian files
convert	Conversion between any of the above formats
Processing	
proc_base	Common NMR processing functions (apodization, shifts, transforms, etc.)
proc_bl	Baseline filtering, smoothing and correcting functions
proc_lp	Linear prediction modeling and extrapolation
pipe_proc	Processing functions with names and parameters similar to those in NMRPipe
Analysis	
peakpick	Numerous peak picking algorithms which work in arbitrary dimensions
linesh	Fitting and simulation of arbitrary dimensional lineshapes

Reading, writing and converting between common file formats

First and foremost, nmrglue can be used to read and write data from and to a number of common NMR file formats. At the present time, nmrglue can access and save data in formats recorded on Bruker and Agilent/Varian spectrometers, as well as files used by NMRPipe (Delaglio et al. 1995), Sparky (Goddard and Kneller 2008), SIMPSON (Bak et al. 2000), and the Rowland NMR Toolkit (Hoch and Stern 1996). Importantly, support for additional data formats can be readily implemented owing to the flexibility of the platform and is planned for future releases of nmrglue. Using the *read* function appropriate for the particular file format, nmrglue returns to the user the NMR data and any spectral parameters or other meta-data contained in the file. One- or multidimensional NMR data are read into memory and contained in an ndarray, a robust multidimensional array object provided by NumPy with which the scientific routines in the NumPy and SciPy libraries can

subsequently interact. The dimensionality, size, direct dimension quadrature, and data type (integer or floating point values) of this object are set and appropriately converted by nmrglue, based on parameters contained in the file being read. Since NumPy provides no mechanism for the representation of hypercomplex data (States et al. 1982; Delsuc 1988), quadrature in the indirect dimensions must be manipulated as required when the data are transposed. Any meta-data and spectral parameters present in the file, such as the carrier frequencies and sweep widths, are stored in a dictionary using key-value pairs and returned to the user. Since each NMR file format contains different numbers and formatting styles of the spectral parameters, the structure and size of these dictionaries are unique to each file format. Nmrglue has a limited ability to convert between the parameter dictionaries corresponding to different file formats as discussed in more detail below.

Oftentimes it is neither necessary nor desirable to read an entire NMR data set into memory, especially when working with large multi-gigabyte 3D or 4D data sets. In many cases, only a small subset of the data is actually required at a given time, and loading the entire data set into memory can be a time and resource consuming process (or, indeed, impossible in cases where memory is limited). For instance, when visualizing a 3D spectrum as a series of 2D planes, only single 2D slices along a given frequency axis need to be accessed at any given time. Nmrglue provides methods for loading limited regions of NMR data into memory only when the data are needed to perform a calculation. This is accomplished via the *read_lowmem* functions, available for most file formats. The resulting array-like object which is returned behaves similarly to a NumPy ndarray object. It can be transposed, but no data are actually loaded into memory until a specific region of the data set is requested through a slicing operation, at which point the requested data are read and returned as a NumPy array.

In addition to being able to read NMR data and parameters in a number of file formats, nmrglue can write to most of the supported file formats using *write* functions present in the modules. In the case of large data sets, for which the low memory reading functions are employed, users can request a region of the data and write that region to disk using the *write* function or write the entire data set to a file trace by trace using the *write_lowmem* function. By using this functionality, only a single 1D data trace must be stored in memory at any given moment. Additional methods, which allow for writing to specific regions of a file, are under development and will be available in the future versions of the package.

Nmrglue can also be used to convert data between different formats. The **convert** module provides tools for this functionality. An example of conversion from Sparky to NMRPipe file format is given in Listing S1 available in the Supporting Information. As demonstrated by this example, the mechanism for the conversion is to read in the NMR data and parameters (line 4), create and load a conversion object (lines 7 and 8), and request the NMR data and parameters in the output format (line 9) which are then written to disk (line 12). Any manipulation of the data necessary for the conversion (sign changes, data type modifications, etc.) is performed internally so that the resulting array is correct for the requested output format. By using the *read_lowmem* and *write_lowmem* functions, large NMR data sets can be converted in this manner from one format to another in such a way that only a single trace is stored in memory at any given time.

Given that the supported file formats store differing numbers of spectral parameters, many of which are not specifically linked to a given dimension, it is not always possible to determine all of the parameters required for a particular format. During the conversion, nmrglue will attempt to find the correct parameters in the initial parameter dictionary or will fill in default values if this information is not available. Internally nmrglue uses a "universal dictionary" for these conversions, which contains the most fundamental parameters necessary to describe NMR data. Users can fill in corrected values of the various parameters by updating the resulting dictionary or by providing a custom universal dictionary when loading in the data. An example of this procedure is given in Listing S8, which will be discussed in detail in the Example applications section. A more advanced file conversion system that is focused on ease of use is currently under development and will be included in future versions of the software.

Often locations in NMR spectra are referenced not by points, but in more convenient units such as Hz or ppm. To allow the use of these units, the file input/output modules contain functions for creating unit conversion objects, which convert to and from points in a given dimension to more common unit types including Hz, ppm, percent, microseconds, milliseconds, and seconds. The use of these objects to extract and display a spectrum in units of ppm will be shown in the Example applications section.

Data processing

In addition to providing the capabilities to read, write, and convert between various NMR file formats, nmrglue also contains a number of modules for processing NMR data. Since most of the data processing methods used in NMR spectroscopy are identical or highly analogous to signal processing techniques used in other scientific fields, the numerical routines in the NumPy and SciPy libraries provide efficient implementations of these algorithms. The

proc_base module contains a number of common NMR processing functions including apodizations, spectral shifts, transforms, and filters. The **proc_bl** module provides functions for filtering, smoothing and flattening spectral baselines, and the **proc_lp** module has functions for linear prediction (Ni and Scheraga 1986), modeling and extrapolation using different algorithms including singular value decomposition (SVD) and total least squares (TLS). Given the widespread use and popularity of the NMRPipe software in the NMR community, the **pipe_proc** module has also been created to provide processing functions with names and parameters similar to their NMRPipe counterparts. In addition, the functions in this module also update the dictionary of spectral parameters as the data are processed, so that the file written after processing is nearly identical to files processed using NMRPipe. Overall, the simplicity and code readability of the nmrglue processing modules—which, together with the open distribution of the source code provide developers with example implementations of many common NMR processing methods—comes at some expense of speed. However, the reduced speed of these functions relative to highly optimized and specialized code is typically not a major hindrance, as Fourier transforms and other operations employed in NMR data processing are no longer time intensive tasks for modern computers. For example, the processing of a test 1500* × 166* 2D NMR data matrix with zero-filling to 4096* and 2048* points, respectively, which required 2.04 s using NMRPipe on a Dell Precision Workstation 470 with two Intel Xeon processors and 6 GB of RAM, could be achieved in 5.85 s using nmrglue. This difference in processing times can be partially attributed to nmrglue's limitation of running on a single processor that is not present in NMRPipe.

Spectral analysis

Extracting the wealth of information present in multidimensional NMR spectra can be a time and labor intensive task. In biomolecular NMR this analysis is oftentimes highly repetitive, with each residue in the protein giving rise to one or more cross-peaks which are analyzed in a similar manner. The process thus lends itself to streamlining and automation. Nmrglue contains basic analysis tools that can be used to create scripts and programs to tease out structural and dynamic information from NMR data. The **peakpick** module contains a function which can perform peak picking of NMR spectra of arbitrary dimensionality using several different algorithms. Estimation of peak positions and linewidths and the clustering of nearby peaks, which are all required for further analysis, can be performed within this module. One commonly used procedure involves fitting the experimental peaks to a model lineshape, and the **linesh** module enables such fitting of a peak or cluster of peaks to be performed in an arbitrary number of dimensions using the Levenberg–Marquardt algorithm (Marquardt 1963). Lorentzian, Gaussian, and Voigt lineshapes most commonly encountered in the analysis of NMR spectra are built into nmrglue, and users may also supply their own lineshape functions to be used in peak fitting.

