SCRUB and nmr_wash User's Manual

Brian E. Coggins

Version 1.0.0, November 2013

Contents

Introduction

SCRUB

SCRUB is an algorithm for the suppression of artifacts in spectra generated from sparsely sampled multidimensional NMR data. SCRUB is derived from the older CLEAN method introduced in the radioastronomy community in the 1970s. It achieves better artifact suppression than CLEAN while preserving signal lineshapes, intensities, and volumes.

For more information about SCRUB and artifact suppression with sparsely sampled data, please see:

> B.E. Coggins, J.W. Werner-Allen, A. Yan, and P. Zhou. "Rapid Protein Global Fold Determination Using Ultrasparse Sampling, High-Dynamic Range Artifact Suppression, and Time-Shared NOESY." *J. Am. Chem. Soc.*, **134**, 18619–18630 (2012)

The *nmr_wash* Suite of Tools

nmr_wash is a suite of command-line programs and a C++ library that carry out both SCRUB and CLEAN calculations. There are standalone *scrub* and *clean* programs for those two algorithms, as well as an NMRPipe plugin, *pipewash*, which allows one to access these algorithms from within an NMRPipe script as if they were built-in NMRPipe functions.

The *libnmr_wash.a* file is a C++ library providing classes for running SCRUB and CLEAN, along with support classes for manipulating sampling patterns, setting options, etc. In the future, we hope to provide a Python binding for this library.

How to process sparse data with NMRPipe and SCRUB: an overview

The *nmr_wash* programs are designed to fit nicely with the tools used in most biomolecular NMR labs for processing and analyzing data. The recommended procedure:

1. Process sparse data using NMRPipe.
    a. Convert from Varian or Bruker format using the normal *var2pipe* and *bruk2pipe* programs.
    b. Expand (de-sparse) the data using the *nusExpand.tcl* script. This copies the sparse data into a normal data file, placing a zero value at every position that was not measured.
    c. Process each dimension using standard NMRPipe commands. This will produce a frequency domain spectrum with artifacts.
2. Run SCRUB, either as part of the NMRPipe script or in a separate command, to remove artifacts. The standalone SCRUB program can save its output in any of the common NMR file formats (NMRPipe, NMRView, Sparky, XEASY), while the NMRPipe plugin can be incorporated into the middle of an existing processing script for conventional data.

Installation

The easiest way to obtain *nmr_wash* is to download a TAR archive containing precompiled binaries for your platform. After downloading this archive, simply unpack it and look for the binaries in the `bin` subdirectory. You may want to copy them to a location on your path.

If you did not download precompiled binaries, or if precompiled binaries are not available for your platform, you will need to compile the programs on your system. The *nmr_wash* distribution includes all the libraries that are needed other than basic system libraries, so building the programs should be relatively straightforward as long as your system is equipped with basic build tools such as gcc. Instructions are given in the `INSTALL.txt` file.

The distribution includes the following:

Subdirectory	Contents
`bin`	Precompiled binaries
`build`	Created by the make process if you compile the software on your own system. Contains the object files generated during compilation. The final products (*scrub*, *clean*, etc.) will be placed in the main distribution directory.
`doc`	Documentation
`include`	C++ header files for the *nmr_wash* library
`libs`	External libraries required to compile this software
`src`	C++ source code

Sample data are not included in the main distribution, but will be released on our website as separate downloads.

Filing Bug Reports and Getting Additional Help/Information

Questions? Problems? Bugs to report? Please visit our website at:

```
http://coggins.biochem.duke.edu/scrub
```

or contact the program author, Brian E. Coggins, by email to bec2 AT duke.edu.

Preparing Your Data Using NMRPipe

Initial Conversion with *var2pipe* or *bruk2pipe*

The initial conversion of sparsely sampled data from Varian or Bruker format to NMRPipe format is very similar to the conversion used with conventional data, but several of the flags must be set differently. In particular:

Parameter	Set to
ndim	`2`
nusDim	*the actual number of dimensions*
aq2D	`States`
yN	*the number of sampling points* × *the number of complex components per point*
yT	*the number of points on the y-axis of the sampling grid*
yMODE	`Complex`
zN	`1`
zT	*the number of points on the z-axis of the sampling grid*
zMODE	`Real`
aN	`1` *(when applicable)*
aT	*the number of points on the a-axis of the sampling grid (when applicable)*
aMODE	`Real` *(when applicable)*

All other parameters should be set as in a normal experiment.

If Rance-Kay encoding was used in any of the indirect dimensions, it will be necessary to use one of the macros in the `$NMRTXT` directory of your NMRPipe installation to decode it. For example, for Varian data one would use the `var_ranceN.M` macro:

```
nmrPipe -fn MAC -macro $NMRTXT/var_ranceN.M -noRd -noWr -var nShuf 1
```

while for Bruker data one would use `bruk_ranceN.M`. Note that the flags `-noRd` and `-noWr` are required for these macros to read and write their data correctly. The flag `-var nShuf 1` indicates that the data are in a 2-D format, as is true for sparse data prior to expansion.

The complete conversion block for a Varian experiment might look something like this, where the raw data are located in `raw/fid`:

```
var2pipe -in raw/fid -noaswap -ndim 2 -nusDim 3 -aqORD 0 -verb          \
                                          -title hnco -aq2D States    \
  -xN              1942  -yN               888   -zN               1    \
  -xT               971  -yT                64   -zT              64    \
  -xMODE        Complex  -yMODE        Complex   -zMODE         Real    \
  -xSW    16181.2297735  -ySW             1350   -zSW           1260    \
  -xOBS     599.7244735  -yOBS       150.80043   -zOBS      60.76955    \
  -xCAR           4.786  -yCAR         176.378   -zCAR       119.615    \
  -xLAB               H  -yLAB               C   -zLAB             N    \
| nmrPipe  -fn MAC -macro $NMRTXT/var_ranceN.M -noRd -noWr -var nShuf 1 \
  -out fid/hnco_nus.fid -ov
```

This is for a 3-D HNCO experiment recorded with 222 sampling points. *ndim* is set to 2 in all sparse experiments, while *nusDim* informs the conversion program that there will be 3 dimensions after expansion/de-sparsing. *xN*, *xT*, and *xMODE* are the direct dimension parameters and are set exactly as for a conventional experiment. The sampling pattern was constructed on a 64 × 64 grid, so *yT* and *zT* are set to 64. The total number of FIDs is 222 sampling points × 4 hyper-complex components per point = 888, which goes in *yN*. *zN* is set to 1. The Y and Z calibration parameters are set in the normal manner. Finally, the data are piped through the *var_ranceN.M* macro for Rance-Kay decoding before being written to an NMRPipe output file.

Note that the output file from this example is a single .fid file, and not a set as we might expect for a 3-D experiment. At this point the data are organized as a pseudo–2-D experiment, with X corresponding to direct dimension time points and Y corresponding to the list of sampling points in the sampling table.

De-Sparsing Data with *nusExpand.tcl*

A sparse dataset initially consists of a sequence of FIDs collected at the specific grid points listed in the sampling pattern. For NMRPipe to be able to process this with an FT, we must *expand* or *de-sparse* it, inserting zero values for the data points that were not measured and restructuring the file to have the correct number of dimensions and correct dimension sizes. This can be done using the TCL script *nusExpand.tcl*, which has been included in recent NMRPipe distributions.

Note: Some recent versions of *nusExpand.tcl* have known bugs interfering with 4-D data processing. Be sure to download the most recent release of NMRPipe before trying to process 4-D data.

A typical *nusExpand.tcl* command looks like this:

```
nusExpand.tcl -in fid/hnco_nus.fid -out fid/hnco_%03d.fid -sample pattern.txt -aqORD 1 -sign -hasW -multW
```

where `pattern.txt` is the sampling pattern file. The script should be able to parse most sampling patterns automatically. If the sampling pattern includes weights for the sampling points, use the *hasW* and *multW* flags to tell nusExpand.tcl to read and use these values. For more information on sampling pattern formats, see the section below on sampling patterns in SCRUB calculations.

