Journal of Biomolecular NMR, 6 (1995) 277–293
ESCOM

J-Bio NMR 305

NMRPipe: A multidimensional spectral processing system based on UNIX pipes*

Frank Delaglio[a,**], Stephan Grzesiek[a], Geerten W. Vuister[b], Guang Zhu[c], John Pfeifer[d] and Ad Bax[a]

[a]*Laboratory of Chemical Physics, National Institute of Diabetes and Digestive and Kidney Diseases, National Institutes of Health, Bethesda, MD 20892, U.S.A.*
[b]*Bijvoet Center for Biomolecular Research, Utrecht University, Padualaan 8, 3584 CH Utrecht, The Netherlands*
[c]*Department of Biochemistry, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong*
[d]*Division of Computer Research and Technology, National Institutes of Health, Bethesda, MD 20892, U.S.A.*

Received 24 May 1995
Accepted 31 July 1995

Keywords: Multidimensional NMR; Data processing; Fourier transformation; Linear prediction; Maximum entropy; UNIX

Summary

The NMRPipe system is a UNIX software environment of processing, graphics, and analysis tools designed to meet current routine and research-oriented multidimensional processing requirements, and to anticipate and accommodate future demands and developments. The system is based on UNIX pipes, which allow programs running simultaneously to exchange streams of data under user control. In an NMRPipe processing scheme, a stream of spectral data flows through a pipeline of processing programs, each of which performs one component of the overall scheme, such as Fourier transformation or linear prediction. Complete multidimensional processing schemes are constructed as simple UNIX shell scripts. The processing modules themselves maintain and exploit accurate records of data sizes, detection modes, and calibration information in all dimensions, so that schemes can be constructed without the need to explicitly define or anticipate data sizes or storage details of real and imaginary channels during processing. The asynchronous pipeline scheme provides other substantial advantages, including high flexibility, favorable processing speeds, choice of both all-in-memory and disk-bound processing, easy adaptation to different data formats, simpler software development and maintenance, and the ability to distribute processing tasks on multi-CPU computers and computer networks.

Introduction

As use of multidimensional NMR has become widespread, demands on multidimensional spectral processing software have increased. Software must keep pace both with NMR applications research, and with the routine use of NMR for biomolecular structure determination. Routine use requires software to accommodate increasing numbers of experiments, larger data sizes, more complicated processing schemes, and common use of 4D NMR (Pelczer and Szalma, 1991; Bax and Grzesiek, 1993). Various vendor-specific modes of quadrature detection and data storage must also be addressed. At the same time, NMR technique development research requires software to serve as a platform for testing and evaluation of new experiments and acquisition methods, as well as new spectral analysis and enhancement approaches.

The user community for multidimensional processing software is also changing, and many practitioners of biological NMR are not necessarily familiar with NMR computer applications or signal processing. In addition, there are generally increasing expectations for software that is graphically oriented, error-free, and works harmoniously with other applications on a variety of net-

**Availability:* The NMRPipe system is available via a secured-access anonymous ftp site. For details on retrieving the software, send a request by electronic mail addressed to 'delaglio@helix.nih.gov'.

**To whom correspondence should be addressed at: National Institutes of Health, Laboratory of Chemical Physics, NIDDK, Building 5 B2-31, 5 Center Drive MSC 0505, Bethesda, MD 20892-0505, U.S.A.

Abbreviations: 1D, 2D, 3D, one-, two-, three-dimensional; nD, multidimensional; CPU, central processing unit; FID, free induction decay; I/O, input/output; LP, linear prediction; MEM, maximum entropy method; Mb, megabyte; NOE, nuclear Overhauser effect.

worked computers. Correspondingly, current software development approaches often favor creation of several small, well-targeted applications, coordinated by standard graphics and command tools.

We present here the NMRPipe system, a comprehensive new multidimensional NMR data processing system that addresses the growing needs for ease of use, efficiency, and flexibility of multidimensional spectral processing in the laboratory network. The NMRPipe system is a UNIX pipeline-based software environment for multidimensional processing, coordinated with spectral graphics and analysis tools. The system was implemented in the C programming language (Kernighan and Ritchie, 1988), using the program development tools of UNIX (Kernighan and Pike, 1984).

Several other multidimensional NMR data processing packages have been developed over the past decade, including the popular FELIX (Biosym Technologies Inc., San Diego, CA), as well as AZARA (W. Boucher, unpublished results), Dreamwalker (Meadows et al., 1994), GIFA (Delsuc, 1989), NMR Toolkit (Hoch, 1985), NMRZ (New Methods Research Inc., Syracuse, NY), Pronto (Kjaer et al., 1994), PROSA (Güntert et al., 1992), and TRIAD (Tripos Inc., St. Louis, MO). The NMRPipe system incorporates a novel approach to spectral processing that is complementary to other methods, and provides many advantages. Spectral processing is performed using modules connected by UNIX pipes, which allow programs running simultaneously to exchange streams of data under user control. In this approach, a stream of spectral data flows through a pipeline of processing programs, each of which performs one component of the overall scheme, such as Fourier transformation or mirror-image linear prediction.

The processing programs of the NMRPipe system work in the same way as ordinary UNIX commands; this means that complete multidimensional processing schemes can be constructed as standard UNIX command scripts, which are easy to learn and manipulate. The pipeline approach provides favorable processing speeds, while at the same time allowing the choice of both all-in-memory and disk-bound processing, easy adaptation of new algorithms and differing data formats, and simpler software development and maintenance. Since processing is achieved via a series of programs running simultaneously, the NMRPipe pipeline approach also provides a way to exploit the capabilities of multiprocessor computers or to distribute processing tasks across a network.

In addition to the general advantages of the pipeline approach, there are other advantages that arise from specific details of NMRPipe's implementation. For example, the components of NMRPipe are engineered to maintain and exploit accurate records of data size, detection mode, calibration information, and processing parameters in all dimensions. This means that schemes can be created and reused easily, since parameters can be specified in terms of spectral units, and there is no need to explicitly define or anticipate data sizes during processing. The parameter record also allows NMRPipe modules to assemble the correct combination of real and imaginary data for a given dimension automatically; this permits dimensions to be processed and reprocessed in any order with schemes that are generally the same, regardless of acquisition mode and vendor-specific storage details.

Methods

The NMRPipe approach relies on the UNIX operating system concepts of data streams, filters, and pipes, so these are discussed in some detail here. By necessity, these concepts are becoming increasingly familiar to the biomolecular NMR community, since modern spectrometers are commonly controlled by UNIX computers, and molecular structures are usually generated and visualized on UNIX workstations.

UNIX commands and filters

UNIX has no strong distinction between commands built into the operating system and programs that are part of 'external' applications such as spectral processing. This means that application programs can potentially be used like ordinary UNIX commands, and the standard UNIX facilities for combining and manipulating them can be exploited. For example, one or more UNIX commands can be placed into an ordinary text file, called a *shell script*. Such a shell script can then be executed by its name, just as if it were also a UNIX command.

A UNIX *filter* is a general term for a command or program that reads input, processes it in some way, and produces an output. One example of a filter is the UNIX command **sort,** which reads lines of text and writes them out again sorted in alphabetical order. Another example is the UNIX command **tr**, which translates characters (e.g. from upper-case to lower-case) in its input before writing them. Depending on the nature of the task involved, UNIX filters may read and process their input data in small parts, such as **tr** (which can process one character at a time), or in its entirety, such as **sort** (which must read the entire input first in order to sort it).

In UNIX terminology, a filter's source of input data is called *standard input* and its destination for output data is called *standard output*. By default, standard input is data entered from the keyboard, and standard output is data displayed on the computer screen. UNIX allows filters to take their input from an existing file instead of the keyboard; this is called *input redirection*, and it is performed using the < character. Correspondingly, filters can send their output to a file instead of to the screen; this is called *output redirection*, and it is performed using the > character. The following two UNIX commands

show examples of redirection. The first command sorts the lines in file 'old.text', and writes the sorted results to file 'new1.text'; the second command converts the text in file 'new1.text' from lower-case to upper-case, and stores the result in file 'new2.text':

```
sort < old.text > new1.text
tr 'a–z' 'A–Z' < new1.text > new2.text
```

Commands like these illustrate the concept of a *data stream*, where data 'flows' from an input source, travels through a filter, and collects at an output destination.

UNIX command-line arguments

The use and behavior of a UNIX command can be adjusted by command-line arguments, which are additional parameters specified after the command. The parameters are usually identified by words or letters prefixed by the - character. For instance, while the UNIX command **sort** will sort text in alphabetical order, adding the argument **-r** will cause text to be sorted in reverse alphabetical order:

```
sort -r < old.text > new1.text
```

Each UNIX command has its own list of possible command-line arguments, which are described in the command's *manual page*, a brief document (but often more than one page) that is available on-line. UNIX manual pages have a standard format, and new manual pages can be added easily, so that application programs can make use of the same on-line help system used by other UNIX commands.

UNIX pipes

UNIX pipes allow commands to be connected together in a series, where the output of one command is used directly as the input to the next command. A series of programs connected in this way is often called a *pipeline*. A pipe is specified in a UNIX command line by the | character inserted between commands. For example, we can combine the sorting and character translation commands into a single pipeline:

```
sort < old.text | tr 'a–z' 'A–Z' > new2.text
```

In this pipeline, data travels from the input file through the **sort** filter, and the sorted result travels via a pipe through the **tr** filter and then to the output file. As shown, pipes allow simple commands to be combined to perform complex tasks, while avoiding the need for intermediate results to be saved in files. Pipeline communication is also relatively fast, since UNIX pipes are generally implemented via physical memory buffers in the operating system (Stevens, 1992).

Pipelines, like UNIX command lines in general, can be split over several lines of text. This is especially useful when the pipeline contains many components. In the UNIX idiom, the \ character is used at the end of a line to continue a command onto the next line. For example, a functionally equivalent version of the **sort** pipeline described above could be entered as follows:

```
sort < old.text \
| tr 'a–z' 'A–Z' > new2.text
```

Spectral processing function as a UNIX filter

The concept of a UNIX filter command can be extended directly to spectral processing. By analogy, a spectral processing function can be implemented as a UNIX filter, which reads an input stream of unprocessed spectral data vectors, applies a spectral processing function to each vector, and writes the result as a stream of processed vectors. We have implemented this concept as a program called **nmrPipe**, the central module of the NMRPipe system.