In addition to the analysis capabilities described above, nmrglue can be used to rapidly develop new data analysis approaches. Such developments are aided by the underlying spectral segmentation and peak picking methods, a Levenberg–Marquardt least squares optimization algorithm which allows fitting parameters to be constrained and a spectral simulation function applicable to an arbitrary number of dimensions, as well as the extensive collections of fast and efficient scientific and computational routines available in NumPy and SciPy. Furthermore, C, C++ or Fortran code can be easily interfaced with Python using the C-API module in NumPy, F2PY, SWIG, or Cython, which allows the analysis routines written in these languages to be included in nmrglue scripts.

Interactivity

In addition to enabling the development of Python scripts that work with NMR data, nmrglue can also be used to examine NMR data in an interactive mode. This is most readily achieved using an enhanced Python shell, such as IPython, that allows data to be viewed in graphical form and includes a number of additional advanced features. Figure 1 shows a sample IPython notebook, in which nmrglue was used to examine and visualize 1D and 2D NMR data from within a web browser.

Installation and documentation

Nmrglue was designed to provide a powerful environment in which NMR data can be converted, processed and analyzed, with particular attention paid to ease-of-use and substantial flexibility that allows users to develop and test new analysis methods. With these goals in mind, creation of quality documentation was given a high priority. Consequently, all user-facing functions and classes in the package have been thoroughly documented and are available using Python's built-in help function or online at the nmrglue website at http://nmrglue.com. In addition, a tutorial introducing many of the features of the package, multiple examples of nmrglue scripts, as well as instructions for software installation, are also available at the website.

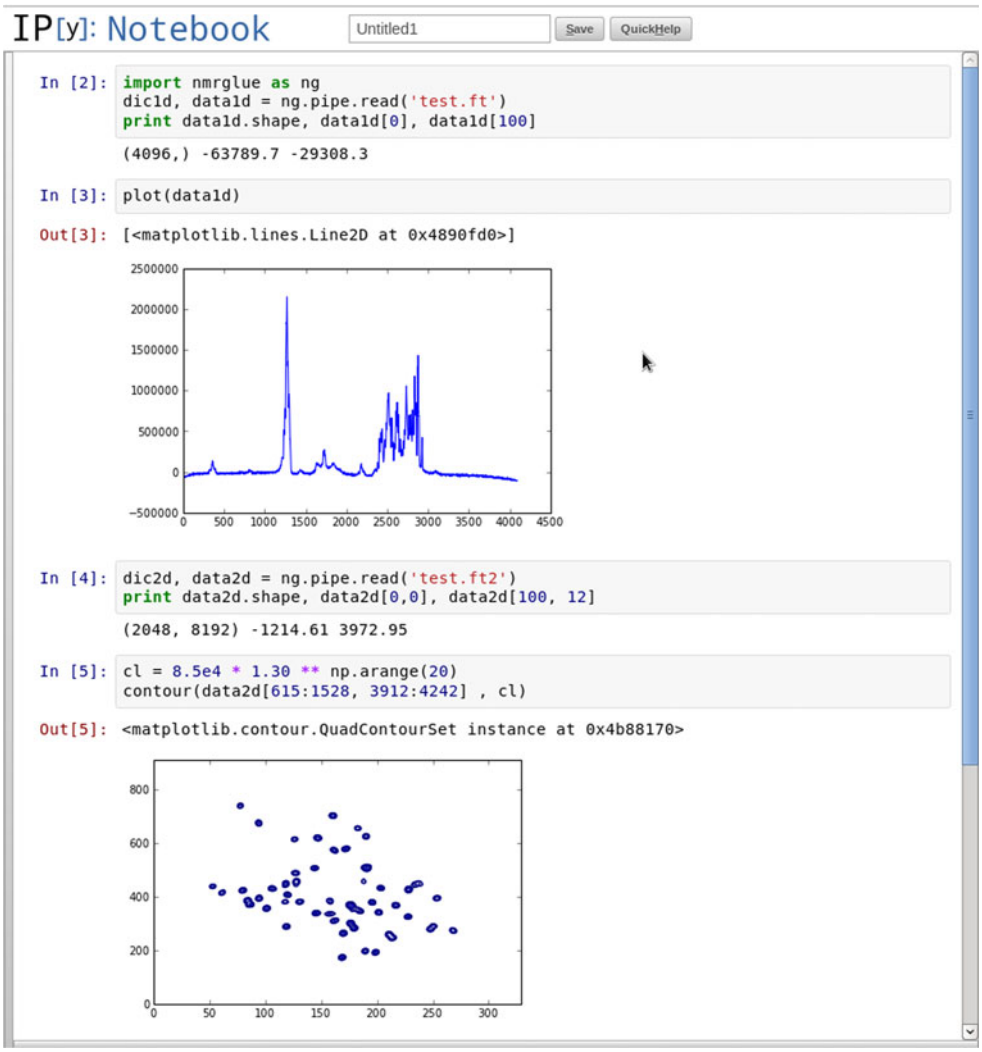

Fig. 1 A sample IPython notebook, in which nmrglue is used to interactively examine and visualize 1D and 2D NMR spectra

Example applications

In this section we discuss a selection of Python scripts, available as Listings S1-S15 in the Supporting Information, as examples of how nmrglue can be used to visualize, process, and analyze NMR data. All of the scripts and corresponding NMR data presented in this article as well as numerous additional examples are available for download from the nmrglue website, http://nmrglue.com. In order for the scripts to run, Python and the NumPy, SciPy and nmrglue libraries must be installed. For the data visualization examples, matplotlib must also be installed. All the scripts can be executed from the command line by issuing the command: `python script.py`, where `script.py` is the name of the script being executed. The files may also be executed directly; on Linux systems this is done by adding `#!/usr/bin/env python` as the first line of the file, setting the appropriate executable mode and typing `script.py` on the command line.

Visualization of 1D time and frequency domain NMR data

Visualization of NMR spectra is an essential step in the assignment of the resonances and is helpful in evaluating various processing methods. Nmrglue can be used in conjunction with matplotlib or another Python plotting package to visualize NMR data stored in any of the supported formats. As an example, Listings S2 and S3 provide Python scripts which plot the free induction decay (Listing S2) and corresponding ^{13}C MAS solid-state NMR spectrum (Listing S3) recorded for a sample of amyloid fibrils formed from ^{13}C,^{15}N-enriched Y145Stop variant of the human prion protein (Helmus et al. 2008b, 2010, 2011). In both scripts the NMR data are read in from an NMRPipe file (line 5) and a unit conversion object is created (line 8), which is used in line 13 to provide a scale for the plot in milliseconds or ppm. Lines 11–19 plot the time and frequency domain spectra, set the scales of the axes and add

appropriate labels using the matplotlib library. The last line in each script creates the output files, "fid.eps" and "spectrum.eps", which are shown in Figs. 2 and 3, respectively. Features can be added to or removed from the figures by modifying the Python scripts or using image editing software, and other graphics output formats, including png and PDF, can be created by appropriately modifying the extension of the filename on the last line.