The *aqORD* and *sign* flags are required depending upon the type of spectrometer and the pulse sequence.

Note that the input to this command is normally a single .fid file, while the output is a set of files, formatted as a normal

3-D or 4-D experiment.

Processing the Direct Dimension

The direct dimension is processed as in any other NMRPipe script. The fact that the other dimensions are sparse imposes no limitations on the processing functions that can be used.

Processing the Indirect Dimensions

Indirect dimensions are generally processed as they would be in a conventional experiment. If the sampling pattern is *not* weighted—that is, if the sampling points in the pattern were distributed uniformly—a window function should be used, but if it *is* weighted, with more sampling points at shorter evolution times and fewer points at longer evolution times, a window function should not be used. Linear prediction is not recommended for sparse data with artifacts still present.

First-point corrections are frequently needed in sparse indirect dimensions. If you are applying a window function, a first-point correction can be accomplished at the same time using `-c 0.5`. In cases where no window function is needed, a first-point correction can be applied using a window function that is, in fact, uniform. One way to achieve this is with an exponential window function with 0 Hz line-broadening:

```
nmrPipe -fn EM -lb 0 -c 0.5
```

and another way is with a sinebell function that starts and ends at π/2:

```
nmrPipe -fn SP -off 0.5 -end 0.5 -c 0.5
```

Finishing Up, and Feeding Data to SCRUB

After all dimensions have been processed, one can feed the data to SCRUB in one of two ways:

- Using the standalone program. In this case, end the NMRPipe script as normal, writing a set of planes with one or two `%03d` fields and with FT3 or FT4 file extensions.

  For very large 4-D datasets, it is advisable to transpose the dimensions so that the direct dimension is in the A or Z position before providing the data to the *scrub* program, as this reorganization speeds up data access and reduces the calculation time.

- Using the NMRPipe plugin. In this case, pipe the data into *pipewash* and choose the appropriate processing function, then continue the NMRPipe script to output the final data.

  In many cases, it will be necessary to include data transposition functions immediately before (and most likely after) calling the *pipewash* processing function, as described below.

Both methods are explained below.

Examples of Complete Processing Scripts

An Example Script for Varian 3-D Data. Here is a complete example of a processing script for a sparsely sampled 3-D HNCO dataset. The initial conversion and expansion steps are described above. The X dimension processing includes a solvent subtraction filter, a window function, a phase correction, and zero-filling/extraction commands to pull out the relevant part of the spectrum at the desired digital resolution. In the indirect dimensions, the data are zero-filled to the desired size, first-point corrections are applied, and a phase correction is used in the Y dimension. Note that window functions are *not* used as the sampling pattern for this experiment was already weighted according to a window function, but first-point corrections are achieved using calls to the NMRPipe apodization processing functions.

```
# Convert data from Varian format.
var2pipe -in raw/fid -noaswap -ndim 2 -nusDim 3 -aqORD 0 -verb          \
                                        -title hnco -aq2D States    \
  -xN              1942  -yN               888   -zN               1    \
  -xT               971  -yT                64   -zT              64    \
  -xMODE        Complex  -yMODE        Complex   -zMODE         Real    \
  -xSW    16181.2297735  -ySW             1350   -zSW           1260    \
  -xOBS     599.7244735  -yOBS       150.80043   -zOBS      60.76955    \
  -xCAR           4.786  -yCAR         176.378   -zCAR       119.615    \
  -xLAB               H  -yLAB               C   -zLAB             N    \
| nmrPipe  -fn MAC -macro $NMRTXT/var_ranceN.M -noRd -noWr -var nShuf 1 \
  -out fid/hnco_nus.fid -ov

# Expand sparse data
nusExpand.tcl -in fid/hnco_nus.fid -out fid/hnco_%03d.fid -sample pattern.txt \
  -aqORD 1 -sign -hasW -multW

# Process X (direct) dimension
xyz2pipe -in fid/hnco_%03d.fid -x -verb               \
| nmrPipe  -fn SOL                                    \
| nmrPipe  -fn SP -off 0.50 -end 0.98 -pow 1 -c 0.5   \
| nmrPipe  -fn ZF -size 1942                          \
| nmrPipe  -fn FT                                     \
| nmrPipe  -fn PS -p0 130.6 -p1 2.2 -di               \
| nmrPipe  -fn EXT -x1 9.3ppm -xn 7.2ppm -sw          \
| pipe2xyz  -out ft/hnco_%03d.ft1 -x -ov

# Process Y (indirect) dimension
xyz2pipe -in ft/hnco_%03d.ft1 -y -verb                \
| nmrPipe  -fn EM -lb 0 -c 0.5                        \
| nmrPipe  -fn ZF -size 128                           \
| nmrPipe  -fn FT                                     \
| nmrPipe  -fn PS -p0 8.0 -p1 0.0 -di                 \
| pipe2xyz -out ft/hnco_%03d.ft2 -y -ov

# Process Z (indirect) dimension
xyz2pipe -in ft/hnco_%03d.ft2 -z -verb                \
| nmrPipe  -fn EM -lb 0 -c 0.5                        \
| nmrPipe  -fn ZF -size 128                           \
```

```
| nmrPipe  -fn FT                                    \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                \
| pipe2xyz  -out ft/hnco_%03d.ft3 -z -ov
```

An Example Script for Bruker 3-D Data. Window functions are used in this example, as would be appropriate if the sampling points were uniformly distributed.

```
# Convert data from Bruker format.
bruk2pipe -in raw/ser  \
  -aswap -ndim 2 -nusDim 3 -title hnco -aq2D States   -verb              \
  -bad 0.0 -DMX -decim 1792 -dspfvs 20 -grpdly 67.9841766357422          \
  -xN              2048   -yN               888   -zN                 1  \
  -xT              1024   -yT                64   -zT                64  \
  -xMODE            DQD   -yMODE        Complex   -zMODE           Real  \
  -xSW        11160.714   -ySW         1936.483   -zSW         2000.000  \
  -xOBS         700.023   -yOBS         176.051   -zOBS          70.941  \
  -xCAR           4.771   -yCAR         176.705   -zCAR         119.083  \
  -xLAB              HN   -yLAB               C   -zLAB               N  \
| nmrPipe -fn MAC -macro $NMRTXT/bruk_ranceN.M -noRd -noWr -var nShuf 1  \
  -out fid/hnco_nus.fid -ov

# Expand sparse data
./nusExpand.tcl -in fid/hnco_nus.fid -out fid/hnco_%03d.fid -sample pattern.txt \
  -aqORD 1 -hasW -multW

# Process X (direct) dimension
xyz2pipe -in fid/hnco_%03d.fid -x -verb              \
| nmrPipe  -fn SOL                                   \
| nmrPipe  -fn SP -off 0.50 -end 0.98 -pow 1 -c 0.5  \
| nmrPipe  -fn ZF -size 2048                         \
| nmrPipe  -fn FT                                    \
| nmrPipe  -fn PS -p0 58.0 -p1 67.0 -di              \
| nmrPipe  -fn EXT -x1 9.3ppm -xn 6.0ppm -sw         \
| pipe2xyz  -out ft/hnco_%03d.ft1 -x -ov

# Process Y (indirect) dimension
| nmrPipe  -fn SP -off 0.50 -end 0.5 -pow 1 -c 0.5   \
| nmrPipe  -fn ZF -size 128                          \
| nmrPipe  -fn FT                                    \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                \
| pipe2xyz -out ft/hnco_%03d.ft2 -y -ov

# Process Z (indirect) dimension
| nmrPipe  -fn SP -off 0.50 -end 0.5 -pow 1 -c 0.5   \
| nmrPipe  -fn ZF -size 128                          \
| nmrPipe  -fn FT                                    \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                \
| pipe2xyz  -out ft/hnco_%03d.ft3 -z -ov
```

An Example Script for Varian 4-D Data. This 4-D script is very similar to the preceding 3-D scripts. The total number of sampling points was 3190, and in 4-D we have 8 hypercomplex components per sampling point, for a total of 25520 FIDs (the *yN* parameter). The grid was designed as a 64 × 96 × 64 grid, so we use those values in the *yT/zT/aT* parameters.

yMODE must be complex, while *zMODE/aMODE* must be real. *nDim* must be 2 while *nusDim* is set to the true number of dimensions, 4.