The **nmrPipe** program applies a given processing function to a stream of spectral data. The processing function is selected via a 'function name' argument **-fn**, and corresponding processing modes and parameters are specified by other optional command-line arguments. For example, the following three commands are filters that apply a forward Fourier transform (FT), an inverse Fourier transform, and a 90-degree zero-order phase correction (PS), respectively:

```
Forward transform filter:     nmrPipe -fn FT
Inverse transform filter:     nmrPipe -fn FT -inv
Phase correction filter:      nmrPipe -fn PS -p0 90
```

The required input stream for **nmrPipe** consists of a header describing the data, followed by the binary data vectors themselves, usually in a sequential order. The output stream consists of the header, which is updated to reflect processing, followed by the processed data vectors. The stream format is meant to resemble the contents of an ordinary 2D file plane, so that such a file can be used directly with **nmrPipe**.

As with other UNIX filters, **nmrPipe** reads and writes streams via standard input and standard output, but for convenience explicit input and output file names can be specified by the command-line arguments **-in** and **-out**. For example, the following two commands perform the same task; they both apply a Fourier transform to all the data vectors in file 'spec.fid', and save the result in file 'spec.ft':

```
nmrPipe -fn FT < spec.fid > spec.ft
nmrPipe -fn FT -in spec.fid -out spec.ft
```

The **nmrPipe** program includes implementations of many

common 1D processing functions, as well as several other useful elements; these are listed in Table 1, and several are discussed in more detail below.

Spectral processing scheme as a UNIX pipeline

The concept of a spectral processing function performed as a UNIX filter leads directly to the idea of a

TABLE 1
PROCESSING FUNCTIONS OF THE nmrPipe PROGRAM[a]

Name	Function	Comments
NULL	Null function	No change to data
MAC	Macro interpreter	User-written functions in a subset of C
FT	Fourier transform	Complex, real, inverse, sign adjust, auto mode, etc.
HT	Hilbert transform	Ordinary, mirror image, auto mode
LP	Linear prediction[b]	Forward–backward[c], mirror image[d], etc.
MEM	Maximum entropy method[e]	Prototype, 1D to 4D, two channel[f], deconvolution[g]
EM	Exponential window	First point scaling, inverse mode
GM	Lorentzian/Gaussian window	First point scaling, inverse mode
TM	Trapezoid window	First point scaling, inverse mode
SP	Sine to a power window	First point scaling, inverse mode
ZF	Zero-fill	Inverse mode
EXT	Extract a region	By points, Hz, ppm, %, or left, right, etc.
PS	Phase correction	Frequency shift, inverse mode
MC	Modulus calculation	Modulus or power spectrum
SOL	Solvent filter	Time-domain convolution[h]
POLY	Polynomial solvent filter	Time-domain polynomial subtraction[i]
POLY	Polynomial base-line correction	Manual or automatic[j], all or selected region
MED	Model-free base-line correction	Automatic median method[k]
BASE	Linear base-line correction	Manually selected series of regions
CBF	Constant FID correction	DC correction of FID
QART	Quad artefact reduction[l]	Manual or automatic
SMO	Smoothing filter	Adjustable filter length and coefficients
TP	2D X/Y transpose	In-memory; identical to YTP
YTP	2D X/Y transpose	In-memory, all combinations of real and complex data
ZTP	3D X/Z transpose	In-memory, all combinations of real and complex data
ATP	4D X/A transpose	In-memory, all combinations of real and complex data
REV	Reverse data	Updates calibration
LS	Left shift	Updates calibration
RS	Right shift	Updates calibration
CS	Circular shift	Updates calibration, can invert signs of shifted data
FSH	Shift via Fourier transform	Provides non-integer shifts
SHUF	Various shuffling functions	Complex interleave, byte swap, etc.
SIGN	Various sign manipulations	Negate all, negate half, sign alternate, etc.
DX	Derivative	
INTEG	Integral	
COAD	Co-addition of data	Linear combination of points, vectors, or planes
ZD	Zero diagonal region	Adjustable diagonal slope, width, and offset
SET	Set data to constant	All data or specified region
ADD	Add a constant	All data or specified region
MULT	Multiply by a constant	All data or specified region

[a] Several functions are described in more detail in the Appendix.
[b] Kumaresan and Tufts, 1982; Barkhuijsen et al., 1985,1987; Stephenson, 1988; Hoch, 1989; Olejniczak and Eaton, 1990; Zhu and Bax, 1992a.
[c] Delsuc et al., 1987; Zhu and Bax, 1992b.
[d] Zhu and Bax, 1990.
[e] Maximum Entropy Reconstruction (Sibisi, 1983; Skilling and Bryan, 1984; Hore, 1985; Laue et al., 1985a; Stephenson, 1988; Kauppinen and Saario, 1993; Schmieder et al., 1994) is implemented according to the method of Gull and Daniell (Gull and Daniell, 1978; Wu, 1984).
[f] Laue et al., 1985b; Hoch et al., 1990.
[g] Ni and Scheraga, 1986; Ni et al., 1986; Mazzeo et al., 1989.
[h] Marion et al., 1989a.
[i] Callaghan et al., 1984.
[j] Details of automated base-line detection are given in the Appendix entry for function POLY.
[k] Friedrichs, 1995.
[l] Parks and Johannesen, 1976; the automated mode uses a grid search to minimize the integral of an interactively selected artefact.

```
bruk2pipe -in ser                                          \   Spectrometer-Format Input
  -xN        1024   -yN         104   -zN          64 \   Total Points in File
  -xT         512   -yT          52   -zT          32 \   Complex Points Acquired
  -xMODE Complex   -yMODE Complex   -zMODE Complex \   Acquisition Mode
  -xSW    7575.76   -ySW    8445.95   -zSW    1515.15 \   Spectral Width, Hz
  -xOBS   500.130   -yOBS    125.76   -zOBS   50.6800 \   Observe Frequency, MHz
  -xCAR     4.683   -yCAR      46.0   -zCAR    117.00 \   Carrier Position, PPM
  -xLAB        HN   -yLAB      CACB   -zLAB         N \   Axis Labels
  -ndim         3   -aq2D   States                   \   Dimension Count, 2D Mode
  -out fid/cbcaconh%03d.fid -verb -ov                      Output File Series
```

Fig. 1. Annotated format conversion script used for a 3D CBCA(CO)NH FID acquired on a Bruker AMX spectrometer. The general form of the conversion script is the same for other spectrometers. Parameters for each dimension are specified via arguments prefixed by **-x**, **-y**, **-z**, and **-a** for the X-axis, Y-axis, Z-axis, and A-axis of the data. In order to accommodate padding that may have been performed by the spectrometer, there are separate parameters for the number of points stored in the input file and the number of points actually acquired. The acquisition modes are specified by keywords such as 'Sequential' (Redfield and Kunz, 1975), 'Complex' or 'States' (States et al., 1982), 'TPPI' (Marion and Wüthrich, 1983), 'States-TPPI' (Marion et al., 1989b), etc., which define the Fourier transform mode and sign manipulation required; chemical shift calibration parameters are also recorded. The NMRPipe format output series is specified by the argument **-out**. Complete argument details are given in the Appendix.

spectral processing scheme implemented as a UNIX pipeline; this is the central concept of the NMRPipe system. In this method, spectral data flows through a pipeline of processing filters, each performing one aspect of the processing scheme. In practice, this is achieved by using multiple instances of the **nmrPipe** program, each with different command-line arguments to select a processing function and optional parameters. For example, the following scheme applies a sinusoid-to-a-power window function (SP), zero-fill (ZF), Fourier transform (FT), and deletes the imaginary part of the result (**-di**). In the absence of additional arguments, the processing functions in this scheme use default parameters, so that the SP function applies a sine bell, the ZF function doubles the data size, and the FT function applies a complex forward transform:

```
nmrPipe -fn SP -in spec.fid \
| nmrPipe -fn ZF \
| nmrPipe -fn FT -di -out spec.ft
```

Considered in more detail, this scheme consists of three instances of **nmrPipe**, connected by pipes, and running 'simultaneously'. This means that the UNIX operating system will alternate CPU time and other resources between the instances of **nmrPipe** while the scheme is executing. During execution, the first instance of nmrPipe reads a data vector from the input file 'spec.fid', applies the window function SP, and writes the result vector to the pipeline. The second instance of **nmrPipe** reads the apodized vector from the pipeline when it becomes available, applies zero-filling, and writes the result to the next stage of the pipeline. The third instance of **nmrPipe** reads the apodized, zero-filled vector from the pipeline when it becomes available, applies a Fourier transform, and writes the result to file 'spec.ft'; meanwhile, the earlier instances of **nmrPipe** may have already begun to read and process the next vector. This procedure continues until all vectors have passed through the pipeline.

Spectrometer format conversion

Many of the advantages of the NMRPipe system stem from the fact that relevant acquisition parameters for all dimensions are established during conversion of data from the spectrometer format to the NMRPipe format. A typical 3D conversion script is given in Fig. 1. As shown, the conversion establishes the acquisition modes, data sizes and chemical shift calibration information for each dimension. The parameters are usually entered manually, but most of these could be extracted automatically from spectrometer parameter files (D. Benjamin, private communication).

The conversion programs themselves have been engineered to compensate for vendor-specific differences in the way that real and imaginary data are interleaved for each dimension, so that the converted result always provides the real and imaginary data for all dimensions in a predictable order. This allows subsequent processing schemes to be independent of spectrometer vendor. Currently, the NMRPipe system includes conversion facilities for GE Omega export format, JEOL GX and Alpha formats, Chemagnetics format, Varian Unity format, and Bruker AM, AMX, and DMX formats.

Like **nmrPipe**, the conversion programs are also implemented as UNIX filters. This means that the output stream of a conversion command can be sent directly into a processing pipeline, without the need to save an intermediate converted result on disk. It also means that a conversion program can read data produced by another pipeline command as an alternative to reading data di-

rectly from a file. One useful example of this is the ability to convert data directly from a tape drive by using a tape reading command (such as the UNIX command **dd**) as the data source. Another example is the ability to convert versions of spectrometer data that were compressed to save space, by using a decompression command (such as the UNIX command **zcat**) as the data source.