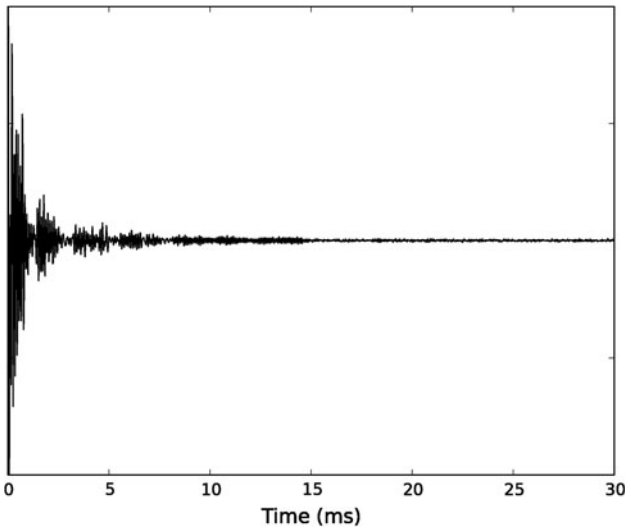

Fig. 2 The ^{13}C free induction decay for a cross-polarization (CP) magic-angle spinning (MAS) NMR experiment recorded for amyloid fibrils formed by the Y145Stop variant of human protein (Helmus et al. 2008b, 2010, 2011). The plot was created using the script in Listing S2

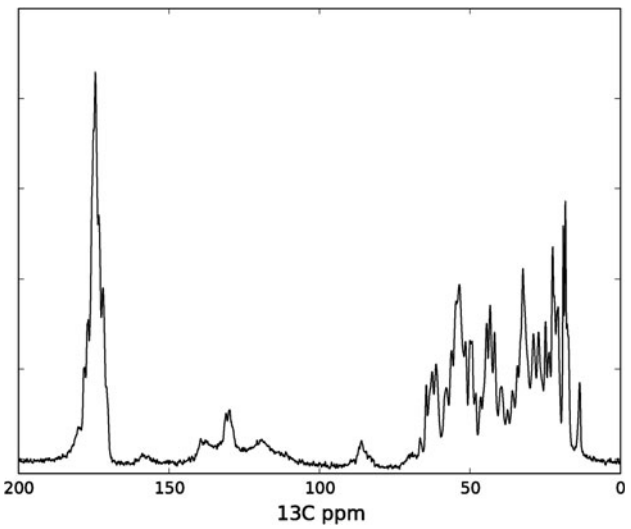

Fig. 3 1D ^{13}C CP MAS NMR spectrum for Y145Stop human prion protein amyloid fibrils created using the script in Listing S3

Visualization of 2D NMR spectra

2D NMR spectra can be plotted in a similar manner as shown in Listing S4 and accompanying Fig. 4. In this script, a 2D ^{15}N–^{13}C spectrum is read from an NMRPipe file (line 5), the limits of the ppm scales for both axes are determined from unit conversion objects (lines 8–11), and a contour plot of the spectrum is created (lines 14–18). Three representative 1D ^{13}C slices were also added to the plot in lines 21–27 to demonstrate the spectral resolution and signal-to-noise ratio, followed by setting the limits and labeling the frequency axes (lines 30–34).

Matplotlib and other Python plotting libraries can also be used to create plots that can be examined interactively. This is achieved by changing the last line of the Python scripts in Listings S2, S3 or S4 to `fig.show()`. Running the scripts with this modification will open an interactive window containing the plots in Figs. 2, 3 or 4. In this environment users can zoom, pan and save the plot, and even more advanced interactive environments can be created using these and other Python tools.

Separation of interleaved pseudo 3D NMR data sets

The next two examples, given in Listings S5 and S6, show how nmrglue can be used to prepare NMR data for use by other software. In both of these examples pseudo 3D data sets, recorded on an Agilent/Varian VNMRS-500 spectrometer and consisting of series of 2D chemical shift correlation spectra acquired in an interleaved fashion as a function of a relaxation or dipolar evolution delay, are separated into their constituent 2D's for processing with NMRPipe or another software package. These types of interleaved experiments, where the relaxation or dipolar

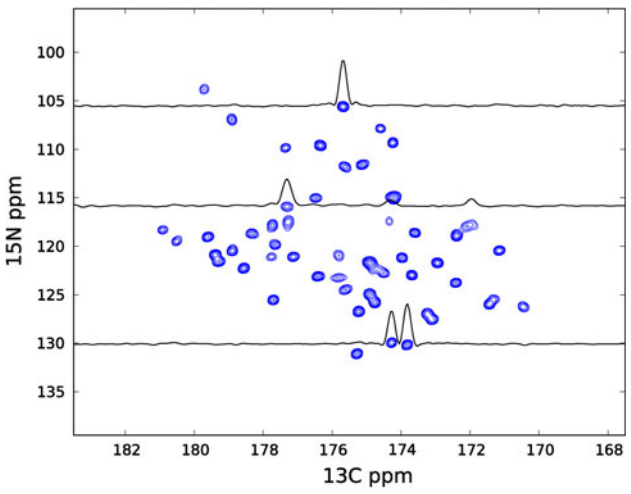

Fig. 4 A 2D ^{15}N–^{13}C spectrum of the K28C-EDTA-Cu^{2+} mutant of GB1 with representative 1D ^{13}C traces inset into the spectrum (Nadaud et al. 2010) created using the script in Listing S4

evolution period is incremented as the innermost loop, are very common in solution and solid-state NMR since the effects of long-term instrument instabilities on the extracted relaxation rates or dipolar coupling constants are minimized for data collected in this manner (as opposed to recording complete 2D's back-to-back with the relaxation or dipolar evolution period incremented as the outermost loop of the pseudo 3D data set). While other tools exist for separating these types of arrayed experiments, they often require the data to be collected in particular order and fail otherwise. By providing direct access to the NMR data, nmrglue can be readily used to create the required scripts that separate these types of interleaved experiments regardless of the order in which the parameters were arrayed.

In the first example (Listing S5), a relaxation experiment consisting of six 2D ^{15}N–^{13}C solid-state NMR spectra was recorded. The experiment was set up so that relaxation delay, labeled by the parameter "techo", was the innermost arrayed parameter, followed by the phase and then the delay for the indirect ^{15}N chemical shift dimension. The Python script separates these data set into six directories with names "techo_X.fid" where X is the value of the "techo" parameter. Each directory contains the corresponding 2D time-domain data in Agilent/Varian format for processing with NMRPipe or another software package. In the script, the interleaved data set is read using the *read* function of nmrglue's **varian** module (line 4). For data collected with the arrayed parameter as the innermost loop, the array containing the NMR data will automatically be shaped by nmrglue so that the next-to-last dimension varies with the arrayed parameter. The size of the individual 2D spectra will be smaller than the full interleaved pseudo 3D data set, so the parameter dictionary is updated with this new size on line 7. Line 10 begins a loop over the relaxation times "techo". Within each iteration of the loop, the appropriate region of the data set is extracted and written to the corresponding directory on line 13, which completes the script. Note, finally, that this script will work to separate other interleaved pseudo 3D NMR data sets, collected with the arrayed parameter as the innermost loop by modifying lines 10 and 11 to account for the name of the parameter being arrayed.