One unusual element in this script is the use of the NMRPipe `ZF` function to truncate the data in the direct dimension. This was carried out to reduce the overall file size, by reducing the digital resolution in the direct dimension.

Finally, note the last section of the script, which transposes the direct dimension into the Z position. This will allow a subsequent call to the *scrub* program to operate more quickly.

```
# Convert data from Varian format
var2pipe -in raw/fid -noaswap -ndim 2 -nusDim 4 -aqORD 0 -verb               \
                                               -title chCH_noe -aq2D States  \
  -xN         1442   -yN        25520   -zN            1   -aN            1  \
  -xT          721   -yT           64   -zT           96   -aT           64  \
  -xMODE   Complex   -yMODE   Complex   -zMODE      Real   -aMODE      Real  \
  -xSW   16025.641   -ySW       12270   -zSW        8800   -aSW       12270  \
  -xOBS   799.9058   -yOBS    201.136   -zOBS   799.9058   -aOBS    201.136  \
  -xCAR      4.797   -yCAR      67.14   -zCAR      4.797   -aCAR      67.14  \
  -xLAB          H   -yLAB          c   -zLAB          h   -aLAB          C  \
  -out fid/chCH_noe_nus.fid -ov

# Expand sparse data
nusExpand.tcl -in fid/chCH_noe_nus.fid -out fid/chCH_noe_%03d_%03d.fid \
  -sample pattern.txt -aqORD 1 -sign -hasW -multW

# Process X (direct) dimension
xyz2pipe -in fid/chCH_noe_%03d_%03d.fid -x -verb      \
| nmrPipe -fn ZF -size 508                            \
| nmrPipe -fn SP -off 0.5 -end 0.98 -pow 1 -c 0.5     \
| nmrPipe -fn FT                                      \
| nmrPipe -fn PS -p0 80.9 -p1 36.0 -di                \
| nmrPipe -fn EXT -x1 5.1ppm -xn 0.1ppm -sw           \
| pipe2xyz  -out ft/chCH_noe_%03d_%03d.ft1 -x -ov

# Process Y (indirect) dimension
xyz2pipe -in ft/chCH_noe_%03d_%03d.ft1 -y -verb       \
| nmrPipe  -fn EM -lb 0 -c 0.5                        \
| nmrPipe  -fn ZF -size 128                           \
| nmrPipe  -fn FT                                     \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                 \
| pipe2xyz  -out ft/chCH_noe_%03d_%03d.ft2 -y -ov

# Process Z (indirect) dimension
xyz2pipe -in ft/chCH_noe_%03d_%03d.ft2 -z -verb       \
| nmrPipe  -fn EM -lb 0 -c 0.5                        \
| nmrPipe  -fn ZF -size 192                           \
| nmrPipe  -fn FT                                     \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                 \
| pipe2xyz -out ft/chCH_noe_%03d_%03d.ft3 -z -ov

# Process A (indirect) dimension
xyz2pipe -in ft/chCH_noe_%03d_%03d.ft3 -a -verb       \
| nmrPipe  -fn EM -lb 0 -c 0.5                        \
| nmrPipe  -fn ZF -size 128                           \
```

```
| nmrPipe  -fn FT                                      \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                  \
| pipe2xyz -out ft/chCH_noe_%03d_%03d.ft4 -a -ov

# Tranpose X and Z for faster SCRUB data access
xyz2pipe -in ft/chCH_noe_%03d_%03d.ft4 -x -verb       \
| nmrPipe  -fn ZTP                                    \
| pipe2xyz -out ft/chCH_noe_ztp_%03d_%03d.ft4 -x -ov
```

Removing Artifacts with the Standalone Program for SCRUB

The Basic *scrub* Command

The standalone program is run from the command line, and it has the following syntax:

`scrub` *pattern_file input_file output_file [options]*

where *pattern_file* is the sampling pattern file, *input_file* is the input file (or set of files, in the case of NMRPipe datasets), and *output_file* is the output file (or set of files, in the case of NMRPipe datasets). The *options* are described below. Note that options can be given with two dashes (e.g. `--threads 3`) or one dash (`-threads 3`); *scrub* understands both. Some options also have short forms (e.g. `-g` for `--gain`). Arguments to options can be separated by a space (`-threads 3`) or an equal sign (`-threads=3`).

The *pattern_file* and *input_file* are required parameters. One must also specify an *output_file*, unless one includes the `--in-place` option to do the calculation in-place.

A synopsis of the command-line options is available by typing `scrup -h`, `scrub -help`, or `scrub --help`.

Sampling Patterns

About Sampling Patterns. A sampling pattern is a text file specifying which sampling points are measured. The most common format for a sampling pattern is for each line of the file to represent one sampling point, and the columns to give the grid coordinates for those points. Thus a pattern file reading:

```
0   4
3   7
10  9
15  3
```

would result in the spectrometer reading FIDs at positions (0, 4), (3, 7), (10, 9), and (15, 3) on the sampling grid. A 3-D experiment has two indirect dimensions, so its sampling pattern would need two columns, while a 4-D experiment has three indirect dimensions and would need three columns. The coordinates are normally integers and would represent multiples of the dwell time in each dimension; for (3, 7), the evolution times would be 3 times the dwell time in the first dimension and 7 times the dwell time in the second dimension. The order of the points in the sampling pattern file determines the order in which they are measured by the spectrometer and the order in which they are stored in the data file.

Some sampling patterns have an additional column giving weighting factors. Incorporating these factors into the FT and SCRUB calculations can improve the accuracy of the final spectrum, and they generally should be used if available. Weighting factors are floating-point values between 0 and 1. A pattern with weighting factors would look like this:

```
0   4   0.99985
```

```
3   7   0.78700
10  9   0.23510
15  3   1.00000
```

Note that a pattern with weighting factors is not the same as a pattern with a weighted distribution of sampling points. The latter refers to how the points are positioned, for example according to an exponential decay curve or a cosine function so that more points are located at shorter evolution times and fewer at larger evolution times, while the former refers to the pattern having small numeric correction factors to help improve the accuracy of calculations. Weighting factors can be included with any kind of pattern, regardless of its distribution of points.

It is a generally accepted convention that comments may be included within sampling pattern files. A comment line should begin with a `#` sign:

```
#   This is a comment.
0   4   0.99985
3   7   0.78700
10  9   0.23510
15  3   1.00000
```

Though almost all sparse experiments today are measured *on-grid*, with the sampled points chosen from among the grid points that would have been measured conventionally, it is also possible to have an *off-grid* experiment, with the samples taken at evolution times that do not correspond to multiples of a dwell time. Because off-grid experiments are so uncommon, there is not an accepted standard for their format. *scrub* can read off-grid patterns where each column contains the actual evolution time, in seconds, used for the corresponding dimension, recorded as a floating-point value.

How *scrub* Reads Sampling Patterns. *scrub* can parse most sampling patterns automatically based on the number of columns in the file and the number formats in those columns. The columns may be separated by any amount of whitespace, and/or with commas. On-grid locations should be given as integers and weighting factors, when present, as floating point values. *scrub* understands both of the common line ending formats used for text files, and should understand any normal text file automatically. Blank lines and comments are ignored. If the coordinates are given as floating-point values, the sampling pattern will be interpreted as an off-grid pattern.

If weights are present, *scrub* will automatically read them in and use them. If you would rather that they not be used, include the `--ignore-weights` option on the command line.