Multidimensional processing via pipelines

The NMRPipe system includes two approaches to extend the pipeline method to multiple dimensions. One approach is to insert an appropriate matrix transpose command into the interior of a processing pipeline. Another approach is to use commands at the beginning or end of the pipeline that are capable of reading or writing vectors from an arbitrary dimension of a multidimensional spectrum. The two approaches can be used separately or in combination.

In a pipeline, a transpose function acts like a reservoir, which accumulates an intermediate result in memory before sending the transposed version down the remainder of the pipeline. Therefore, functions before a transpose receive and process a stream of vectors from a given dimension, and functions after the transpose receive and process a stream of vectors from the exchanged dimension. Depending on which dimensions are being exchanged, a transpose function may require only enough memory for a 2D plane from the data, or it may require enough memory for an entire 3D or 4D matrix, so it is not generally applicable.

As noted above, the pipeline approach can be extended to multidimensional processing simply by adding two kinds of modules, as an alternative to in-memory transpose. The first module is a program at the head of the pipeline, which creates a data stream by reading vectors from a given dimension of a multidimensional input. The second module is a program at the tail of the pipeline, which gathers processed vectors and writes them to a given dimension of a multidimensional output. We have implemented two such programs, **xyz2pipe** and **pipe2xyz**, which are suitable for reading and writing multidimensional data in the multifile 2D plane format suggested by Kay et al. (1989). The programs take their names from the nomenclature X-axis, Y-axis, Z-axis, A-axis, etc., which we use to describe the dimensions of the spectral data. Correspondingly, the dimension to be read or written is specified simply as a command-line argument **-x**, **-y**, **-z**, or **-a**. When reading or writing from a given dimension, the programs alter the sequential order of the other dimensions in the data stream in a regular, predictable way, by a multidimensional rotation. This means that schemes can be created to conserve the original data order, or change it to accommodate a particular processing or analysis strategy. The programs require at most enough physical memory to contain only four or so 2D planes from the data. In addition, the programs have

```
xyz2pipe -in fid/hnco%03d.fid -x  -verb           \   Read Vectors from X-Axis
| nmrPipe -fn SOL                                 \   Solvent Filter
| nmrPipe -fn SP -off 0.4 -end 0.98 -pow 2 -c 0.5 \   Window, 1st Point Scale
| nmrPipe -fn ZF                                  \   Zero Fill
| nmrPipe -fn FT                                  \   Fourier Transform
| nmrPipe -fn PS -p0 43 -p1 0.0 -di               \   Phase, Delete Imaginaries
| nmrPipe -fn EXT -x1 11ppm -xn 5.5ppm -sw        \   Extract Amide Region
| nmrPipe -fn TP                                  \   2D Transpose X/Y
| nmrPipe -fn SP -off 0.4 -end 0.95 -pow 1        \   Window
| nmrPipe -fn ZF                                  \   Zero Fill
| nmrPipe -fn FT                                  \   Fourier Transform
| nmrPipe -fn PS -p0 -90 -p1 180 -di              \   Phase, Delete Imaginaries
| pipe2xyz -out ft/hnco%03d.ft2 -y                    Write Vectors to Y-Axis

xyz2pipe -in ft/hnco%03d.ft2 -z -verb             \   Read Vectors from Z-Axis
| nmrPipe -fn SP -off 0.4 -end 0.95 -pow 1 -c 0.5 \   Window, 1st Point Scale
| nmrPipe -fn ZF                                  \   Zero Fill
| nmrPipe -fn FT                                  \   Fourier Transform
| nmrPipe -fn PS -p0 0.0 -p1 0.0 -di              \   Phase, Delete Imaginaries
| pipe2xyz -out ft/hnco%03d.ft3 -z                    Write Vectors to Z-Axis
```

Fig. 2. Annotated processing script for 3D amide proton-detected data, illustrating the use of 2D transpose. In this scheme, the X-axis and Y-axis are read, processed, and written in the first pass, and the Z-axis is read, processed and written in the second pass. Each pass consists of a pipeline beginning with the **xyz2pipe** program and ending with the **pipe2xyz** program; these programs use the arguments **-x**, **-y**, **-z**, and **-a** to specify which dimension is being read or written. The input and output file series are specified by the template arguments **-in** and **-out**. Complete argument details are given in the Appendix.

```
bruk2pipe -in ser $ARGS                                  \  Convert Bruker Format
| nmrPipe -fn SP -off 0.35 -end 0.95 -pow 2 -c 0.5 \  Window, Scale 1st Point
| nmrPipe -fn ZF -size 512                               \  Zero Fill
| nmrPipe -fn FT -di                                     \  Fourier Transform
| nmrPipe -fn TP                                         \  2D Transpose X/Y
| nmrPipe -fn SP -off 0.35 -end 1.0 -pow 1  -c 0.5 \  Window, Scale 1st Point
| nmrPipe -fn ZF -size 128                               \  Zero Fill
| nmrPipe -fn FT -di                                     \  Fourier Transform
| pipe2xyz -out ft/noe%02d%03d.DAT -y                       Write Vectors to Y-Axis

xyz2pipe -in ft/noe%02d%03d.DAT -z -verb                 \  Read Vectors from Z-Axis
| nmrPipe -fn SP -off 0.35 -end 0.95 -pow 1 -c 1.0 \  Window
| nmrPipe -fn ZF -size 64                                \  Zero Fill
| nmrPipe -fn FT -di                                     \  Fourier Transform
| pipe2xyz -out ft/noe%02d%03d.DAT -z -inPlace              Write Vectors to Z-Axis

xyz2pipe -in ft/noe%02d%03d.DAT -a -verb                 \  Read Vectors from A-Axis
| nmrPipe -fn SP -off 0.35 -end 0.95 -pow 1 -c 1.0 \  Window
| nmrPipe -fn ZF -size 64                                \  Zero Fill
| nmrPipe -fn FT -di                                     \  Fourier Transform
| pipe2xyz -out ft/noe%02d%03d.DAT -a -inPlace              Write Vectors to A-Axis
```

Fig. 3. Annotated 4D format conversion and processing script for a 256*×64*×16*×16* point 4D ^{13}C-^{13}C correlated ^{1}H-^{1}H NOE FID, illustrating the use of 2D transpose (the asterisks denote complex data). Acquisition parameters have been abbreviated by $ARGS and phase correction steps have been omitted to save space. In this scheme, the results of the format conversion program **bruk2pipe** are sent directly to the processing pipeline without the need to save an intermediate converted FID on disk. The size of the final result is 512×128×64×64 points. Processing time: 8 h and 20 min on a Sun Sparc 10 workstation.

been engineered to allow in-place processing (i.e., same input and output files), and to provide the correct combinations of real and imaginary data so that dimensions can be processed in any order.

In the simplest multidimensional scheme, each dimension of the data is processed in a separate pass, which requires reading the entire input from disk, and writing the entire result. Such a scheme can be simplified and made more efficient by adding one or more in-memory transpose steps, which eliminates the need to save an intermediate result on disk. A typical 3D processing script employing a 2D transpose approach is shown in Fig. 2. In this script, the X-axis and Y-axis are processed together in the first pass, after which the Z-axis is processed in a second pass. Such a script represents an effective compromise between disk access and physical memory use, since in practice only a small number of 2D planes are being manipulated in memory at any given time by the various programs in the pipeline. If large amounts of physical memory are available, schemes with 3D or 4D in-memory transpose steps can also be constructed, again reducing the need to save intermediate results. The overall approach provides basic multidimensional schemes, which require only modest amounts of memory for 3D or 4D processing, but which can be altered easily to take advantage of large memory systems. Complementary examples in the case of 4D processing are given in Figs. 3 and 4.

The script shown in Fig. 3 converts and processes a 4D spectrum in three passes, using only 2D in-memory transpose. In this case, the spectrometer format conversion, X-axis processing, and Y-axis processing are all performed in the first pass, the Z-axis is processed in the second pass, and the A-axis is processed in the third pass. The corresponding script in Fig. 4 performs the same processing, but it has been rearranged so that the spectrum is processed in only two passes by the addition of a 3D in-memory transpose function. The first pass performs the spectrometer format conversion and the processing for the X-, Y- and Z-axes. The A-axis is processed in the second pass. As these examples show, in-memory processing is achieved at the discretion of the user, simply by use of appropriate transpose functions. Only minor alteration of a given processing scheme is needed, and no reconfiguration or recompilation of the software is required. Instead, the transpose functions, like all other functions of the NMRPipe system, allocate suitable amounts of memory automatically.

Processing functions and options

The NMRPipe system utilizes a relatively small number of processing functions, but these are augmented by a variety of modes and options; the processing functions listed in Table 1 and in the Appendix include over 300 options and parameters. For example, the functions

```
bruk2pipe -in ser $ARGS                                  \  Convert Bruker Format
| nmrPipe -fn SP -off 0.35 -end 0.95 -pow 2 -c 0.5 \  Window, Scale 1st Point
| nmrPipe -fn ZF -size 512                               \  Zero Fill
| nmrPipe -fn FT -di                                     \  Fourier Transform
| nmrPipe -fn YTP                                        \  2D Transpose X/Y
| nmrPipe -fn SP -off 0.35 -end 1.0 -pow 1  -c 0.5 \  Window, Scale 1st Point
| nmrPipe -fn ZF -size 128                               \  Zero Fill
| nmrPipe -fn FT  -di                                    \  Fourier Transform
| nmrPipe -fn ZTP                                        \  3D Transpose X/Z
| nmrPipe -fn SP -off 0.35 -end 0.95 -pow 1 -c 1.0 \  Window
| nmrPipe -fn ZF -size 64                                \  Zero Fill
| nmrPipe -fn FT -di                                     \  Fourier Transform
| pipe2xyz -out ft/noe%02d%03d.DAT -z                       Write Vectors to Z-Axis

xyz2pipe -in ft/noe%02d%03d.DAT -a -verb                 \  Read Vectors from A-Axis
| nmrPipe -fn SP -off 0.35 -end 0.95 -pow 1 -c 1.0 \  Window
| nmrPipe -fn ZF -size 64                                \  Zero Fill
| nmrPipe -fn FT -di                                     \  Fourier Transform
| pipe2xyz -out ft/noe%02d%03d.DAT -a -inPlace              Write Vectors to A-Axis
```

Fig. 4. Annotated 4D format conversion and processing script for a 256* × 64* × 16* × 16* point 4D ^{13}C-^{13}C correlated ^{1}H-^{1}H NOE FID, illustrating the use of both 2D and 3D transpose. Acquisition parameters have been abbreviated by $ARGS and phase correction steps have been omitted to save space. This scheme performs the same processing as the script shown in Fig. 3, but in this version, a 3D in-memory transpose is used to avoid saving one of the intermediate results. The size of the final result is 512 × 128 × 64 × 64 points. Processing time: 7 h and 55 min on a Sun Sparc 10 workstation.