For interleaved experiments with a different order of the arrayed parameters the script in Listing S5 will fail, as will other programs that separate experiments collected in this manner. However, nmrglue provides the facilities for the rapid development of new scripts to deal with these cases. One such Python script, which separates a pseudo 3D data set consisting of a series of 2D z-filtered TEDOR spectra (Jaroniec et al. 2002) collected with the quadrature phase as the innermost loop, is given in Listing S6. In this case the number of applied TEDOR dipolar mixing cycles, set by the parameter "nredor", corresponds to the next-to-last loop. This script has a similar layout to Listing S5, with a few minor modifications as required by the difference in the data ordering. Specifically, the Agilent/Varian data are read on line 5 with the keyword `as_2d` assigned a value of `True` which results in the NMR data being returned as a two-dimensional array. Lines 8–10 calculate and set the new size of the outputted data. Line 13 begins the loop over the "nredor" values. For each iteration of the loop a directory name is created (line 14) and the traces which make up the individual 2D spectra are extracted and added in the correct order to the `sdata` array (lines 16–18), which are then written to disk (line 19).

Processing of 2D NMR spectra collected with a S^{3}E filter

Recently it has been demonstrated that the spin-state selective excitation (S^{3}E) approach (Meissner et al. 1997) can be used in the context of protein solid-state NMR to obtain high-resolution ^{15}N–^{13}C chemical shift correlation spectra for which the effects of ^{13}C–^{13}C J-coupling evolution during signal acquisition have been suppressed (Laage et al. 2009). We have employed this methodology for proteins containing covalently attached paramagnetic tags to rapidly collect 2D and 3D NMR spectra with high resolution and sensitivity (Nadaud et al. 2010, 2011; Sengupta et al. 2012). Processing data collected with a S^{3}E filter involves separating the 'A' and 'B' blocks of the experiment, recombining them as the sum and difference data sets, processing these two data sets independently, and finally coadding the two resulting spectra. While this type of processing can be carried out entirely with existing software, such as NMRPipe, creating the necessary processing scripts requires the use of NMRPipe functions that are not common in routine processing scripts. Nmrglue can be used to accomplish such data processing in a very straight forward manner as illustrated by the Python script in Listing S7. This script takes an Agilent/Varian 2D data set recorded with a S^{3}E filter, extracts the 'A' and 'B' blocks which were collected as the even and odd traces of the experiment (lines 4 and 5), and saves the sum and difference data sets as binary Agilent/Varian files (lines 6 and 7). These files can be converted and processed as conventional 2D ^{15}N–^{13}C NMR spectra using NMRPipe or other software, or processed further using nmrglue as demonstrated below.

Prior to processing the two data sets they are converted to NMRPipe files using the script in Listing S8. Here the Agilent/Varian "fid_sum" file is read (line 4), spectral parameters for the 2D spectra are recorded in the universal parameter dictionary (lines 8–14), a converter object is created and loaded with the Agilent/Varian data and the universal

dictionary (lines 17 and 18) and NMRPipe formatted data are requested (lines 19). Finally, these data are written to the "test_sum.fid" file. The entire process is then repeated for the difference data set (lines 25–29), but without setting the spectral parameters which are identical to the sum data set. Listing S9 provides a script in which the direct dimensions of the sum (lines 4–12) and difference (lines 15–24) data sets are processed using nmrglue's **pipe_proc** module. Note that the requisite 27.5 Hz shift of the spectra, corresponding to one half of the one-bond ^{13}CO–^{13}Cα J-coupling constant, is converted to points using a unit conversion object (lines 8–9). With the direct dimension processing completed, the two data sets can be coadded (lines 27) and the indirect dimension can be processed (lines 31–37) before saving the resulting spectra (line 40).

Covariance processing of 2D NMR spectra

In addition to conventional NMR data processing, nmrglue provides a convenient platform for the facile development of new processing methods that are not included in the common software packages. One example is covariance processing, which is an alternative to the Fourier transform for the processing of 2D NMR spectra (Brüschweiler and Zhang 2004). Recently, the Covariance NMR Toolbox, which enables covariance processing of NMR data within MATLAB or OCTAVE (Short et al. 2011), was released; to the authors' best knowledge, this is currently the only available non-commercial software package for such data processing. At the root of covariance NMR is the computation of the covariance of a matrix. The NumPy library contains an efficient algorithm to perform such a calculation and can be used for covariance processing, provided that the data are appropriately prepared before and after the calculation. The script in Listing S10 uses nmrglue to read in data from a 2D solid-state NMR ^{13}C–^{13}C chemical shift correlation experiment—recorded with the dipolar assisted rotational resonance (DARR) mixing scheme (Takegoshi et al. 2001) for a sample of nanotubes formed by the bolaamphiphilic self-assembly of 1,4,5,8-naphthalenetetracarboxylic acid diimide with L-lysine headgroups (Shao et al. 2010)—which have been Fourier transformed along the direct dimension (line 5). Subsequently, the script computes the covariance of the data using NumPy's efficient algorithm (line 8), updates the necessary spectral parameters (line 11–14), and writes out the resulting covariance processed spectrum (line 17), which is shown in Fig. 5.

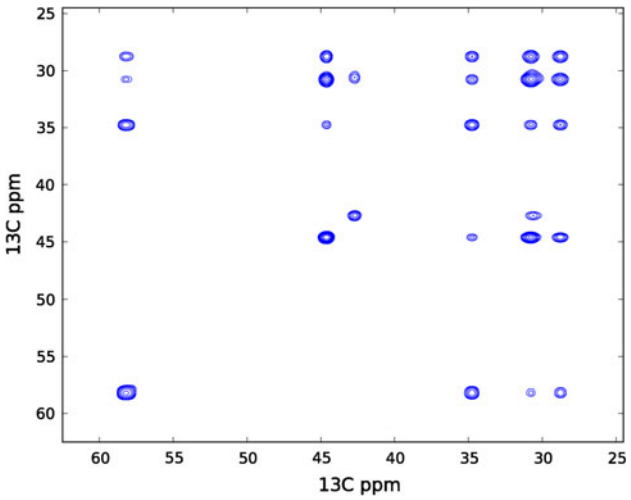

Fig. 5 2D ^{13}C–^{13}C DARR solid-state NMR spectrum generated using covariance processing for a sample of nanotubes formed by the bolaamphiphilic self-assembly of 1,4,5,8-naphthalenetetracarboxylic acid diimide with L-lysine headgroups (Shao et al. 2010). The figure was generated from the spectrum processed using the script in Listing S10

Preparation of strip plots from multiple 3D NMR spectra

Triple resonance 3D chemical shift correlation experiments have become the standard methodology for determining the sequential backbone resonance assignments for ^{13}C,^{15}N-enriched proteins. In solid-state NMR these assignments can frequently be established within Sparky or other software by using a combination of several complementary 3D ^{15}N–^{13}C–^{13}C spectra (e.g., NCACX and NCOCX) (Baldus 2002). For publications and presentations, resonance assignment data are most conveniently summarized by showing aligned strips from multiple 3D data sets. This process, which can be rather tedious and time consuming, can be readily automated in nmrglue. This is illustrated in Fig. 6, which shows representative strip plots from 3D CONCA, NCACX, and NCOCX spectra of the B3 immunoglobulin binding domain of protein G (GB3) in the microcrystalline solid phase (Nadaud et al. 2007). The Python script used to generate Fig. 6 is presented in Listing S11. This script can be readily edited to alter the various aspects of the figure such contour colors, fonts, tick placement and labeling.