Special Case: Your Pattern Has Extra Columns. If your sampling pattern has extra columns of information in addition to the sampling point coordinates and weighting factors, you may need to specify *which* columns contain the relevant information. *scrub* provides the options `--u-col`, `--v-col`, and `--w-col` to specify the column numbers containing the first, second, and third coordinates for each sampling point, where the first column is numbered 0, the second is 1, etc. The `--weight-col` option specifies the column number for weights, if they are available. If you supply any one of these flags, you must supply all of the others that are applicable to your pattern, as supplying any one of these flags will deactivate the automatic interpretation algorithm. In addition, for off-grid experiments, if you specify the column locations with these flags you will also need to include the `--off-grid` flag.

Spectrum Representation Formats for Sampling Patterns. An on-grid sampling pattern can also be represented using a spectrum file, and *scrub* can read and use such files. A spectrum representation of a sampling pattern consists of a spectrum file with the same number of dimensions as the number of sparse dimensions in the experiment, and the same dimension sizes as for the complete sampling grid. For each position in the experiment that is sampled, the corresponding position in this representation is set to a value of one, while the positions that are not measured are set to a value of zero. This representation is thus something like a binary mask showing which sampling grid points are measured.

To use such a file with *scrub*, simply ensure that it has the correct extension for the NMR spectrum file format in which it is written and include it as the first argument on the *scrub* command-line.

Input Choices

The input data can be supplied in any of the common NMR file formats (NMRPipe, NMRView, Sparky, XEASY) and must already have been processed with the FT into the frequency domain before running *scrub*. *scrub* will identify the file format based on the file's extension. For NMRPipe datasets, use the appropriate `%03d` tag(s). Note that at this time *scrub* does not support 3-D and 4-D NMRPipe datasets stored in a single file (the "stream" format) but rather requires that each plane of the 3-D or 4-D dataset be stored in a separate file.

It generally does not matter which file format you choose for input to *scrub*. *However*, if your processing included apodization or first-point corrections in any of the indirect dimensions, you may find it easier to use the NMRPipe format for input. The NMRPipe header stores apodization information, which is needed for a correct SCRUB calculation. If you supply input data in NMRPipe format, *scrub* will read this automatically from the file header. Otherwise, you will need to supply the apodization information via additional options on the command line.

For very large 4-D datasets, the organization of the data—i.e. the order of the dimensions—can significantly affect the speed of the SCRUB calculation, because of the need to swap data to and from disk if the data are organized unfavorably. To ensure the fastest possible calculation, swap the direct dimension from X into either Z or A prior to running SCRUB. This is especially important when the input data are in NMRPipe format, but it is true to some extent for all input formats. If you swap dimensions, be sure to adjust the dimension order flags provided to *scrub* to ensure that your dimension assignments are correct.

Output Choices

scrub can produce output in the NMRPipe, NMRView, Sparky, and XEASY formats. To indicate the desired file format, simply give your output filename the corresponding extension. For example, if you would like the output in NMRView format, specify a filename ending in `.nv`. The output format need not be the same as the input format. For NMRPipe datasets, use the appropriate `%03d` tag(s). Note that at this time *scrub* does not support 3-D and 4-D NMRPipe datasets stored in a single file (the "stream" format) but rather requires that each plane of the 3-D or 4-D dataset be stored in a separate file.

If there is already a file with your output filename, *scrub* will not overwrite it unless you supply the `--overwrite` or `-w` option.

scrub can also carry out processing in-place. For this, omit the output filename and instead provide the `--in-place` option. The input data will be overwritten with the SCRUB-processed output.

Dimension Assignments

A sampling pattern specifies coordinates for sampling points, but how those coordinates are used—which coordinate is used for which experimental dimension—depends on how the pulse sequence program is written. Furthermore, processing scripts can transpose the dimensions of an experiment to a different order from how they were collected. For these reasons, *scrub* makes it easy to specify how the dimensions in the sampling pattern are to be mapped onto the dimensions of the experiment.

scrub refers to the dimensions of a sampling pattern as U, V, and W, where U is the first column, V is the second, and W (where applicable) is the third. *scrub* refers to the dimensions of the input data as F1, F2, F3, and (where applicable) F4. In a 3-D experiment, F1 is the NMRPipe Z dimension, F2 is Y, and F3 is X. In a 4-D experiment, F1 is the NMRPipe A dimension, F2 is Z, F3 is Y, and F4 is X. *scrub* refers to a conventionally collected dimension, such as the directly observed dimension, as an *index* dimension.

By default, for 3-D experiments *scrub* assumes that the first column of the sampling pattern (U) should be matched to the F1 dimension of the input data, the second column (V) to the F2 dimension, and that the F3 dimension is a conventionally collected directly observed *index* dimension. For 4-D experiments, *scrub* assumes that the first column of the sampling pattern (U) should be matched to the F1 dimension of the input data, the second column (V) to the F2 dimension, the third column (W) to the F3 dimension, and that the F4 dimension is a conventionally collected directly observed *index* dimension.

This mapping can be changed by providing the `--f1` (`-1`), `--f2` (`-2`), `--f3` (`-3`), and `--f4` (`-4`) options on the command line. Each can be set to `u`, `v`, `w`, or `index`. If you provide one, you must provide the rest, up to the number of dimensions in your input data. An example of a complete mapping for 3-D would be:

```
-1 v -2 u -3 index
```

scrub can also be applied in less common cases, for example a 4-D experiment with only two sparse dimensions, a 2-D experiment with one sparse dimension, or an experiment with no directly detected dimension (perhaps having been removed during processing). *scrub* will apply the available sampling pattern dimensions to the experiment's dimensions starting with the first column of the sampling pattern (U) and the first dimension of the experiment (F1). Any extra dimensions beyond the number of sampling pattern dimensions are treated as *index* dimensions. These assignments can be changed using the options described above. For example, a 4-D experiment with sparse sampling in F2 and F3 could be processed with the mapping options `-1 index -2 u -3 v -4 index`. The *scrub* software can handle up to three sparse dimensions and up to four dimensions total.

Apodization and First Point Corrections

If you used a window function and/or first point correction when processing any of the indirect dimensions of your experiment, *scrub* needs to know about this. In general, every dimension requires a first point correction, and apodizations

are required when the sampling pattern is not already weighted according to a window function.

If your input data are in NMRPipe format, *scrub* will read the apodization information directly from the file headers.

For other file formats, use the `--f1-apod`, `--f2-apod`, `--f3-apod`, and `--f4-apod` options to tell *scrub* how your data were apodized. For each apodized indirect dimension, provide, in quotes, the flags from the corresponding NMRPipe apodization command. For example, if you processed the F2 dimension with a sinebell function using the NMRPipe command:

```
nmrPipe -fn SP -off 0.5 -end 0.98 -pow 2 -c 0.5
```

you should use the following *scrub* flag:

```
--f2-apod "-fn SP -off 0.5 -end 0.98 -pow 2 -c 0.5"
```

If you applied a first point correction but not a window function, you can use the `--f1-fpc`, `--f2-fpc`, `--f3-fpc`, and/or `--f4-fpc` flags. Provide the correction value, typically 0.5, as an argument. Thus for an F3 dimension with a normal 0.5 correction:

```
--f3-fpc 0.5
```

Threading and Processor Cores

scrub carries out calculations in parallel whenever possible. *scrub* detects the number of processor cores in your system and will attempt to use all of them. If you would rather that *scrub* use fewer processor cores, provide the `--threads` (`-t`) option on the command line and specify the number of cores to use. For example, `--threads 3` tells *scrub* to use up to three cores.

Information from *scrub*

By default, the *scrub* program is fairly verbose, telling you

- about the sampling pattern, including the number of dimensions and sampling points, and whether or not weighting information is available
- the input data, including its dimensionality, size, and format
- dimension assignments: the mapping of sampling pattern dimensions to input dimensions
- apodization: what *scrub* knows about how the data were apodized

We recommend letting *scrub* print this information and then examining it to be sure that the experiment is configured correctly. However, if you'd rather not see it, the `--quiet` (`-q`) option will suppress most of it.