POLY (polynomial fitting) and LP (linear prediction) each have a wide collection of parameters, which allows them to perform many tasks. The POLY function can be used as a solvent filter in the time domain, as well as for manual or automated correction according to a reliable in-house algorithm, and the corrections can be limited to selected spectral regions if desired. The linear prediction function LP can be used to predict points in either the start, end, or interior of existing data, in backward, forward or mixed forward–backward mode, with or without mirror-image methods and root reflection. In addition to this flexibility, the LP function has also been implemented using a matrix inversion procedure instead of the iterative (and often unstable) root-searching approach, making it especially robust (G. Zhu and A. Bax, unpublished results).

The NMRPipe processing functions make extensive use of default parameter settings. This helps to make argument lists more concise, since individual parameters can be adjusted while leaving default settings intact. For example, when used with no other arguments, LP will apply linear prediction and root reflection with eight complex coefficients to extend the original data to twice its size. The number of coefficients (the LP order) can be changed via the **-ord** option, and the number of predicted points can be changed independently via the **-pred** parameter. Mirror-image LP can be selected simply by adding either flag **-ps0-0** or **-ps90-180** to any LP command line, depending on whether data have no acquisition delay, or a half-dwell delay.

Many of the functions exploit or update the spectral header parameters during processing. For example, apodization, zero-filling, and phase correction details are recorded, and chemical shift calibrations can be updated automatically by any function that extracts or shifts the data. The functions also keep track of the valid time-domain size of the data, as influenced by time-domain shifts or frequency-domain extractions. Where appropriate, parameters can be specified in ppm or Hz as well as in points.

Inverse processing

Multidimensional enhancement schemes commonly call for inverse processing, so several functions have been implemented with an inverse mode for convenience. For instance, window functions support an inverse mode that divides by the window function, and zero-filling supports an inverse mode that strips away previous zero padding. These conveniences make it possible to construct complicated inverse processing protocols concisely, and if parameters are selected appropriately, the original data can commonly be recovered to a precision of better than one part in 10^5. Examples are given in Figs. 5 and 6, which show forward/inverse processing scripts for applying linear prediction and Maximum Entropy reconstruction in the two indirectly detected dimensions of a 3D spectrum.

In the case of the LP scheme in Fig. 5, forward and inverse processing is used to minimize the number of signals that must be predicted in any given vector in order to increase the prediction's stability and incidentally decrease the time required (Kay et al., 1991). In the case of the MEM scheme in Fig. 6, forward and inverse processing is used to allow a more stable automated base-line correction by using data processed with window functions, before data is reprocessed without window functions for Maximum Entropy reconstruction.

New capabilities and data formats

One of the special advantages of the pipeline approach is the ease and flexibility with which new capabilities and data formats can be implemented. The primary data format of the NMRPipe system consists of one or more 2D file planes, each with a 2048-byte header, followed by four-byte floating-point spectral data values in a sequential order. Other multidimensional data formats can be adapted simply by use of alternative programs to read or write data at the head or tail of a pipeline; the submatrix

```
xyz2pipe -in fid/cbcanh%03d.fid -x -verb          \   Read Vectors from X-Axis
| nmrPipe -fn POLY -time                          \   Solvent Filter
| nmrPipe -fn SP -off 0.4 -end 0.98 -pow 2 -c 0.5 \   Window, Scale 1st Point
| nmrPipe -fn ZF -auto                            \   Zero Fill
| nmrPipe -fn FT                                  \   Fourier Transform
| nmrPipe -fn PS -p0 125 -p1 0 -di                \   Phase, Delete Imaginaries
| nmrPipe -fn EXT -x1 10.3ppm -xn 5.9ppm -sw      \   Extract Amide Region
| pipe2xyz -out ft/cbcanh%03d.ft3 -x                  Write Vectors to X-Axis

xyz2pipe -in ft/cbcanh%03d.ft3 -z -verb          \   Read Vectors from Z-Axis
| nmrPipe -fn SP -off 0.4 -end 0.95 -pow 1        \   Window
| nmrPipe -fn ZF -auto                            \   Zero Fill
| nmrPipe -fn FT                                  \   Fourier Transform
| nmrPipe -fn PS -p0 -90 -p1 180 -di              \   Phase Correct
| pipe2xyz -out ft/cbcanh%03d.ft3 -z -inPlace         Write Vectors to Z-Axis

xyz2pipe -in ft/cbcanh%03d.ft3 -y -verb          \   Read Vectors from Y-Axis
| nmrPipe -fn LP -ps90-180 -ord 16                \   Mirror-Image LP
| nmrPipe -fn SP -off 0.4 -end 0.98 -pow 1        \   Window
| nmrPipe -fn ZF -auto                            \   Zero Fill
| nmrPipe -fn FT                                  \   Fourier Transform
| nmrPipe -fn PS -p0 -90 -p1 180 -di              \   Phase, Delete Imaginaries
| pipe2xyz -out ft/cbcanh%03d.ft3 -y -inPlace         Write Vectors to Y-Axis

xyz2pipe -in ft/cbcanh%03d.ft3 -z -verb          \   Read Vectors from Z-Axis
| nmrPipe -fn HT -auto                            \   Hilbert Transform
| nmrPipe -fn PS -inv -hdr                        \   Undo Previous Phase
| nmrPipe -fn FT -inv                             \   Inverse Fourier Transform
| nmrPipe -fn ZF -inv                             \   Undo Previous Zero Fill
| nmrPipe -fn SP -inv -hdr                        \   Undo Previous Window
| nmrPipe -fn LP -ps90-180 -ord 8                 \   Mirror-Image LP
| nmrPipe -fn SP -off 0.4 -end 0.98 -pow 1        \   Window
| nmrPipe -fn ZF -auto                            \   Zero Fill
| nmrPipe -fn FT                                  \   Fourier Transform
| nmrPipe -fn PS -hdr -di                         \   Rephase
| pipe2xyz -out ft/cbcanh%03d.ft3 -z -inPlace         Write Vectors to Z-Axis
```

Fig. 5. Annotated 3D processing script for amide-detected data, illustrating the use of inverse processing features in a linear prediction scheme. The scheme took 4 h and 55 min to perform on a Sun Sparc 10 workstation with a 3D CBCA(CO)NH FID of 512* × 52* × 32* points. The result is based on an intermediate amide proton dimension size of 1024 points, yielding a 3D spectrum of 299 × 256 × 128 points after extraction of the amide proton region and deletion of imaginary data. In the scheme, LP is used on the indirectly detected Y-axis and Z-axis of the data. This scheme is arranged so that when LP is applied to double the size of a given dimension, the other dimensions have been completely processed with a window function, zero-filling, and phasing. This localizes the signals as much as possible in the other dimensions and thus simplifies the signal content of the dimension to be predicted (Kay et al., 1991). In the scheme, the X-axis is processed in the first pass, the Z-axis is processed in the second pass, the Y-axis is extended via LP and processed in the third pass, and the Z-axis is inverse-processed, extended via LP, and reprocessed in the fourth pass.

```
xyz2pipe -in fid/noe%03d.fid -x -verb                \ Read Vectors from X-Axis
| nmrPipe -fn SOL                                    \ Solvent Filter
| nmrPipe -fn SP -off 0.35 -end 0.99 -pow 2 -c 0.5   \ Window, Adjust 1st Point
| nmrPipe -fn ZF -auto                               \ Zero Fill
| nmrPipe -fn FT                                     \ Fourier Transform
| nmrPipe -fn PS -p0 0.0 -p1 0.0 -di                 \ Phase, Delete Imaginaries
| nmrPipe -fn EXT -x1 5ppm -xn 10.5ppm -sw           \ Extract Amide Region
| nmrPipe -fn TP                                     \ 2D Transpose X/Y
| nmrPipe -fn ZF -zf 2 -auto                         \ Zero Fill Twice
| nmrPipe -fn RS -rs  1 -sw                          \ Right-Shift (1-dwell Delay)
| nmrPipe -fn SP -off 0.45 -end 0.95 -pow 1          \ Window
| nmrPipe -fn FT -di                                 \ Fourier Transform
| nmrPipe -fn POLY -auto -ord 0                      \ Auto Baseline Correct
| pipe2xyz -out ft/noe%03d.ft3 -x                      Write Vectors to X-Axis

xyz2pipe -in ft/noe%03d.ft3 -z -verb                 \ Read Vectors from Z-Axis
| nmrPipe -fn ZF -zf 2 -auto                         \ Zero Fill Twice
| nmrPipe -fn RS -rs  1 -sw                          \ Right-Shift (1-dwell Delay)
| nmrPipe -fn SP -off 0.35 -end 0.95 -pow 1          \ Window
| nmrPipe -fn FT -di                                 \ Fourier Transform
| nmrPipe -fn POLY -auto -ord 0                      \ Auto Baseline Correct
| nmrPipe -fn HT                                     \ Hilbert Transform
| nmrPipe -fn FT -inv                                \ Inverse Fourier Transform
| nmrPipe -fn SP -inv -hdr                           \ Undo Previous Window
| nmrPipe -fn FT -di                                 \ Fourier Transform
| nmrPipe -fn TP                                     \ 2D Transpose X/Y
| nmrPipe -fn HT                                     \ Hilbert Transform
| nmrPipe -fn FT -inv                                \ Inverse Fourier Transform
| nmrPipe -fn SP -inv -hdr                           \ Undo Previous Window
| nmrPipe -fn FT -di                                 \ Fourier Transform
| nmrPipe -fn TP                                     \ 2D Transpose X/Y
| nmrPipe -fn MEM -ndim 2 -neg -zero -alpha 0.001    \ 2D MEM, +/- Mode with
          -xconv EM -xcQ1 20  -yconv EM -ycQ1 15     \  Deconvolution In Both
          -sigma 200 -freq                           \  Dimensions
| pipe2xyz -out ft/noe%03d.ft3 -z -inPlace            Write Vectors to Z-Axis
```