Analysis of NMR relaxation data

A multitude of NMR methods have been developed to extract structural and dynamic data in biological macromolecules in site-specific fashion. These methods typically rely on the quantification of an observable, such as a dipolar coupling or relaxation rate constant, in one of the indirect dimensions of a multidimensional NMR experiment. While existing software packages, such as NMRPipe, provide facilities for the rapid analysis of such data sets based on peak heights or volumes extracted using lineshape fitting routines, they often yield suboptimal

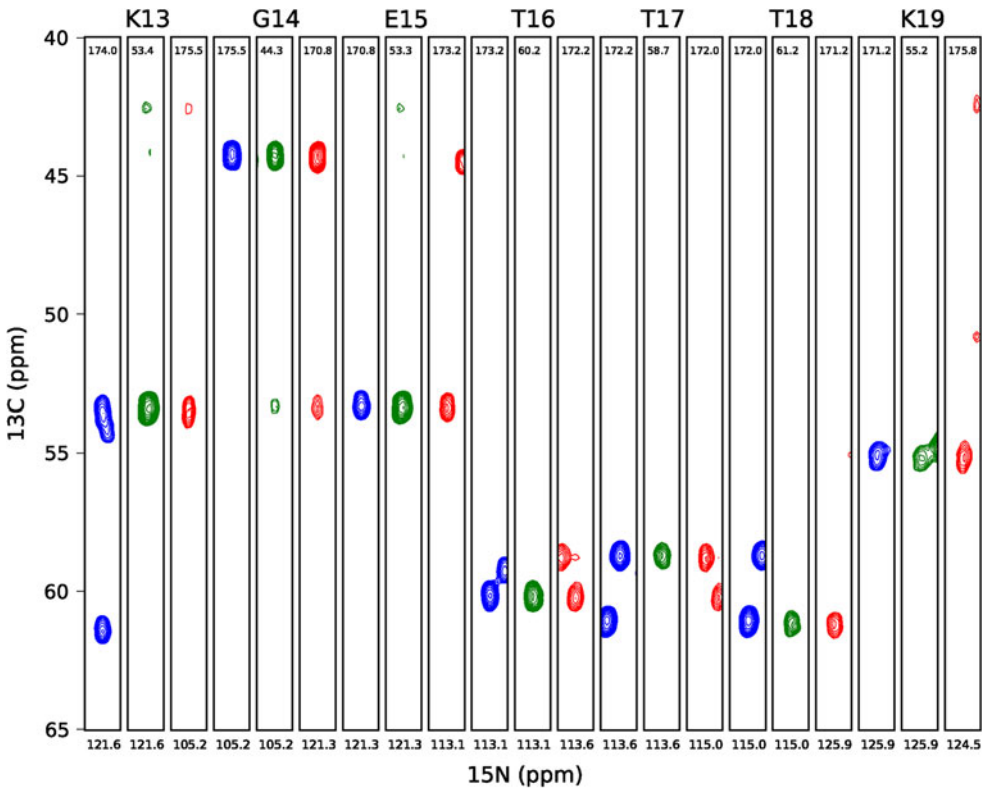

Fig. 6 Representative strips from 3D CONCA (*blue contours*), NCACX (*green contours*), and NCOCX (*red contours*) spectra corresponding to amino acid residues K13-K19 of GB3 (Nadaud et al. 2007). The figure was generated using the script in Listing S11

results in the context of solid-state NMR spectra which exhibit irregular lineshapes and/or limited sensitivity and resolution. In such cases a simple summation of spectral intensities within a small rectangular region around each peak provides a robust approach for estimating the experimental peak volumes, and also, importantly, their uncertainties based on the intrinsic spectral signal-to-noise ratio. Our group has made extensive use of this approach to analyze multiple dipolar (Helmus et al. 2008a, 2010) and relaxation (Nadaud et al. 2009, 2010, 2011; Helmus et al. 2010; Sengupta et al. 2012) trajectories in solid proteins in an automated manner.

This type of analysis in nmrglue is demonstrated here for a set of residue-specific longitudinal relaxation trajectories extracted from a series of 2D chemical shift correlation spectra recorded for the model B1 immunoglobulin binding domain of protein G (GB1). The Python script in Listing S12 reads in the integration limits and a list of spectra to analyze from text files (examples of which are available at the nmrglue website) on lines 5 and 6. An array that will hold the trajectories is created on line 9. Line 12 begins a loop in which a spectrum stored in an NMRPipe formatted file is read (line 16). An inner loop (lines 19–27) determines the correct order of the integration limits for each peak (lines 21–24) and computes the summation of all points within the spectrum that fall within these integration limits (line 27). Lines 30–36 create a series of text files ending with a ".dat" extension that contain the normalized relaxation trajectories. Trajectories which have not been normalized can be obtained by omitting line 31. Accurate locations of the integration regions around the cross-peaks are crucial for the reliable extraction of individual trajectories, especially for clusters of peaks exhibiting partial overlap. To aid in determining these locations the Python script in Listing S13 creates a series of 2D contour plots for each peak of interest (see Fig. 7 for a representative example). These plots display the region defined by the current set of integration limits as well as somewhat smaller and larger integration rectangles.

The relaxation trajectories extracted using the script in Listing S12 can be further analyzed with a variety of commercial or home-built software. Here this analysis, consisting of a fit of the experimental trajectory to a decaying single exponential, is illustrated using the constrained least-squares optimization method in nmrglue. The Python script in Listing S14 defines the fitting function (lines 7–9) as well as a residuals function that calculates the difference between the experimental and simulated trajectories (lines 12–14). The latter function is required for the least squares optimization procedure on line 33. Lines 17–19 read in the relaxation times and set the initial values and constraints on the fit parameters. The bounds defined on line 19 restrict the amplitude scaling factor A to values between 0.98 and 1.02, a reasonable range for the normalized experimental relaxation trajectories. The script then creates an output file "fits.txt" for storing the fitting

results (line 22–23). Line 26 begins a loop over all the peak trajectory files which have the ".dat" extension. The peak assignment is determined (line 28) and the corresponding trajectory loaded (line 32) based on the filename. Nmrglue's least-squares optimization algorithm is then utilized to find the best-fit values for the amplitude scaling and relaxation rate parameters (lines 33 and 34), and the fitting results are stored in the output file. The quality of the fitting can be examined using the Python script in Listing S15, which creates plots of the experimental and simulated relaxation trajectories for each cross-peak. An example of such a plot, for GB1 residue D40, is shown Fig. 8.