What *scrub* Does During the Run

During a run, the *scrub* program:

1. Reads and interprets the sampling pattern,
2. Examines the input data, verifying that the dimensionality matches the sampling pattern, and determines the dimension assignments,
3. Creates the output file(s),
4. Calculates the point response (or PSF) and the "pure component," which are needed for the SCRUB algorithm,
5. Applies the SCRUB algorithm to each plane (for 3-D) or cube (for 4-D) of the spectrum, and finally,
6. Computes statistics on the performance of the run, and writes logs and reports when requested (see below).

The *point response* or *point spread function* (PSF) is the expected artifact pattern, which is used in subtractive steps that remove artifacts from the spectrum. It is calculated based on the sampling pattern. These steps remove signal intensity as well as artifact intensity, and the signal intensity must be added back at the end of the SCRUB calculation. The *pure component* is the function used to add back signal intensity, and it is determined from the PSF.

In a 3-D sparse experiment where F3 is the directly observed dimension, each 2-D F1/F2 plane is independent from its neighbors in terms of artifacts, and each 2-D F1/F2 plane is therefore processed independently by SCRUB. Likewise in 4-D where F4 is the directly observed dimension, each F1/F2/F3 cube is an independent space and processed independently by SCRUB. It is for this reason that the *scrub* program refers to the F3 dimension of a 3-D experiment and the F4 dimension of a 4-D experiment as "index" dimensions: they separate/index the independent artifact suppression calculations belonging to an experiment.

During the main part of the calculation, *scrub* displays a progress bar showing the percentage of these independent calculations that have been completed.

When Does SCRUB Stop?

Each independent plane or cube is processed by SCRUB until all identified signals have been processed fully and the residuals of the spectrum appear to be entirely random noise. If there are still signals present at this point, they are not strong enough to be distinguished from random noise with any confidence.

Noise Reports and Estimated Suppression

At the end of each run, *scrub* estimates the percentage of artifacts suppressed and prints this to the screen. For a more detailed look at the extent of artifact suppression, the `--noise-report-csv` (`-r`) option will produce a report in

CSV format showing the estimated noise level at each position in the index dimension before and after SCRUB. These numbers reflect the thermal noise plus the sampling artifacts. To find the thermal noise level, look at planes/cubes with no signals; these should be free of sampling artifacts and their noise levels will reflect the real random noise of the experiment. In a favorable case, where SCRUB is able to eliminate all or almost all artifacts, the noise levels after SCRUB will be the same across all planes/cubes.

Troubleshooting Poor Artifact Suppression

SCRUB generally achieves 98–100% artifact suppression for 4-D experiments and 80–95% suppression for 3-D experiments. The performance is not as good in 3-D as the artifact levels tend to be higher (due to fewer sampling points) at the same time that the spectrum tends to be more crowded. SCRUB errs on the side of caution and does not attempt to process peaks that can not clearly be identified as signals rather than noise. In a very crowded plane in a 3-D spectrum, it may be hard for SCRUB to identify all signals with confidence.

When performance is extremely poor, i.e. artifact suppression under 50%, it is almost certain that the calculation was not set up correctly. The most likely problems are incorrect dimension assignments and incorrect information about apodization and first point corrections.

Troubleshooting Slow Calculations

With 4-D datasets, the calculation speed can depend significantly on the organization of the data. Most significantly, with data in NMRPipe format, having the direct dimension in X can slow down the calculation by multiple orders of magnitude. It is highly recommended that the direct dimension be transposed into the Z or A position for all large 4-D datasets prior to running *scrub*.

Removing Artifacts with SCRUB Using the *pipewash* NMRPipe Plugin

The second way to run SCRUB is through the *pipewash* plugin to NMRPipe. *pipewash* allows you to include the SCRUB calculation step directly in an NMRPipe script, as part of the pipe that funnels data through the various processing functions.

Getting Help on *pipewash* Functions

The *pipewash* program file behaves a lot like the NMRPipe program file: it won't do very much outside the context of an NMRPipe pipe. However, you *can* run it directly on the command line to get help on the available options and the correct syntax for running the SCRUB function. To get this help, type:

```
pipewash –fn SCRUB –help
```

pipewash also supports many of the standard NMRPipe flags available in every call to NMRPipe, including flags for reading and writing data, discarding imaginary data, parallel processing, and header manipulation. For details, type:

```
pipewash –help
```

Calling the *pipewash* SCRUB Function from an NMRPipe Script

The proper way to use *pipewash* is within an NMRPipe processing script. The *pipewash* filename appears at the beginning of the line, followed by the `–fn` flag specifying the processing function to use, in this case `–fn SCRUB` to get the SCRUB processing function. After this, one includes the sampling pattern option `–pattern` and any other options needed. Thus:

```
xyz2pipe –in input –x \
...
| nmrPipe –fn (some function) \
| pipewash –fn SCRUB –pattern pattern_file options \
| nmrPipe –fn (some function) \
...
| pipe2xyz –out output –x
```

Note that the standard way to indicate a flag in NMRPipe is with a single dash, as in `–fn` or `–pattern`. It is probably best to use one dash for all options in NMRPipe scripts even though the command-line parser used by *pipewash* to process its own flags understands other formats.

Dimension Order

When processing data using the SCRUB function in *pipewash*, the sparsely sampled dimensions must begin in the X dimension and continue consecutively. Thus for an experiment with a single sparse dimension, that dimension would need to be X; for an experiment with two sparse dimensions, they would need to be X and Y; for an experiment with three sparse dimensions, they would need to be X, Y, and Z.

Since most NMR experiments are collected with the direct dimension as X, this means that data normally need to be transposed before and after calling the SCRUB function.

3-D Experiments. There are many ways to transpose a 3-D dataset into a valid order for running *pipewash*, depending on the nature of the processing script, but the most simple may be to call the NMRPipe `ZTP` function before and after SCRUB. Thus:

```
xyz2pipe –in input –x \
…
| nmrPipe –fn ZTP \
| pipewash –fn SCRUB –pattern pattern_file options \
| nmrPipe –fn ZTP \
…
| pipe2xyz –out output –x
```

This swaps the direct dimension originally in X for the indirect dimension originally in Z, then restores the dimension order after SCRUB.

In the context of a broader script that will process each dimension in turn and finally carry out SCRUB, one can use the following scheme in 3-D:

```
xyz2pipe –in input –x \
…
Process the original X dimension…
…
| nmrPipe –fn TP \
…
Process the original Y dimension…
…
| nmrPipe –fn TP \
| nmrPipe –fn ZTP \
…
Process the original Z dimension…
…
| pipewash –fn SCRUB –pattern pattern_file options \
| nmrPipe –fn ZTP \
```

```
| pipe2xyz -out output -x
```

This scheme processes the X dimension first. It then swaps X and Y and processes the original Y dimension (temporarily in the X position). Finally, it swaps X and Z so that the original Z dimension is temporarily in X for processing, and so that the two sparse indirect dimensions will be in the X and Y positions for SCRUB. Finally, the original dimension order is restored.

4-D Experiments, without ATP. In four dimensions, we have to use a slightly different approach, since NMRPipe does not at present have an ATP function to match ZTP. First, we process each of the four dimensions:

```
xyz2pipe -in input -x \
...
Process the original X dimension...
...
| pipe2xyz -out proc1 -x

xyz2pipe -in proc1 -y \
...
Process the original Y dimension...
...
| pipe2xyz -out proc2 -y

xyz2pipe -in proc2 -z \
...
Process the original Z dimension...
...
| pipe2xyz -out proc3 -z

xyz2pipe -in proc3 -a \
...
Process the original A dimension...
...
| pipe2xyz -out proc4 -a
```

Next, we transpose the dimensions so that the dimension order is YZAX:

```
xyz2pipe -in proc4 -x \
| pipe2xyz -out transposed.ft4 -a
```

Then, we run SCRUB:

```
xyz2pipe –in transposed.ft4 –x \
| pipewash –fn SCRUB –pattern pattern_file options \
| pipe2xyz –out scrubbed_transposed.ft4 –x
```

And finally, we transpose the data back:

```
xyz2pipe –in scrubbed_tranposed.ft4 –a \
| pipe2xyz –out output –x
```

4-D Experiments, with ATP. Should an ATP function to swap X and A become available in the future, it will be possible to process a 4-D experiment through NMRPipe and the *pipewash* SCRUB function in a single pass, according to the following scheme:

```
xyz2pipe –in input –x \
…
Process the original X dimension…
…
| nmrPipe –fn TP \
…
Process the original Y dimension…
…
| nmrPipe –fn TP \
| nmrPipe –fn ZTP \
…
Process the original Z dimension…
…
| nmrPipe –fn ZTP \
| nmrPipe –fn ATP \
…
Process the original A dimension…
…
| pipewash –fn SCRUB –pattern pattern_file options \
| nmrPipe –fn ATP \
| pipe2xyz –out output –x
```

Please note that at the time of this writing, no ATP function is available in NMRPipe, but it is under consideration to be added in the future. It is recommended that you check the release notes for future versions of NMRPipe in case this function becomes available.