Fig. 6. Annotated 3D processing script for amide-detected data, illustrating the use of inverse processing features in a 2D Maximum Entropy Reconstruction scheme. The scheme took 16 h and 45 min to perform on a Sun Sparc 10 workstation for a 3D ^{15}N-NOE FID of 512* × 128* × 64* points. The result is based on an intermediate amide proton dimension size of 1024 points, yielding a 3D spectrum of 420 × 512 × 128 points after extraction of the amide proton region and deletion of imaginary data. In the scheme, 2D MEM is applied to planes in the indirectly detected Y-axis (^{1}H) and Z-axis (^{15}N) of the data, which were each acquired with a one-dwell delay. The scheme is arranged to temporarily reorder the data so that the MEM function is provided with a stream of data planes from the indirect dimensions (the original Y- and Z-axes). The indirect dimensions are first processed by right-shifting, Fourier processing, and automated zero-order base-line correction to compensate for the one-dwell-time acquisition delay; the Fourier processing includes use of window functions to increase the effectiveness of the automated base-line correction. The planes are then reprocessed so that they are presented for Maximum Entropy reconstruction already phased, base-line corrected, and extensively zero-filled, but transformed without any window functions. Additional argument details are given in the Appendix.

formats of the powerful spectral analysis programs NMRView (Johnson and Blevins, 1994) and ANSIG (Kraulis, 1989; Kraulis et al., 1994) have been accommodated by their authors in this way. To facilitate work of this kind, the standard NMRPipe installation includes C source code for the spectrometer format conversion programs, file header interpretation and general I/O utilities, as well as the multidimensional I/O programs **xyz2pipe** and **pipe2xyz**.

New processing functions can be implemented as simple UNIX filter programs, which can be inserted directly in the pipeline data stream without the need to alter the **nmrPipe** program itself. As an alternative to writing a complete program, nmrPipe includes the MAC function, a macro interpreter that implements a subset of the C programming language, augmented with a variety of vector processing commands. The interpreter was implemented primarily for development purposes, using the

UNIX compiler generator Yacc (Johnson, 1986). The macro language allows direct manipulation of the data points, and the possibility to control the details of file I/O during processing. In its default mode, the MAC function will apply the contents of a user-written macro to every 1D vector in the given dimension, so that new functions can be implemented simply by placing a list of vector functions or other processing steps in a text file. This provides a convenient way to prototype new processing applications. For example, special processing steps for drift correction, gradient-enhanced data (Cavanagh et al., 1991; Palmer et al., 1991; Kay et al., 1992) and Bruker DMX digitally oversampled data have been developed this way.

Parallel processing

Many possible approaches can be envisioned for performing a multidimensional processing task in parallel over a network of computers or on a multi-CPU machine. By modifying only the multidimensional I/O programs (**xyz2pipe** and **pipe2xyz**), we have implemented one simple but broadly applicable approach, which relies only on standard UNIX network file sharing, and avoids the need for special machine-specific parallel compiling or configuration of software. This particular implementation uses static load balancing, which means that the amount of data to be processed by each computer is fixed at the outset of a task, and therefore there is no compensation for possible changes in CPU performance during the course of a calculation. In practice, the user performs parallel processing by creating a single script that processes a complementary subset of a complete spectrum, depending on which computer is used to execute it; the same script is then executed simultaneously on all CPUs involved. The division of data is performed automatically according to a user-supplied list of computers and their approximate relative speeds, so that only minor modification of an ordinary scheme is needed to convert it to a parallel scheme.

Graphical interface

As noted by Güntert et al. (1992), it is a difficult task to create and maintain a single, integrated spectral graphics and processing program. Nevertheless, in our experience we have found it essential to be able to graphically inspect the FID data, to interactively choose processing parameters, and to examine intermediate processing results on the workstation screen or in hard copy. In an attempt to meet these needs, we have developed a supplemental graphics interface called NMRDraw, using the X11 network graphics library and the XView graphical interface toolkit (Heller and Van Raalte, 1993). The program, shown in Fig. 7, currently runs on Sun, SGI, and IBM RS6000 UNIX workstations.

The NMRDraw program provides facilities for inspecting raw and processed data via 1D and 2D slices or projections from all dimensions, as well as a macro editor for creating and executing complete multidimensional processing scripts. NMRDraw also allows real-time display and interactive phasing of an arbitrary number of 1D slices selected from any dimension of the spectrum and displayed simultaneously. Interactive 1D processing is performed via program-controlled pipelines to **nmrPipe**, providing the functionality of both graphics and processing without the need to incorporate the two in a single program. In keeping with the philosophy of well-separated applications, the data extraction and display facilities of NMRDraw can also be operated remotely by two-way pipelines to other programs, in order to construct graphical spectral analysis schemes. A prototype example of this approach, modeled after the NMRView spectral analysis package (Johnson and Blevins, 1994), is shown in Fig. 8.

Independently of our graphics interface development, spectroscopists at a test site for the NMRPipe system have used the TCL graphics command language to create interactive **nmrPipe** schemes (N. Tjandra, private communication). TCL provides a method to build graphics applications using shell scripts alone, without the need to write, compile, and link a complete program (Ousterhout, 1994). Since TCL provides an easy method for building

TABLE 2
3D PROCESSING TIMES ON VARIOUS WORKSTATIONS FOR A 512*×64*×32* POINT HNCO FID PROCESSED BY THE SCRIPT GIVEN IN FIG. 2[a]

Computer type	Time (s)
SGI Challenge, 4 R4400 CPUs[b]	154
SGI Challenge, 4 R4400 CPUs[c]	187
HP 9000/755	239
SGI Indigo	408
DEC Alpha 3000[d]	487
SGI Challenge, 1 R4400 CPU[e]	525
Sun Sparc 10	644
IBM RS6000/530	1128
Sun Sparc 2	1208
Sun Sparc 1	1864
Convex C3830[f]	2146

[a] Times reported are actual times elapsed. No special attempt was made to vectorize or parallelize the code; only ordinary optimizing compilers were used. During processing, each axis size was doubled by zero-filling, yielding a spectrum of 417×128×64 points after extraction of the amide proton region and deletion of imaginary data.
[b] This time is based on a distributed version of the processing script, which divides each processing task into four equal parts, one for each CPU.
[c] This time is based on an ordinary version of the processing script, whose components are distributed automatically between CPUs by the operating system because they are separate programs.
[d] This version of the software was compiled with a four-byte floating-point compatibility mode, which is roughly half as fast as the best speed of the CPU.
[e] This time is based on execution of the script on a single CPU.
[f] This time was measured under heavy loading (44 users).