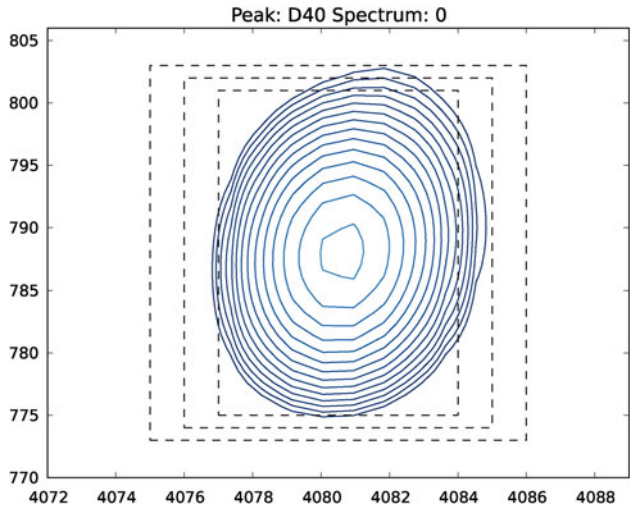

Fig. 7 A sample contour plot of a peak with lines indicating the current integration limits and limits which would correspond to changes of ±1 points in each dimension. The figure was generated using the script in Listing S13

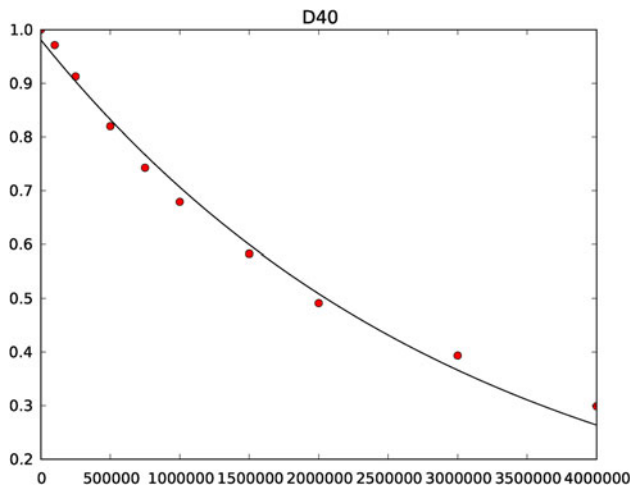

Fig. 8 A sample output of the script in Listing S15 showing the experimental longitudinal ^{15}N relaxation trajectory for the residue D40 in GB1 (*red dots*) and the best-fit simulated trajectory (*solid black line*) as determined by Listing S14

Collectively, the set of nmrglue scripts shown in Listings S12-15 enables the extraction and fitting of relaxation trajectories from a series of 2D NMR spectra in an automated and interactive manner. With relatively minor modifications these scripts can be used to analyze relaxation data from a series of 3D NMR chemical shift correlation spectra, or fit data from other types of arrayed NMR experiments by changing the fitting function in Listings S14 and S15.

Conclusions

We have introduced nmrglue, an open source software package for working with multidimensional NMR spectra in the Python programming language. Nmrglue harnesses the powerful and efficient NumPy, SciPy and matplotlib scientific Python libraries to provide a robust, flexible, and easy-to-use platform for NMR data processing, analysis and visualization, and contains numerous useful functionalities including linear prediction, peak picking and lineshape fitting. The software can also read, write and convert data stored in a variety of file formats including Bruker, Agilent/Varian, NMRPipe, Sparky, SIMPSON, and Rowland NMR Toolkit, enabling a number of existing programs to be easily interconnected. To demonstrate the utility and versatility of nmrglue, representative applications to the visualization of 1D, 2D and 3D NMR spectra, separating interleaved pseudo 3D experiments, covariance processing and analysis of solid-state NMR relaxation data were presented. Numerous additional examples, a tutorial and detailed documentation for the entire package are available online at the nmrglue website, http://nmrglue.com. Also available at the website are the install files for Linux, Windows and OS X operating systems, as well as the source code which can be freely redistributed and modified under the New BSD license.

Acknowledgments This work was supported in part by the National Science Foundation (CAREER Award MCB-0745754 to C.P.J.), the National Institutes of Health (R01GM094357 to C.P.J.), the Camille and Henry Dreyfus Foundation (Camille Dreyfus Teacher-Scholar Award to C.P.J.) and Eli Lilly and Company (Young Investigator Award to C.P.J.). The authors thank the current and former members of the Jaroniec research group (in particular P.S. Nadaud, M. Gao, C. Gupta, S.P. Pondaven, I. Sengupta, B. Wu and S. Mukherjee) for testing and providing valuable feedback on the early versions of nmrglue, and M. Fenwick and P. Semanchuk for reporting bugs and providing patches for the package. This work would not have been possible without the Scientific Python community, whose efforts have produced a powerful environment for scientific computing. The members of this community are too numerous to list here, however special thanks go to the late J.D. Hunter for his dedication to the community and contributions to creating the indispensable matplotlib package. J.J.H. also thanks J. Hoch (U. Connecticut Health Center) for supporting his continuing work on the development of nmrglue.

References

Bak M, Rasmussen JT, Nielsen NC (2000) SIMPSON: a general simulation program for solid-state NMR spectroscopy. J Magn Reson 147:296–330

Baldus M (2002) Correlation experiments for assignment and structure elucidation of immobilized polypeptides under magic angle spinning. Prog Nucl Magn Reson Spect 41:1–47

Beazley DM (2003) Automated scientific software scripting with SWIG. Future Gener Comput Syst 19:599–609

Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K (2011) Cython: the best of both worlds. Comput Sci Eng 13: 31–39

Blanton WB (2003) BlochLib: a fast NMR C++ tool kit. J Magn Reson 162:269–283

Brüschweiler R, Zhang F (2004) Covariance nuclear magnetic resonance spectroscopy. J Chem Phys 120:5253–5260

Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, Friedberg I, Hamelryck T, Kauff F, Wilczynski B, De Hoon MJL (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics 25:1422–1423

Delaglio F, Grzesiek S, Vuister GW, Zhu G, Pfeifer J, Bax A (1995) NMRPipe: a multidimensional spectral processing system based on UNIX pipes. J Biomol NMR 6:277–293

Delsuc MA (1988) Spectral representation of 2D NMR spectra by hypercomplex numbers. J Magn Reson 77:119–124

Goddard TD, Kneller DG (2008) SPARKY 3. University of California, San Francisco

Gorgolewski K, Burns CD, Madison C, Clark D, Halchenko YO, Waskom ML, Ghosh SS (2011) Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. Front Neuroinform. doi:10.3389/fninf.2011.00013

Günther UL, Ludwig C, Rüterjans H (2000) NMRLAB: advanced NMR data processing in Matlab. J Magn Reson 145:201–208

Helmus JJ, Nadaud PS, Höfer N, Jaroniec CP (2008a) Determination of methyl ^{13}C–^{15}N dipolar couplings in peptides and proteins by three-dimensional and four-dimensional magic-angle spinning solid-state NMR spectroscopy. J Chem Phys 128:052314

Helmus JJ, Surewicz K, Nadaud PS, Surewicz WK, Jaroniec CP (2008b) Molecular conformation and dynamics of the Y145Stop variant of human prion protein in amyloid fibrils. Proc Natl Acad Sci USA 105:6284–6289

Helmus JJ, Surewicz K, Surewicz WK, Jaroniec CP (2010) Conformational flexibility of Y145Stop human prion protein amyloid fibrils probed by solid-state nuclear magnetic resonance spectroscopy. J Am Chem Soc 132:2393–2403

Helmus JJ, Surewicz K, Apostol MI, Surewicz WK, Jaroniec CP (2011) Intermolecular alignment in Y145Stop human prion protein amyloid fibrils probed by solid-state NMR spectroscopy. J Am Chem Soc 133:13934–13937

Hoch JC, Stern A (1996) NMR data processing, 1st ed. Wiley-Liss, New York

Hunter JD (2007) Matplotlib: a 2D graphics environment. Comput Sci Eng 9:90–95

Jaroniec CP, Filip C, Griffin RG (2002) 3D TEDOR NMR experiments for the simultaneous measurement of multiple carbon-nitrogen distances in uniformly ^{13}C,^{15}N-labeled solids. J Am Chem Soc 124:10728–10742

Jones E, Oliphant T, Peterson P, et al (2001) SciPy: open source scientific tools for Python. http://www.scipy.org/