Sampling Patterns and Dimension Assignments

The SCRUB function in *pipewash* parses sampling patterns in exactly the same manner as the standalone *scrub* program, as described above. Dimension assignments work a little bit differently, however. By default, the SCRUB function in *pipewash* applies the U, V, and W dimensions of the sampling pattern to the NMRPipe dimensions in reverse order, as those NMRPipe dimensions are arranged when the data reach SCRUB (i.e. taking into account any transpositions).

Thus in a 3-D experiment with two sparsely sampled indirect dimensions, assuming the data were transposed with ZTP to swap the X and Z dimensions, the default assignments would be:

This column in the sampling pattern	which SCRUB calls	is assigned to current dimension	which was originally
First	U	Y	Y (F2)
Second	V	X	Z (F1)

In a 4-D experiment processed as described above without ATP:

This column in the sampling pattern	which SCRUB calls	is assigned to current dimension	which was originally
First	U	Z	A (F1)
Second	V	Y	Z (F2)
Third	W	X	Y (F3)

Should an ATP function become available, an experiment processed using it would have the following dimension assignments:

This column in the sampling pattern	which SCRUB calls	is assigned to current dimension	which was originally
First	U	Z	Z (F2)
Second	V	Y	Y (F3)
Third	W	X	A (F1)

The default dimension assignments can be overridden using the flags `-x`, `-y`, and `-z`, which specify the dimension assignments (U/V/W) for the (transposed) NMRPipe X, Y, and Z dimensions, respectively.

When ZTP (or ATP, if it becomes available) is used, a change in the dimension assignments may well be required. The SCRUB function prints out the dimension assignments at the start of each run, and it is recommended that these be reviewed carefully.

The Full Script

Here is an example of a complete processing script for a 3-D spectrum using *pipewash*.

```
var2pipe -in raw/fid -noaswap -ndim 2 -nusDim 3 -aqORD 0 -verb          \
                                     -title hnco -aq2D States          \
  -xN              1924  -yN               888   -zN               1    \
  -xT               962  -yT                64   -zT             128    \
  -xMODE        Complex  -yMODE        Complex   -zMODE         Real    \
  -xSW         16025.64  -ySW            2100   -zSW    1949.979980    \
  -xOBS      799.9059.9 -yOBS       201.135971   -zOBS     81.053757    \
  -xCAR            4.66  -yCAR         175.860   -zCAR       117.300    \
  -xLAB               H  -yLAB               C   -zLAB             N    \
| nmrPipe  -fn MAC -macro $NMRTXT/var_rance N.M -noRd -noWr -var nShuf 1 \
-out ./fid/hnco_nus.fid -ov

nusExpand.tcl -in ./fid/hnco_nus.fid -out ./fid/hnco_%03d.fid -sample sampling_pattern.txt -aqORD 1 -sign

xyz2pipe -in ./fid/hnco_%03d.fid -x -verb               \
| nmrPipe  -fn SOL                                     \
| nmrPipe  -fn SP -off 0.50 -end 0.98 -pow 1 -c 0.5     \
| nmrPipe  -fn ZF -size 1924                           \
| nmrPipe  -fn FT                                      \
| nmrPipe  -fn PS -p0 53.0 -p1 0.0 -di                 \
| nmrPipe  -fn EXT -x1 11.0ppm -xn 5.0ppm -sw          \
| nmrPipe  -fn TP                                      \
| nmrPipe  -fn ZF -size 128                           \
| nmrPipe  -fn FT                                      \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                  \
| nmrPipe  -fn TP                                      \
| nmrPipe  -fn ZTP                                     \
| nmrPipe  -fn ZF -size 256                           \
| nmrPipe  -fn FT                                      \
| nmrPipe  -fn PS -p0 0.0 -p1 0.0 -di                  \
| pipewash -fn SCRUB -pattern sampling_pattern_pipe.txt  \
| nmrPipe  -fn ZTP                                     \
| pipe2xyz  -out ft/hnco_%03d.ft3 -x -ov
```

SCRUB Options for Special Cases

Datasets Other Than 3-D and 4-D NMR Experiments

Though the *scrub* program was designed with 3-D and 4-D NMR experiments in mind, it is sufficiently flexible to handle many other cases. *scrub* must match each sampling pattern dimension to an experimental dimension; any remaining dimensions are treated as index dimensions. There may be any number of index dimensions, so long as the total number of dimensions is no more than four. Thus one can run SCRUB on a 2-D dataset with two sparse dimensions and no index dimensions (an extracted plane of indirect dimensions from a 3-D experiment?) or on a 4-D dataset with two sparse dimensions and two index dimensions (one directly observed, the other indirect but collected conventionally rather than sparsely?). One can run SCRUB on a dataset with only one sparse dimension, with or without additional index dimensions. The only hard limitations are: there can be no more than three sparse dimensions and four dimensions total, and every dimension in the sampling pattern must match to an experimental dimension.

Diagnostic Logs

Both the standalone *scrub* program and the SCRUB function in *pipewash* will generate a log upon request, documenting all of the information normally printed to the terminal during a SCRUB calculation as well as detailed information about the processing steps applied during the SCRUB calculation. This log is available using the option `--log`.

The option `--verbose-log` can be added to `--log` to obtain an even-more-detailed log. This provides details about every single step of the SCRUB calculation. Note that these log files can be extremely large and usually provide more detail than is needed unless one is debugging changes in the SCRUB source code.

The `--noise-report-csv` option provides a report of the estimated noise levels at each index dimension position before and after SCRUB, allowing one to assess the degree of artifact suppression. This report is in comma-separated-value (CSV) format.

Calculating Only Specific Parts of the Spectrum

The standalone *scrub* program provides the option to process only selected locations within a larger spectrum rather than the full spectrum, for those rare cases where this may be desirable. Since each position on an index dimension is independent in terms of artifacts from all other index dimension positions, one can choose individual index dimension positions for processing. These are specified using the `--position` flag, followed by an integer giving the desired index dimension position (measured in points, starting at zero). Thus to process only the 12th N/CO plane out of a 3-D HNCO spectrum, one would use:

```
--position 11
```

since the 12th plane is numbered 11 when starting from zero. Positions are numbered in the order they are found in the data file. To calculate more than one position, separate them with whitespace:

```
--position 11 13 28
```

or include the `--position` option multiple times:

```
--position 11 --position 13 --position 28
```

Though most NMR spectra have only a single index dimension, *scrub* supports cases where there are more than one. In those cases, each position is given as a comma-separated list starting from the least-frequently-varying dimension. Thus if F3 and F4 are index dimensions in a 4-D spectrum and we wish to calculate the plane at F3 = 42 points and F4 = 19 points, we would use:

```
--position 42,19
```

In addition to one or more `--position` flags, you must also include a flag to tell *scrub* where to store the results from these calculations. The following options are available:

- The flag `--insert-in-place` instructs *scrub* to store these results in the output file designated elsewhere on the command line.