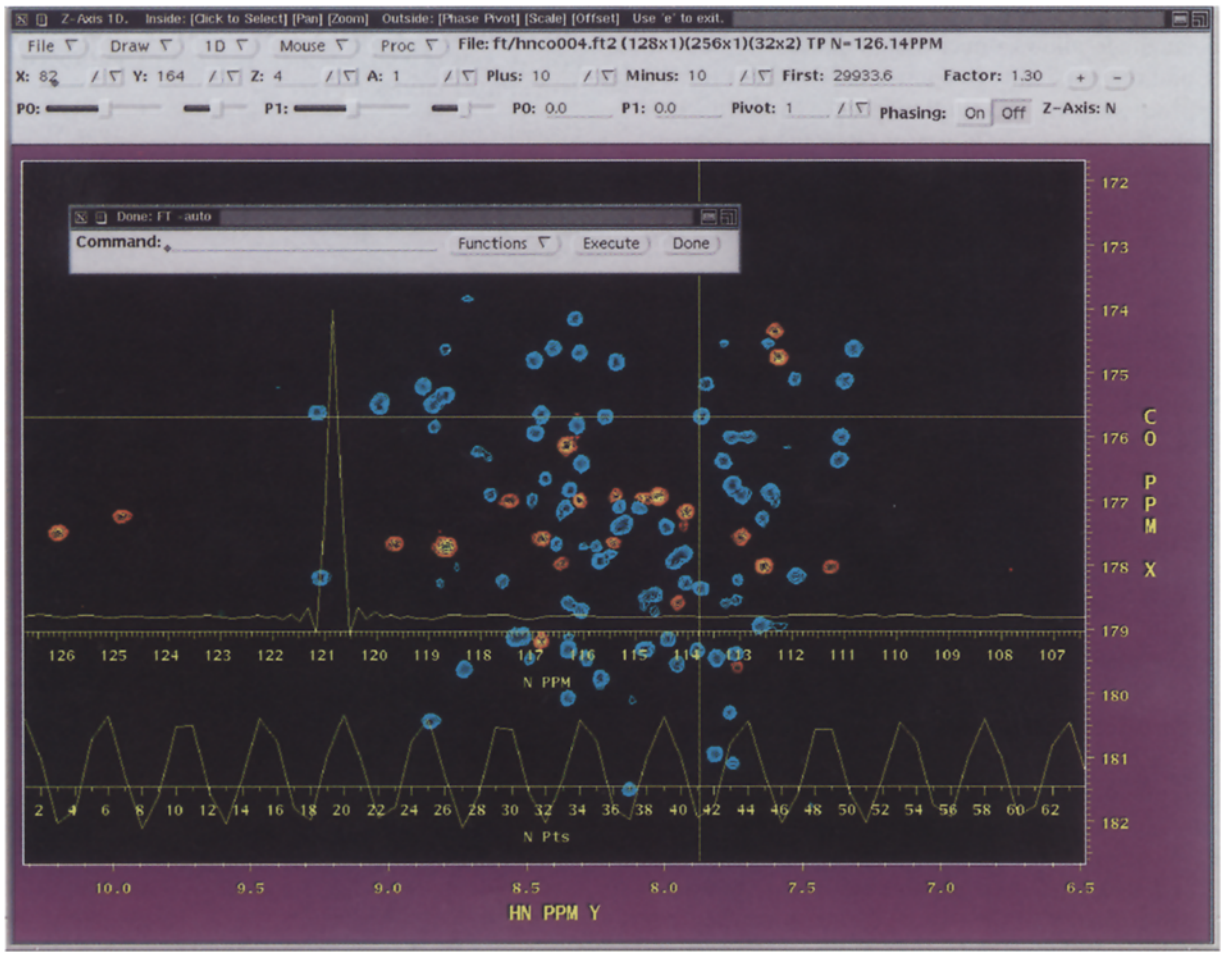

Fig. 7. The NMRDraw graphical processing and analysis interface, illustrating interactive processing of a 1D vector extracted from the Z-axis of a 3D interferogram. The topmost border of the program window describes the current functions of the mouse buttons. The command panel along the top contains graphical tools for executing commands, selecting the region of data to view, setting contour parameters, and adjusting phase values. The 2D contour display shows the fourth transformed $H^N/^{13}CO$ plane from a partially transformed HNCO spectrum (Z-axis (^{15}N) data is still in the time domain), with positive data drawn in a continuous range of blue colors, and negative data in a range of red colors. The small window over the contour display at the top left is a pop-up command area for entering **nmrPipe** processing commands. The cross-hair superimposed over the contour display shows the user-selected location for extraction of the Z-axis 1D vector. The time-domain vector itself, drawn along the bottom of the display, is shown after interactive extension via linear prediction. The Fourier-processed version of the vector, also prepared interactively, is drawn above the 1D time-domain data.

graphical applications at the UNIX shell script level, it is ideal for use with NMRPipe schemes, which also operate at the shell script level. Using this approach, it was possible to create a graphical interface that provides routine format conversion and processing without the requirement for users to edit shell scripts directly.

Companion software

In addition to the processing and display facilities described above, the NMRPipe system includes several other applications, such as algebraic combination of spectra, simulation of time-domain or frequency-domain data from peak tables, multidimensional nonlinear least-squares modeling of spectral line shapes, general-purpose functional fitting with Monte Carlo error estimation, and Principal Component Analysis. Stand-alone functions for examining and adjusting spectral header parameters are also included. Processed data from the NMRPipe system can be used directly with the PIPP/CAPP system for computer-assisted spectral analysis (Garrett et al., 1991); together, these software systems have been used to help generate roughly 10% of the NMR structures deposited in the Brookhaven Protein Databank since the beginning of 1994.

Results and Discussion

The NMRPipe system has been tested in over 50 laboratories, and has proven to be easy to use, robust, and thorough in its capabilities. In our direct experience, it is also more efficient than previous approaches we have tried, and it has successfully been adapted to new data formats and acquisition modes. Because of its design principles, it has been easy to port and maintain this

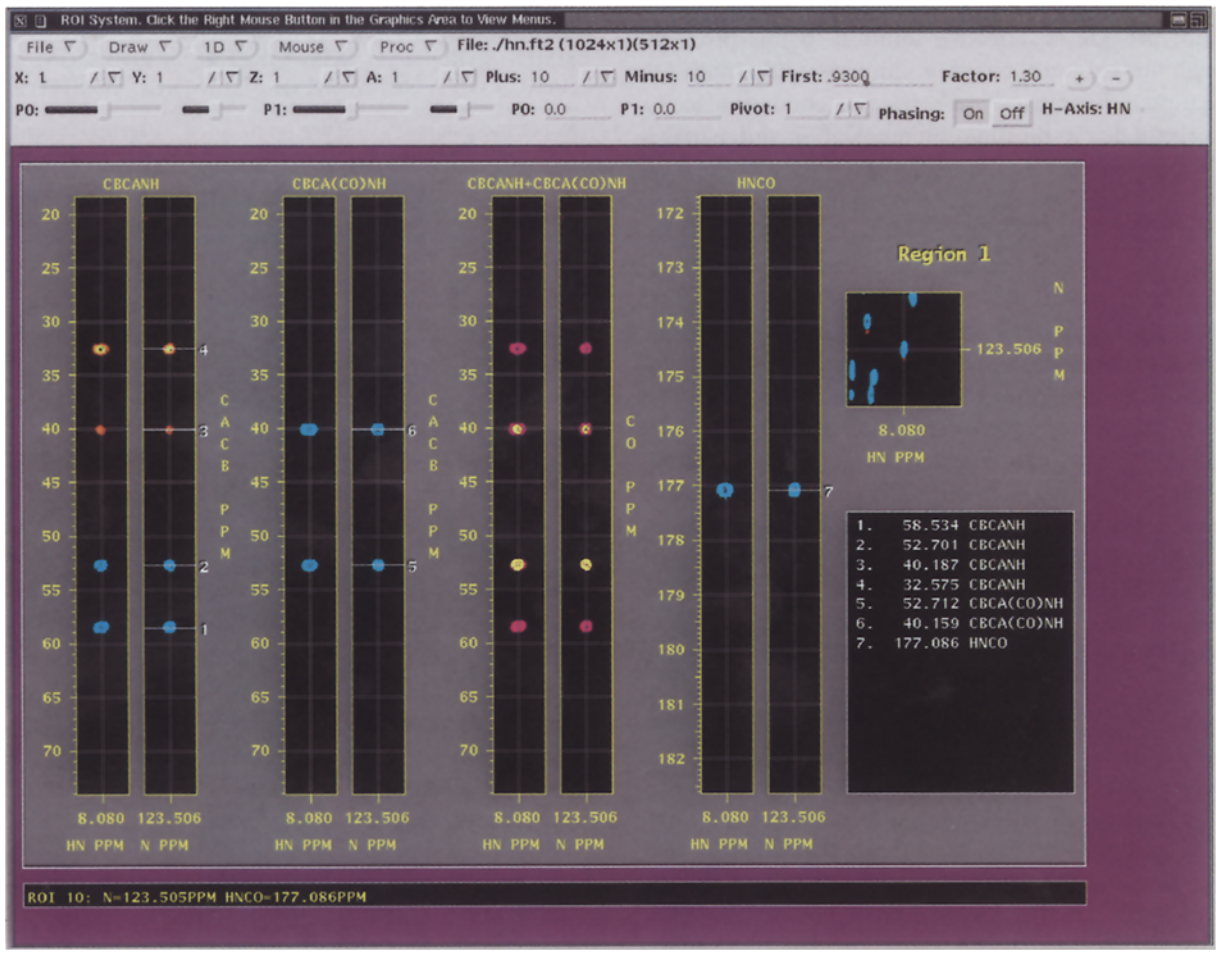

Fig. 8. The NMRDraw graphical processing and analysis interface, illustrating operation of the program's facilities by pipeline communication with a remote application, allowing separation of assignment and analysis programs and the graphics system. The remote application can be a program or a TCL script. Shown is a prototype application for browsing through strips from related amide-detected 3D experiments. In the application, the remote program decides what spectral regions and other graphics should be displayed, and transmits appropriate instructions to NMRDraw. In turn, NMRDraw transmits information about user input such as mouse clicks, so the remote program can respond to the user. The strips from a given spectrum are displayed in pairs showing orthogonal views at the given $^{1}H^{N}/^{15}N$ coordinate, and strips from related spectra can be overlaid to highlight corresponding signals if desired. In this illustration, the four pairs of strips displayed show data from a CBCANH spectrum, a CBCA(CO)NH spectrum, an overlay of CBCANH and CBCA(CO)NH spectra, and an HNCO spectrum. The square inset at the upper right displays the corresponding location from a 2D $^{1}H/^{15}N$ correlated spectrum, and the list at the lower right tabulates peak locations selected by the user via the mouse.

system on several different computer platforms, and to coordinate it with a variety of graphics and analysis systems.

Processing times on various computers for a typical 3D application are given in Table 2, and times for some other applications are given in the legends of Figs. 3–6. The main source of performance overhead in these examples is due to the multiplane data format and to pipeline communication. We decided to use the multiplane format in order to accommodate preexisting software that also used this format. While the format has the advantage of simplicity, it is not necessarily the best choice in all respects, especially for 4D data, since the number of file planes can become very large and relatively inefficient to manipulate. But, since the source and destination formats are independent of the processing pipeline itself, other formats could easily be implemented, for instance by substituting the programs that read and write multiplane format data by programs that read and write submatrix format data. In this respect, the processing pipeline can be thought of as a format-independent processing engine.

The overhead due to data format, while measurable, is not important in many cases. For example, consider the processing times for two versions of 4D processing given in Figs. 3 and 4. The version in Fig. 4 is 25 min faster than that in Fig. 3, because it avoids one intermediate read/write of the 4D data. However, this improvement amounts to only a 5% decrease in the overall processing time. This also suggests that an all-in-memory approach such as the one employed by PROSA (Güntert et al., 1992) is not always an advantage, since the performance gain will often be small, but the physical memory requirements (> 1024 Mb in this case) may constitute a serious obstacle.

As noted by Levy et al. (1986), use of virtual memory does not provide an effective solution to this problem, although in years to come, computers with multi-Gb physical memory capacity may become commonplace.

Overhead due to pipeline communication and management is an intrinsic aspect of the NMRPipe system. This overhead is examined in Fig. 9. As shown, the overhead time increases roughly linearly with the number of programs in the pipeline. For the Sun Sparc 10 workstation, this overhead contributes about 2 min to a typical 3D processing scheme. This amounts to about 15% of the time used for ordinary Fourier processing, and an insubstantial percentage for linear prediction applications.

A distinct performance advantage of the NMRPipe system is the ease with which processing tasks can be distributed over more than one CPU or workstation. The processing scripts themselves are naturally parallel, since they consist of several programs running simultaneously. Thus, as shown in Table 2, an ordinary NMRPipe scheme can show speed improvements on a multi-CPU computer without the need for special machine-specific compiling or vectorization, since the various programs in the script will be distributed at the discretion of the operating system. In the case shown for the four-CPU SGI Challenge, this simple approach yielded a 70% parallel efficiency compared to the same scheme executed on one CPU. In addition, the facilities of the NMRPipe system allow a processing task to be explicitly distributed by the user, an approach that yields even better performance, and still avoids the need for machine-specific optimization. An example is given in Table 3, which shows the results of a network-distributed processing application, with an efficiency of over 90% on five SGI workstations.