Keller RLJ (2004) The computer aided resonance assignment tutorial. Cantina Verlag, Goldau

Laage S, Lesage A, Emsley L, Bertini I, Felli IC, Pierattelli R, Pintacuda G (2009) Transverse-dephasing optimized homonuclear J-decoupling in solid-state NMR spectroscopy of uniformly ^{13}C-labeled proteins. J Am Chem Soc 131:10816–10817

Lauterbur PC (2005) All science is interdisciplinary: from magnetic moments to molecules to men (Nobel Lecture). Angew Chem Int Ed 44:1004–1011

Lewis IA, Schommer SC, Markley JL (2009) rNMR: open source software for identifying and quantifying metabolites in NMR spectra. Magn Reson Chem 47:S123–S126

Lutz M (2011) Programming Python, 4th ed. O'Reilly Media, Sebastopol

Marquardt DW (1963) An algorithm for least-squares estimation of nonlinear parameters. J Soc Ind Appl Math 11:431–441

Meissner A, Duus JO, Sørensen OW (1997) Spin-state-selective excitation. Application for E.COSY-type measurement of J_{HH} coupling constants. J Magn Reson 128:92–97

Nadaud PS, Helmus JJ, Jaroniec CP (2007) ^{13}C and ^{15}N chemical shift assignments and secondary structure of the B3 immunoglobulin-binding domain of streptococcal protein G by magic-angle spinning solid-state NMR spectroscopy. Biomol NMR Assign 1:117–120

Nadaud PS, Helmus JJ, Kall SL, Jaroniec CP (2009) Paramagnetic ions enable tuning of nuclear relaxation rates and provide long-range structural restraints in solid-state NMR of proteins. J Am Chem Soc 131:8108–8120

Nadaud PS, Helmus JJ, Sengupta I, Jaroniec CP (2010) Rapid acquisition of multidimensional solid-state NMR spectra of proteins facilitated by covalently bound paramagnetic tags. J Am Chem Soc 132:9561–9563

Nadaud PS, Sengupta I, Helmus JJ, Jaroniec CP (2011) Evaluation of the influence of intermolecular electron-nucleus couplings and intrinsic metal binding sites on the measurement of ^{15}N longitudinal paramagnetic relaxation enhancements in proteins by solid-state NMR. J Biomol NMR 51:293–302

Ni F, Scheraga HA (1986) Phase-sensitive spectral analysis by maximum entropy extrapolation. J Magn Reson 70:506–511

Nicholson JK, Lindon JC, Holmes E (1999) "Metabonomics": understanding the metabolic responses of living systems to pathophysiological stimuli via multivariate statistical analysis of biological NMR spectroscopic data. Xenobiotica 29:1181–1189

Nowling RJ, Vyas J, Weatherby G, Fenwick MW, Ellis HJC, Gryk MR (2011) CONNJUR spectrum translator: an open source application for reformatting NMR spectral data. J Biomol NMR 50:83–89

Oliphant TE (2007) Python for scientific computing. Comput Sci Eng 9:10–20

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Pellecchia M, Bertini I, Cowburn D, Dalvit C, Giralt E, Jahnke W, James TL, Homans SW, Kessler H, Luchinat C, Meyer B, Oschkinat H, Peng J, Schwalbe H, Siegal G (2008) Perspectives on NMR in drug discovery: a technique comes of age. Nat Rev Drug Discov 7:738–745

Perez F, Granger BE (2007) IPython: a system for interactive scientific computing. Comput Sci Eng 9:21–29

Peterson P (2009) F2PY: a tool for connecting Fortran and Python programs. Int J Comp Sci Eng 4:296

Pons J-L, Malliavin TE, Delsuc MA (1996) Gifa V. 4: a complete package for NMR data set processing. J Biomol NMR 8:445–452

Seabold S, Perktold J (2010) Statsmodels: econometric and statistical modeling with python. Proceedings of the 9th Python in science conference, pp 57–61

Sengupta I, Nadaud PS, Helmus JJ, Schwieters CD, Jaroniec CP (2012) Protein fold determined by paramagnetic magic-angle spinning solid-state NMR spectroscopy. Nat Chem 4:410–417

Shao H, Seifert J, Romano NC, Gao M, Helmus JJ, Jaroniec CP, Modarelli DA, Parquette JR (2010) Amphiphilic self-assembly of an n-type nanotube. Angew Chem Int Ed 49:7688–7691

Short T, Alzapiedi L, Brueschweiler R, Snyder D (2011) A covariance NMR toolbox for MATLAB and OCTAVE. J Magn Reson 209:75–78

Shuker SB, Hajduk PJ, Meadows RP, Fesik SW (1996) Discovering high-affinity ligands for proteins: SAR by NMR. Science 274:1531–1534

Smith SA, Levante TO, Meier BH, Ernst RR (1994) Computer simulations in magnetic resonance. An object-oriented programming approach. J Magn Reson A 106:75–105

States D, Haberkorn R, Ruben D (1982) A two-dimensional nuclear overhauser experiment with pure absorption phase in four quadrants. J Magn Reson 48:286–292

Stevens TJ, Fogh RH, Boucher W, Higman VA, Eisenmenger F, Bardiaux B, Van Rossum B-J, Oschkinat H, Laue ED (2011) A software framework for analysing solid-state MAS NMR data. J Biomol NMR 51:437–447

Takegoshi K, Nakamura S, Terao T (2001) ^{13}C–^{1}H dipolar-assisted rotational resonance in magic-angle spinning NMR. Chem Phys Lett 344:631–637

Turk MJ, Smith BD, Oishi JS, Skory S, Skillman SW, Abel T, Norman ML (2011) yt: a multi-code analysis toolkit for astrophysical simulation data. Astrophys J (Suppl Ser) 192:9

Van Beek JD (2007) matNMR: a flexible toolbox for processing, analyzing and visualizing magnetic resonance data in Matlab((R)). J Magn Reson 187:19–26

Van Rossum G (1995) Python tutorial, Technical Report CS-R9526

Veshtort M, Griffin RG (2006) SPINEVOLUTION: a powerful tool for the simulation of solid and liquid state NMR experiments. J Magn Reson 178:248–282

Vranken WF, Boucher W, Stevens TJ, Fogh RH, Pajon A, Llinas M, Ulrich EL, Markley JL, Ionides J, Laue ED (2005) The CCPN data model for NMR spectroscopy: development of a software pipeline. Proteins Struct Funct Bioinf 59:687–696

Wassenaar TA et al (2012) WeNMR: structural biology on the grid. J Grid Comp 10:743–767

Wüthrich K (2003) NMR studies of structure and function of biological macromolecules (Nobel Lecture). Angew Chem Int Ed 42:3340–3363

Supporting Information

Nmrglue: An Open Source Python Package for the Analysis of Multidimensional NMR Data

Jonathan J. Helmus and Christopher P. Jaroniec

```python
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()

# write results to NMRPipe file
ng.pipe.write('data.ft2', pdic, pdata)
```

Listing S1. A Python script for converting a 2D NMR data set between the Sparky and NMRPipe formats.

```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.eps")
```

Listing S2. A Python script for plotting the time domain data contained in an NMRPipe formatted file. The resulting plot is shown in Figure 2.

```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, 2500000)

# save the figure
fig.savefig("spectrum.eps")
```

Listing S3. A Python script which plots the 1D ^{13}C CP MAS NMR spectrum shown in Figure 3.

```python
import nmrglue as ng
import matplotlib.pyplot as plt

# read in data
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.eps")
```