  If this file does not already exist, it will be created with the size and dimensionality of the full spectrum, but only the requested positions will be filled with data; all other positions will be filled with zero values. For example, if you request to calculate planes 11, 13, and 28 from a 3-D spectrum called `hnco.nv` with 64 points in its direct dimension, using the following command:

  ```
  scrub sampling_pattern.txt hnco.nv hnco_scrub.nv --insert-in-place --position 11 13 28
  ```

  a 3-D spectrum file called `hnco_scrub.nv` with a full 64 planes will be constructed; planes 11, 13, and 28 will be calculated and filled with data; and the other 61 planes will be filled with zeroes.

  If a file is already present with the output filename, *scrub* will first check that the dimensionality is the same as expected and that the dimension sizes are the same as in the input data. If so, *scrub* will replace the data at the requested positions and leave all other positions untouched. For the same example given above:

  ```
  scrub sampling_pattern.txt hnco.nv hnco_scrub.nv --insert-in-place --position 11 13 28
  ```

  but where `hnco_scrub.nv` already exists, *scrub* will first check that the output file `hnco_scrub.nv` is a 3-D spectrum with the same dimension sizes. It will then calculate planes 11, 13, and 28 from the input data `hnco.nv` and overwrite the existing data on those planes in the output spectrum; the other 61 planes will be untouched.

- The flag `--insert-in-place` together with the flag `--in-place` instructs *scrub* to store these results in the original input file, overwriting the input data for the selected planes only. For example, if you request to calculate planes 11, 13, and 28 from a 3-D spectrum called `hnco.nv` with 64 points in its direct dimension, using the following command:

  ```
  scrub sampling_pattern.txt hnco.nv --in-place --insert-in-place --position 11 13 28
  ```

  scrub will calculate planes 11, 13, and 28 from `hnco.nv` and then overwrite the input data on those planes, leaving the other 61 planes untouched.

- The flag `--separate-output` instructs *scrub* to store the result for each position in a separate output file labeled with an underscore and a three-digit position number at the end of its filename. These files will have a lower

dimensionality than the input data, as any index dimensions are excluded. An output filename must be provided; *scrub* will use its extension to determine the output file format, and will insert the position label immediately before the extension.

Thus for the example of calculating planes 11, 13, and 28 from the 3-D spectrum called `hnco.nv` with 64 planes in its direct dimension, the following command:

```
scrub sampling_pattern.txt hnco.nv hnco_scrub.nv --separate-outputs --position 11 13 28
```

instructs *scrub* to build the output files `hnco_scrub_011.nv`, `hnco_scrub_013.nv`, and `hnco_scrub_028.nv`, each with two dimensions since there are two sparsely sampled dimensions in the input data, and each containing the results for its respective plane. The input data are unchanged. *scrub* generates these files using NMRView format since the extension for the output file template is `nv`.

The SCRUB Gain Parameter

The SCRUB calculation consists of a series of subtractions, each one removing a portion of the artifacts. Since each signal in the spectrum generates its own pattern of artifacts, SCRUB must calculate artifacts for each signal. The *gain* parameter determines how much of the artifacts from any one signal SCRUB will attempt to remove at a time. A gain of 100% would direct SCRUB to remove all of the calculated artifacts in one step, while a gain of 1% would instruct SCRUB to remove only 1% of the artifact intensity at any time. (Note that the gain is applied to the amount remaining after preceding subtractions. For a gain of 10%, for example, the first subtraction would reduce the intensity from 100% to 90% of the original intensity, the second would reduce it from 90% to 81% of the original intensity since 10% of 90% is 9%, the third would reduce it from 81% to 71.9%, etc. For any gain setting under 100%, the subtractive process will approach the baseline but never reach it.)

The reason for using a gain of less than 100% is that the artifacts from one signal alter the intensities of all other signals, and at any given time the true intensity of a signal is not known. An attempt to remove 100% of the artifacts based on a faulty, artifact-corrupted measurement of signal intensity would lead to either an overcorrection or an undercorrection, resulting in residual artifacts in the final spectrum. By removing the artifacts in stages, and by alternating between the signals in the spectrum, the interference between the artifact pattern from one signal and the artifact pattern from another signal is gradually removed. In the limit of many iterations, the true intensities of the signals emerge, and a more complete suppression of artifacts is achieved.

If the gain is too low, however, the calculation requires more iterations and is slower. Thus one seeks to achieve a balance between a speedy calculation and maximal artifact suppression.

Based on extensive empirical testing, we have found that the gain may reasonably be set to 50% for 4-D experiments and 10% for 3-D experiments. 3-D experiments benefit from a lower gain as they tend to be more crowded with signals and therefore more subject to interference effects between artifact patterns. These gain values are used by default, and in general there is no need to set the gain explicitly in a SCRUB run.

However, there may be cases where one wishes to achieve either better artifact suppression at the expense of more calculation time, or a faster calculation at the expense of some artifact suppression. In those cases, one can change the gain by supplying the `--gain` or `-g` flag on the command line. Supply the desired gain (in percent) as a parameter. For example:

```
--gain 20
```

would set the gain to 20%.

The SCRUB Base Parameter

When SCRUB subtracts artifacts using a gain of less than 100%, the subtraction process will approach, but never reach, the baseline. The subtractive process could go on forever, but SCRUB instead chooses a stopping point to break off further subtraction. This stopping point is determined using a parameter called *base*. SCRUB continues subtractions until the combined residual artifacts from all of the identified signals are estimated to contribute less than *base* times the noise level to the observed noise. For example, if base is set at its default value of 0.01, then SCRUB continues to subtract artifacts until it estimates that the residual artifacts from all signals are contributing less than 1% of the observed noise.

It might seem that the artifact level in a spectrum could be made arbitrarily low simply by decreasing the *base* parameter; unfortunately, however, there is a limit to how low of a *base* value is helpful. The problem is that SCRUB can only estimate the residual artifact level generated from a signal by reference to its original intensity, and the original intensities of the signals in the spectrum are corrupted by artifacts from other signals as well as truly random thermal noise. Because the true intensities can not be known, SCRUB can only make a rough estimate of the amount of artifacts remaining. Since overcorrection of the artifacts will actually lead to an increase in the artifact level, attempting to reduce the artifacts below a certain point can prove counterproductive. The *base* parameter value of 0.01 has been chosen based on empirical testing to achieve near-maximal artifact suppression, and we recommend using this value in SCRUB calculations.

SCRUB monitors the noise level throughout the subtractive process, and looks for cases of overcorrection, in which case it reverses any subtractions that seem to be making the artifact level worse. Thus it is possible to set *base* to a lower value and then allow SCRUB to push as far as possible, knowing that it will self-correct if its subtraction goes too far. However, we have not found *base* values below 0.01 to be advantageous in our testing.

Calculation of the Pure Component

During the subtractive process, SCRUB subtracts away not only the artifacts, but also the signals, meaning that SCRUB must add back the signals when it is done subtracting. By longstanding convention in the world of CLEAN, a single subtraction operation is said to remove a *component* of the signals in the spectrum; the adding-back process is a process of adding back these components. We use the term *pure component* to refer to the function that is added back for each component of each signal so as to restore the removed signals; it is a *component* that is *pure*, or free of artifacts.

SCRUB obtains the pure component function for a given run from a function called the *point response* or *point-spread function* (PSF), which contains both the signal and its artifact pattern. The challenge lies in separating the signal from the artifacts. Though the signal appears in the center of the PSF, it can be tricky to determine where the signal ends and the artifacts begin.

We are used to thinking of peaks as having circular or elliptical shapes when viewed from above, based on the contours we see in spectrum contour plots, forgetting that the shapes at peak bases can be quite a bit different. The bases of

Lorentzian peaks are not circularly symmetric, and peaks convolved with window functions can have both positive and negative lobes. Including or excluding these features from the pure component can significantly alter the volume of the peak as reported by peak integration tools, and could also change how the peak's lineshape is fitted by programs that attempt to fit analytical functions to spectral data.

Such effects are not *necessarily* important, but in cases where one is comparing integrated peak volumes and/or fitted lineshapes between sparse spectra and conventional spectra, or between two sparse spectra collected with different sampling patterns, the differences may be noticeable.