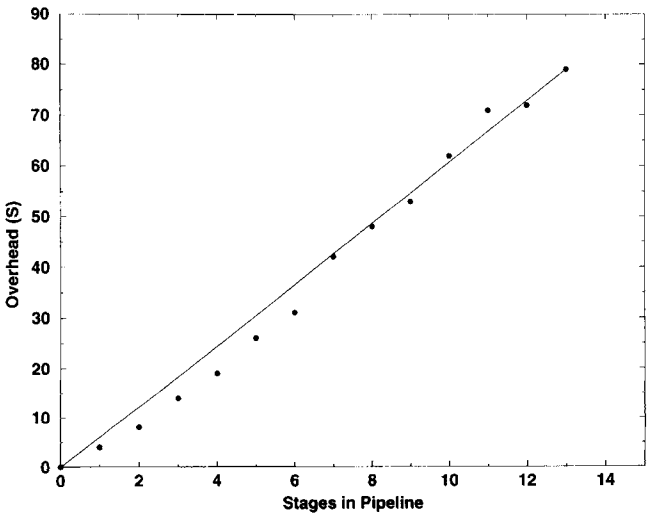

Fig. 9. Overhead processing time due to pipeline communication and management for a 32 Mb data set measured on a Sun Sparc 10 workstation. As shown, the overhead time increases roughly linearly with increasing numbers of functions in the pipeline. In this case, the best fit least-squares line, also shown, represents an overhead of 0.19 s/Mb for each additional stage in the pipeline.

TABLE 3
NETWORK-DISTRIBUTED PARALLEL PROCESSING TIMES FOR A Z-AXIS LINEAR PREDICTION APPLICATION ON A NETWORK OF SGI INDIGO COMPUTERS[a]

No. of processors	Time (min)	Parallel efficiency[b] (%)
1	119	100
2	59	99
3	40	99
4	30	99
5	26	91

[a] An interferogram of 512×128×32* points was extended to 512×128×64* points by forward–backward LP with eight complex coefficients, and the result was doubled by zero-filling and Fourier processed. The processing task was divided equally on each computer involved.

[b] The parallel efficiency is computed assuming that the ideal increase in processing speed is proportional to the number of computers used.

Conclusions

The NMRPipe implementation of multidimensional spectral processing via UNIX pipes provides an approach that is comprehensive, easy to use, flexible, extensible, and efficient. It naturally accommodates parallel processing approaches, and encourages and supports use of well-separated applications for graphics and analysis. Since the NMRPipe approach is complementary to existing methods that rely on monolithic programs, its unique combination of advantages is likely to prove increasingly useful as biomolecular NMR continues to advance.

Acknowledgements

In the course of the past two years, many people have assisted in the development, evaluation, and refinement of the software system presented; for this invaluable assistance, the authors wish to thank M. Akutsu, S. Archer, D. Benjamin, R.A. Byrd, R.M. Clore, M. Donlan, N. Farrow, J. Forman-Kay, S. Gagne, D. Garrett, H. Grahn, A.M. Gronenborn, T. Harvey, H. Hatanaka, E. Henry, M. Ikura, Y. Ito, L.E. Kay, W. Klaus, J. Kordel, R. Martino, L. Nicholson, I. Pelczer, R. Powers, M. Shirakawa, S. Tate, N. Tjandra, H. Tsuda, T. Yamazaki, and T. Yamazaki. Thanks is also extended to A. Wang for critical reading of the manuscript. This work was supported in part by the AIDS Targeted Anti-Viral Program of the Office of the Director of the National Institutes of Health.

References

Barkhuijsen, H., De Beer, R., Bovée, W.M.M.J. and Van Ormondt, D. (1985) *J. Magn. Reson.*, **61**, 465–481.

Barkhuijsen, H., De Beer, R. and Van Ormondt, D. (1987) *J. Magn. Reson.*, **73**, 553–557.

Bax, A. and Grzesiek, S. (1993) *Acc. Chem. Res.*, **26**, 131–138.

Callaghan, P.T., MacKay, A.L., Pauls, K.P., Soderman, O. and

Bloom, M. (1984) *J. Magn. Reson.*, **56**, 101–109.
Cavanagh, J., Palmer, A.G., Wright, P.E. and Rance, M. (1991) *J. Magn. Reson.*, **91**, 429–436.
Delsuc, M.A., Ni, F. and Levy, G.C. (1987) *J. Magn. Reson.*, **73**, 548–552.
Delsuc, M.A. (1989) *Maximum Entropy and Bayesian Methods*, Kluwer, Amsterdam.
Friedrichs, M.S. (1995) *J. Biomol. NMR*, **5**, 147–153.
Garrett, D.S., Powers, R., Gronenborn, A.M. and Clore, G.M. (1991) *J. Magn. Reson.*, **94**, 214–220.
Gull, S.F. and Daniell, G.J. (1978) *Nature*, **272**, 686–690.
Güntert, P., Doetsch, V., Wider, G. and Wüthrich, K. (1992) *J. Biomol. NMR*, **2**, 619–629.
Heller, D. and Van Raalte, T. (1993) *XView Programming Manual*, O'Reilly and Associates, Inc., Sebastopol, CA.
Hoch, J.C. (1985) *Rowland Institute for Science Technical Memorandum RIS-18t*, Rowland Institute, Cambridge, MA.
Hoch, J.C. (1989) *Methods Enzymol.*, **176**, 216–241.
Hoch, J.C., Stern, A.S., Donoho, D.L. and Johnstone, I.M. (1990) *J. Magn. Reson.*, **86**, 236–246.
Hore, P.J. (1985) *J. Magn. Reson.*, **62**, 561–567.
Johnson, B. and Blevins, R.A. (1994) *J. Biomol. NMR*, **4**, 603–614.
Johnson, S. (1986) In *UNIX Programmer's Manual: Supplementary Documents 1*, University of California, Berkeley, CA.
Kauppinen, J. and Saario, E.K. (1993) *Appl. Spectrosc.*, **47**, 1123–1127.
Kay, L.E., Marion, D. and Bax, A. (1989) *J. Magn. Reson.*, **84**, 72–84.
Kay, L.E., Ikura, M., Zhu, G. and Bax, A. (1991) *J. Magn. Reson.*, **91**, 422–428.
Kay, L.E., Keifer, P. and Saarinen, T. (1992) *J. Am. Chem. Soc.*, **114**, 10663–10666.
Kernighan, B.W. and Pike, R. (1984) *The UNIX Programming Environment*, Prentice-Hall, Englewood Cliffs, NJ.
Kernighan, B.W. and Ritchie, D.M. (1988) *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ.
Kjaer, M., Andersen, K.V. and Poulsen, F.M. (1994) *Methods Enzymol.*, **239**, 288–307.
Kraulis, P.J. (1989) *J. Magn. Reson.*, **84**, 627–633.
Kraulis, P.J., Domaille, P.J., Campbell-Burk, S.L., Van Aken, T. and Laue, E.D. (1994) *Biochemistry*, **33**, 3515–3531.
Kumaresan, R. and Tufts, D.W. (1982) *IEEE Trans. Acoust. Speech Signal Process.*, **30**, 833–840.
Laue, E.D., Skilling, J. and Staunton, J. (1985a) *J. Magn. Reson.*, **63**, 418–424.
Laue, E.D., Skilling, J., Staunton, J., Sibisi, S. and Brereton, R. (1985b) *J. Magn. Reson.*, **62**, 437–452.
Laue, E.D., Mayger, M.R., Skilling, J. and Staunton, J. (1986) *J. Magn. Reson.*, **68**, 14–29.
Levy, G.C., Delaglio, F., Macur, A. and Begemann, J. (1986) *Comput. Enhanced Spectrosc.*, **3**, 1–12.
Marion, D. and Wüthrich, K. (1983) *Biochem. Biophys. Res. Commun.*, **113**, 967–974.
Marion, D., Ikura, M. and Bax, A. (1989a) *J. Magn. Reson.*, **84**, 425–430.
Marion, D., Ikura, M., Tschudin, R. and Bax, A. (1989b) *J. Magn. Reson.*, **85**, 393–399.
Mazzeo, A.R., Delsuc, M.A., Kumar, A. and Levy, G.C. (1989) *J. Magn. Reson.*, **81**, 512–519.
Meadows, R.P., Olejniczak, E.T. and Fesik, S.W. (1994) *J. Biomol. NMR*, **4**, 79–96.
Ni, F. and Scheraga, H.A. (1986) *J. Magn. Reson.*, **70**, 506–511.
Ni, F., Levy, G.C. and Scheraga, H.A. (1986) *J. Magn. Reson.*, **66**, 385–390.
Olejniczak, E.T. and Eaton, H.L. (1990) *J. Magn. Reson.*, **87**, 628–632.
Ousterhout, J.K. (1994) *TCL and the Tk Toolkit*, Addison-Wesley, Reading, MA.
Palmer, A.G., Cavanagh, J., Wright, P.E. and Rance, M. (1991) *J. Magn. Reson.*, **93**, 151–170.
Parks, S.I. and Johannesen, R.B. (1976) *J. Magn. Reson.*, **22**, 265–267.
Pelczer, I. and Szalma, S. (1991) *Chem. Rev.*, **91**, 1507–1524.
Redfield, A.G. and Kunz, S.D. (1975) *J. Magn. Reson.*, **19**, 250–254.
Schmieder, P., Stern, A.S., Wagner, G. and Hoch, J.C. (1994) *J. Biomol. NMR*, **4**, 483–490.
Sibisi, S. (1983) *Nature*, **301**, 134–136.
Skilling, J. and Bryan, R.K. (1984) *Mon. Not. R. Astr. Soc.*, **211**, 111–124.
States, D.J., Haberkorn, R.A. and Ruben, D.J. (1982) *J. Magn. Reson.*, **48**, 286–292.
Stephenson, M. (1988) *Prog. NMR Spectrosc.*, **20**, 515–626.
Stevens, W.R. (1992) *Advanced Programming in the UNIX Environment*, Addison-Wesley, Reading, MA, pp. 428–434.
Wu, N.L. (1984) *Astron. Astrophys.*, **139**, 555–557.
Zhu, G. and Bax, A. (1990) *J. Magn. Reson.*, **90**, 405–410.
Zhu, G. and Bax, A. (1992a) *J. Magn. Reson.*, **98**, 192–199.
Zhu, G. and Bax, A. (1992b) *J. Magn. Reson.*, **100**, 202–207.