Listing S4. A Python script which creates the plot of the 2D ^{15}N-^{13}C NMR spectrum and representative 1D traces shown in Figure 4.

```
import nmrglue as ng

# read in the NMR data
dic, data = ng.varian.read('arrayed_data.fid')

# set the new size of the separated data
dic['nblocks'] = data.shape[0]

# loop over the echo times, separating and saving each 2D
for i, techo in enumerate(dic['procpar']['techo']['values']):
    dir_name = 'techo_' + techo + '.fid'
    print "Creating directory:", dir_name
    ng.varian.write(dir_name, dic, data[:, i, :], overwrite=True)
```

Listing S5. A Python script which separates a pseudo 3D NMR data set collected in an interleaved manner into a series of 2D NMR chemical shift correlation spectra. The relaxation delay, given by the parameter "techo", was the innermost loop.

```
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)
```

Listing S6. A Python script which separates a pseudo 3D NMR data set consisting of a set of z-filtered TEDOR spectra collected in an interleaved manner. The dipolar mixing period, given by the parameter "nredor", was the next-to-last loop with the indirect dimension quadrature phase being the innermost loop.

```
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 S7. A Python script for manipulating the data from a 2D NMR data set collected using a S^3E filter (Laage S et al. (2009) J Am Chem Soc 131:10816-10817) in which the sum and difference between the alternating traces, the 'A' and 'B' blocks, must be calculated.

```
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 S8. A Python script for converting the two data sets created by Listing S7 from the Agilent/Varian format into NMRPipe format.

```
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)
ddic, ddata = ng.pipe_proc.ps(ddic, ddata, p0=-90.0, p1=0.0)
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=-79.0, p1=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, p1=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)
```

Listing S9. A Python script which processes the two NMRPipe formatted files created in Listing S8. The direct dimensions of the two components of the S^3E filtered 2D NMR experiment are processed independently and subsequently the two data sets are coadded before processing the indirect dimension.

```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)
```

Listing S10. A Python script for performing covariance processing on a NMRPipe formatted file, which has been previously processed along the direct dimension using a Fourier transform.

```
#! /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-3DNCACX-040507.fid/ft/test%03d.ft3'
spectrum_3 = 'GB3-CN-3DNCOCX-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
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
```

```
58
59  y_min = 40.0    # y axis minimum in ppm
60  y_max = 65.0    # y axis minimum in ppm
61
62  z_plane_s1 = table['CO_PPM'][:]    # strip plane in ppm, spectrum 1
63  z_plane_s2 = table['CA_PPM'][1:]   # strip plane in ppm, spectrum 2
64  z_plane_s3 = table['CO_PPM'][1:]   # strip plane in ppm, spectrum 3
65
66
67  fig = plt.figure()
68  for i in xrange(7):
69
70      ### spectral 1, CONCA
71      # find limits in units of points
72      idx_s1_a1 = uc_s1_a1(z_plane_s1[i], "ppm")
73      min_s1_a0 = uc_s1_a0(x_center_s1[i] + x_width, "ppm")
74      max_s1_a0 = uc_s1_a0(x_center_s1[i] - x_width, "ppm")
75      min_s1_a2 = uc_s1_a2(y_min, "ppm")
76      max_s1_a2 = uc_s1_a2(y_max, "ppm")
77
78      if min_s1_a2 > max_s1_a2:
79          min_s1_a2, max_s1_a2 = max_s1_a2, min_s1_a2
80
81      # extract strip
82      strip_s1 = data_1[min_s1_a0:max_s1_a0+1, idx_s1_a1,
83  min_s1_a2:max_s1_a2+1]
84
85      # determine ppm limits of contour plot
86      strip_ppm_x = uc_s1_a0.ppm_scale()[min_s1_a0:max_s1_a0+1]
87      strip_ppm_y = uc_s1_a2.ppm_scale()[min_s1_a2:max_s1_a2+1]
88      strip_x, strip_y = np.meshgrid(strip_ppm_x, strip_ppm_y)
89
90      # add contour plot of strip to figure
91      ax1 = fig.add_subplot(1, 21, 3 * i + 1)
92      ax1.contour(strip_x, strip_y, strip_s1.transpose(), cl_s1,
93                      colors=colors_s1, linewidths=0.5)
94      ax1.invert_yaxis()  # flip axes since ppm indexed
95      ax1.invert_xaxis()
96
97      # turn off tick and labels, add labels
98      ax1.tick_params(axis='both', labelbottom=False, bottom=False, top=False,
99                      labelleft=False, left=False, right=False)
100     ax1.set_xlabel('%.1f'%(x_center_s1[i]), size=6)
101     ax1.text(0.1, 0.975, '%.1f'%(z_plane_s1[i]), size=6,
102                                         transform=ax1.transAxes)
103
104     # label and put ticks on first strip plot
105     if i == 0:
106         ax1.set_ylabel("13C (ppm)")
107         ax1.tick_params(axis='y', labelleft=True, left=True, direction='out')
108
109     ### spectra 2, NCACX
110     # find limits in units of points
111     idx_s2_a0 = uc_s2_a0(z_plane_s2[i], "ppm")
112     min_s2_a1 = uc_s2_a1(x_center_s2[i] + x_width, "ppm")
113     max_s2_a1 = uc_s2_a1(x_center_s2[i] - x_width, "ppm")
114     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)

    # turn off ticks and labels, add labels and assignment
    ax2.tick_params(axis='both', labelbottom=False, bottom=False, top=False,
                    labelleft=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)

    # turn off ticks and labels, add labels
    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.eps')
```

Listing S11. A Python script which creates the strip plots of the 3D NMR spectra shown in Figure 6.

```
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:", spectra
    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 S12. A Python script used to extract relaxation trajectories from a series of 2D NMR spectra by integrating over rectangular regions around the assigned peaks. See the nmrglue website (http://nmrglue.com) for examples of the “boxes.in” and “spectra.in” input files.

```
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, 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, 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 S13. A Python script to create plots of showing small spectral and integration regions from which the relaxation trajectories were extracted using the script in Listing S12. A sample plot created by this script is shown in Figure 7.

```
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.
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\tA\t\tR2\t\tier\n")

# loop over the trajecory files
for filename in glob.glob('*.dat'):

    peak = filename[:3]
    print "Fitting Peak:", peak

    # fit the trajectory using contrainted least squares optimization
    trajectory = np.loadtxt(filename)
    x, ier = leastsqbound(residuals, x0, bounds=bounds,
                          args=(trajectory, relaxation_times))

    # write fitting results to output file
    output.write('%s\t%.6f\t%.6f\t%i\n' % (peak, x[0], x[1], ier))

output.close()  # close the output file
```

Listing S14. A Python script that fits the experimental relaxation trajectories extracted by Listing S12 to a single exponential decay by using nmrglue's constrained least-square minimization algorithm. Example input and output files for this script are available at the nmrglue website (http://nmrglue.com).

```
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_relaxation_times, experimental_trajectory, 'or')
    ax.plot(similated_relaxation_times, simulated_trajectory, '-k')
    ax.set_title(peak)
    fig.savefig(peak + "_plot.eps")
```

Listing S15. A Python script that plots the experimental relaxation trajectories extracted in Listing S12 and simulated relaxation trajectories obtained by the fitting performed in Listing S14. Example input and output files for this script are available at the nmrglue website (http://nmrglue.com), and a sample plot created by the script is provided in Figure 8.