Appendix

Description of selected processing modules and arguments

Generic arguments

The following is a list of arguments used by more than one program or function in the examples and figures.

-di deletes imaginary data from the current dimension after the given processing function is performed.

-hdr extracts parameters recorded during previous processing from the spectral header rather than the command line.

-in specifies the input file or file template (see 'Input and output templates' below).

-inPlace permits in-place processing, which is replacement of the input data by the output result.

-inv activates the inverse mode of a given function; function PS will apply inverse (negative) phase correction; function FT will perform an inverse Fourier transform; function ZF will undo any previous zero-filling; function SP will apply the inverse window function and first point scaling.

-out specifies the output file or file template (see 'Input and Output Templates' below).

-ov permits overwriting of any preexisting files.

-sw updates the sweep width and other ppm calibration information to accommodate an extraction or shift function.

-verb performs processing in verbose mode, with status messages.

Processing functions

The following is an alphabetical list of the **nmrPipe** processing functions used in the examples and figures. The functions and arguments described are not complete lists, but rather only those used in the examples.

EXT extracts a region from the current dimension with limits specified by the arguments **-x1** and **-xn**; the limits can be labeled in points, percent, Hz, or ppm. Alternatively, the left or right half of the data can be extracted with the arguments **-left** and **-right**.

FT applies a real or complex forward or inverse Fourier transform, with sign alternation or complex conjugation, as indicated by spectral parameters or command-line arguments.

HT performs a Hilbert transform to reconstruct imaginary data, choosing between ordinary and mirror-image mode if the argument **-auto** is used.

LP extends the data to twice its original size by default, using a complex prediction polynomial whose order is specified by argument **-ord**. Mixed forward–backward LP is performed if the **-fb** argument is used. Mirror-image LP for data with no acquisition delay is performed if the argument **-ps0-0** is used; mirror-image LP for data with a half-dwell acquisition delay is performed if the argument **-ps90-180** is used.

MEM applies Maximum Entropy reconstruction according to the method of Gull and Daniell (1978): argument **-ndim** specifies the number of dimensions to reconstruct, argument **-neg** activates the two-channel mode, for reconstruction of data with both positive and negative signals, argument **-zero** corrects the zero-order offset introduced during reconstruction, argument **-alpha** specifies the fraction of a given iterate that will be added to the current MEM spectrum, argument **-sigma** specifies the estimated standard deviation of the noise in the time domain, argument **-freq** produces the final MEM result in the frequency domain, arguments **-xconv** and **-yconv** specify the line-sharpening function, which in Fig. 6 is EM (Exponential Multiplication) for both dimensions, and arguments **-xcQ1** and **-ycQ1** specify the corresponding line-sharpening parameters, which in Fig. 6 are 20 Hz and 15 Hz for the ^{15}N and ^{1}H dimensions, respectively. Other arguments can be used to optimize convergence speed, or to increase stability for reconstruction of data with high dynamic range.

POLY (frequency domain) applies a polynomial base-line correction of the order specified by argument **-ord**, via an automated base-line detection method when used with argument **-auto**. The default is a fourth-order polynomial. The automated base-line mode works as follows: a copy of a given vector is divided into a series of adjacent sections, typically eight points wide. The average value of each section is subtracted from all points in that section, to generate a 'centered' vector. The intensities of the entire centered vector are sorted, and the standard deviation of the noise is estimated under the assumption that a given fraction (typically about 30%) of the smallest intensities belong to the base-line, and that the noise is normally distributed. This noise estimate is multiplied by a constant, typically about 1.5, to yield a classification threshold. Then, each section in the centered vector is classified as base line only if its standard deviation does not exceed the threshold. These classifications are used to correct the original vector.

POLY (time domain), when used with the argument **-time**, fits all data points to a polynomial, which is then subtracted from the original data. It is intended to fit and subtract low-frequency solvent signal in the FID, a procedure that often causes less distortion than time-domain convolution methods. By default, a fourth-order polynomial is used. For speed, successive averages of regions are usually fit, rather than fitting all of the data points.

PS applies the zero- and first-order phase corrections as specified in degrees by the arguments **-p0** and **-p1**. The PS function applies no processing if these values are both zero; for this reason, a zero,zero phase correction step is commonly kept in a processing scheme for completeness, so that the scheme can be copied and reused more easily.

RS, when used in the time domain, applies a right-shift by the number of points specified by argument **-rs**, and updates the recorded time-domain size if the argument **-sw** is used.

SOL uses time-domain convolution and polynomial extrapolation to suppress solvent signal with a default moving average window of +/– 16 points.

SP applies a sine-bell window extending from $\sin^r(a\pi)$ to $\sin^r(b\pi)$ with offset a, end point b, and exponent r specified by arguments **-off**, **-end**, and **-pow**, and first-point scaling specified by argument **-c**. The default length is taken from the recorded time-domain size of the current dimension. By default, a = 0.0, b = 1.0, r = 1.0 (sine bell), and the first point scale factor is 1.0 (no scaling).

TP exchanges vectors from the X-axis and Y-axis of the data stream, so that the resultant data stream consists of vectors from the Y-axis of the original data. It is identical to YTP.

YTP is another name for the TP transpose function, which exchanges vectors from the X-axis and the Y-axis of the data stream. The alternative name is provided for contrast with the other transpose functions **ZTP** (X-axis/Z-axis transpose) and **ATP** (X-axis/A-axis transpose).

ZF pads the data with zeros; the amount of padding

can be specified by argument **-zf**, which defines the number of times to double the data size, or by the argument **-size**, which specifies the desired complex size after zero-filling. By default, the data size is doubled by zero-filling. Use of the argument **-auto** will cause the zero-fill size to be rounded up to the nearest power of two.

ZTP exchanges vectors from the X-axis and Z-axis of the data stream, so that the resultant data stream consists of vectors from the Z-axis of the original data.

Input and output templates

The following describes the method used to specify input and output data in the multifile 2D plane format.

3D File Name Templates: 3D data in the multifile 2D plane format is specified as a template, a single name that stands for a series of 2D file planes. The template includes a format specification, usually '%03d', which is substituted by the Z-axis plane number in the actual file names. The format specification is interpreted by rules of the C programming language; the '03d' in the template means that the plane number will be included as a zero-padded three-digit number, to give a series of names such as fid/noe001.fid, fid/noe002.fid, fid/noe003.fid, etc.

4D File Name Templates: 4D data in the multifile 2D plane format is specified as a template, a single name that stands for a series of 2D file planes. The template includes a format specification, usually '%02d%03d', which is substituted by the A-axis and Z-axis plane numbers in the actual file names. The format specification is interpreted by rules of the C programming language; the '02d' and '03d' in the template mean that the A-axis plane number will be included as a zero-padded two-digit number, followed by the Z-axis plane number as a zero-padded three-digit number.

Data input and output programs

In the following, programs are described that are used along with **nmrPipe** in the examples and figures. The arguments described are not complete lists, but rather only those used in the examples.

bruk2pipe converts binary data from various types of Bruker spectrometers to the nmrPipe data format. The related programs **var2pipe** and **bin2pipe** perform Varian Unity conversions and general-purpose binary conversions, respectively. The programs take as input a file or data stream in the binary spectrometer format, and produce a file, file series, or data stream in the NMRPipe format. The programs require a collection of arguments defining the acquisition parameters for each dimension, prefixed by **-x**, **-y**, **-z**, and **-a**. Following are the commonly required arguments: arguments **-xN** etc. define the total number of points saved in the input file for a given dimension; arguments **-xT** etc. define the number of valid complex points actually acquired, in case this differs from the number of points saved in the input file; arguments **-xMODE** etc. define the quadrature detection mode of the given dimension; arguments **-xSW** etc. define the full spectral width in Hz for the given dimension; arguments **-xOBS** etc. define the observe frequency in MHz for a given dimension, while arguments **-xCAR** etc. define the carrier position in ppm; arguments **-xLAB** etc. define unique axis labels; argument **-ndim** defines the number of dimensions in the input; argument **-aq2D** defines the type of 2D output file planes produced as either magnitude mode, States/States-TPPI, or TPPI.

pipe2xyz writes vectors from a data stream to the selected axis of nD data in the multiplane format. The arguments **-x**, **-y**, **-z**, and **-a** select the axis, and the argument **-out** is used to specify the output file series as a template (see 'Input and output templates' above). In order to write to a given axis, the program **pipe2xyz** performs rotations of the data complementary to those performed by **xyz2pipe**. This means that a pipeline that begins with **xyz2pipe** reading from a given dimension and ends with **pipe2xyz** writing to the same dimension will conserve the original data order if no transpose steps are included in-between.

xyz2pipe creates a data stream for multidimensional processing via pipeline by reading vectors from the selected axis of nD data in the multiplane format. The arguments **-x**, **-y**, **-z**, and **-a** select the axis, and the argument **-in** is used to specify the input file series as a template (see 'Input and output templates' above). Depending on the dimension selected, the other dimensions are reordered by a multidimensional rotation, which is similar, but not always identical, to a transpose. If the original order of dimensions is described as XYZA..., the relative reordering of data can be summarized as follows:

nmrPipe -fn TP	Exchange of the first two dimensions:	XYZA... to YXZA...
nmrPipe -fn ZTP	Exchange of the first and third dimensions:	XYZA... to ZYXA...
nmrPipe -fn ATP	Exchange of the first and fourth dimensions:	XYZA... to AYZX...
xyz2pipe -x	No change in data order:	XYZA... to XYZA...
xyz2pipe -y	Rotation of the first two dimensions (same as TP):	XYZA... to YXZA...
xyz2pipe -z	Rotation of the first three dimensions:	XYZA... to ZXYA...
xyz2pipe -a	Rotation of the first four dimensions:	XYZA... to AXYZ...