For these reasons, SCRUB offers several options for how it determines the pure component from the PSF. The default method is called the *contour-ellipsoid* method, and it should work well in most cases. In this approach, SCRUB starts in the center of the PSF, the point of maximal signal intensity, and works outward until the signal intensity reaches a threshold. SCRUB draws a contour around the peak at this threshold. SCRUB then draws an ellipsoid just large enough to contain this contour, and uses the PSF values within this ellipsoid as the pure component. By default, the contour is drawn at 1% of the central peak intensity. If this would fall below the artifact level in the PSF, the contour is instead set just above the artifact level.

An alternative is the *contour-irregular* method, in which SCRUB draws the contour as described in the preceding paragraph, but does not circumscribe an ellipse; instead, in this method SCRUB simply takes the PSF values within the contour. The name for this method arises from the fact that this contour is frequently irregular in shape.

The third and final method is the *fixed-ellipsoid* method, in which SCRUB draws an ellipsoid around the central peak of a fixed, user-defined size. All points within this ellipsoid are used in the pure component.

We would only recommend changing these settings in cases where close examination of the bases of peaks in multiple spectra processed with SCRUB reveal differences and these differences can be shown to impair the ability to compare results between the spectra. In these cases, we recommend experimenting with the settings for each SCRUB-processed spectrum until the bases of the peaks appear consistently between the spectra.

The pure component calculation mode is changed using the `--pure-comp-mode` option, with possible values `contour-ellipsoid`, `contour-irregular`, and `fixed-ellipsoid`. In the *contour-ellipsoid* mode, one can vary the contour threshold using `--pc-contour-ellipsoid-level`, specifying the percentage of the central peak intensity at which to draw the contour (e.g. `--pc-contour-ellipsoid-level 3` would indicate to draw the contour at 3% of the central peak height). One can also increase the size of the circumscribed ellipsoid from the minimum needed to enclose the contour by adding a margin. The `--pc-contour-ellipsoid-margin` option specifies to expand the ellipsoid in all dimensions by a fixed percentage (e.g. `--pc-contour-ellipsoid-margin 30` indicates to calculate the size needed to enclose the contour, then enlarge each semiaxis by 30%).

The *contour-irregular* options allow one to set the threshold, in the same manner as for the contour-ellipsoid mode, and in addition one can choose to add a margin of one additional data point to the contour in all directions (`--pc-contour-irregular-margin 1`)

The *fixed-ellipsoid* mode has one option: the flag `--pc-fixed-ellipsoid-size`, which indicates the size of the ellipsoid as a percentage of the axis length. For example, for a 4-D spectrum with 64 × 256 × 128 points in its indirect, sparsely sampled dimensions, the option `--pc-fixed-ellipsoid-size 3` would specify that each semiaxis of the ellipsoid be 3% of the overall axis length, resulting in semiaxes of 1.9 × 7.7 × 3.8 points in the three dimensions, respectively.

Viewing the PSF and/or Pure Component

The PSF can be examined by including the `--psf` flag followed by a valid spectrum filename (a filename with an extension matching one of the supported file formats). SCRUB's calculated PSF will be written to this file at the end of the run. To view the calculated pure component, include the corresponding `--pure-comp` flag.

Using CLEAN

In addition to SCRUB, the *nmr_wash* package includes an implementation of the original CLEAN algorithm. Our implementation was described in the following publication:

> B.E. Coggins and P. Zhou. "High Resolution 4-D Spectroscopy with Sparse Concentric Shell Sampling and FFT-CLEAN." *J. Biomol. NMR*, **42**, 225–239 (2008)

CLEAN vs. SCRUB

SCRUB is based on CLEAN, and the methods are quite similar. The principal difference between SCRUB and CLEAN is that SCRUB continues the subtraction process until the signals are far below the apparent noise level, almost completely to the baseline, whereas CLEAN stops subtraction when the signals are still above the noise. For this reason, SCRUB achieves substantially better artifact suppression than CLEAN, which leaves behind significant residual artifacts.

Standalone CLEAN

The standalone *clean* program works very much like the standalone *scrub* program. Data should be prepared in the same ways as described above for *scrub*, and the program is run using the same command line syntax. Sampling patterns are interpreted in the same ways, and the same options are available for specifying the input and output, configuring the dimension assignments, supplying apodization information, setting the number of parallel calculations to run at a given time, obtaining diagnostic logs and reports, and controlling the calculation of the pure component.

CLEAN from *pipewash*

The *pipewash* program includes a CLEAN function, which is accessed using:

```
pipewash –fn CLEAN
```

This function works analogously to the SCRUB function in *pipewash*. The CLEAN function should be called from within NMRPipe scripts, after all dimensions have been processed and after appropriate transposition of the dimensions. For the most part, the same options are supported as for the SCRUB function.

CLEAN Parameters: Gain

Like SCRUB, CLEAN has a gain parameter, which has the same purpose and which is used in the way. CLEAN's default gain setting is 10%. The same flag (`––gain` or `–g`) is used to set the gain for CLEAN as for SCRUB.

CLEAN Parameters: Stopping Thresholds

CLEAN under our implementation has two stopping criteria, and in this regard it works differently both from SCRUB and from other CLEAN implementations.

One of the two stopping thresholds is a simple test of how high the signals are above the noise, where CLEAN stops the calculation once the signals have been subtracted down to this level. This is set using `--snr-threshold`, with the threshold specified in multiples of the standard deviation of the noise. Thus `--snr-threshold 5` would indicate to stop when the signals have been subtracted down to five times the standard deviation of the noise. The default value is five.

The second threshold is a check for how much CLEAN is improving the solution, where CLEAN stops when the artifact level is no longer decreasing significantly. The threshold is specified as a percentage, and CLEAN stops when the average noise level has not changed more than this percentage for 25 consecutive iterations. The flag is `--noise-change-threshold`. Thus `--noise-change-threshold 10` indicates that CLEAN should stop if the average noise has not decreased by at least 10% over 25 consecutive iterations. The default is 5%.

CLEAN Parameters: Maximum Number of Iterations

The `--max-iter` flag instructs CLEAN to stop the calculation after a certain number of iterations, if it has not already reached its stopping criteria by then. The default is to allow unlimited iterations.

Using SCRUB or CLEAN from a C++ Program

The SCRUB and CLEAN algorithms used in *scrub*, *clean*, and *pipewash* are implemented in C++ and available for use in other programs through the header and library files included in the *nmr_wash* distribution. The following documentation is not comprehensive, but should serve as an introduction. Please direct additional questions to the program author.

The *nmr_wash* and *nmrdata* Libraries

The *nmr_wash* library supplies a set of C++ classes for working with sampling patterns and for carrying out SCRUB and CLEAN calculations. It uses an additional library, the *nmrdata* library, for accessing NMR spectrum files. Programs wishing to use *nmr_wash* classes will also need to use *nmrdata* classes to make data available to *nmr_wash*. *nmrdata* is included in the *nmr_wash* distribution, and an *nmrdata* tutorial is available by running the Doxygen documentation generator on the *nmrdata* header file.

Header Files and Namespaces

The recommended way to include the *nmr_wash* and *nmrdata* headers is to add the `include` and `libs/nmrdata/include` subdirectories of your *nmr_wash* distribution to your compiler's include path, and then add:

```
#include <nmrdata/nmrdata.h>
#include <nmr_wash/nmr_wash.h>
```

to each source file using these libraries.

nmr_wash classes are within the namespace `nmr_wash`, while *nmrdata* classes are within the namespace `BEC_NMRData`.

Sampling Patterns

The `sampling_pattern` class is used for working with sampling patterns. It has constructors that parse sampling patterns from text files on disk, following the rules described above for the *scrub* program, as well as functions allowing a sampling pattern to be configured programmatically.

Dimension Maps

Dimension assignments are specified using the `dimension_map` struct. The `dimension_map` member variables `F1`, `F2`, `F3`, and `F4` are used to set the dimension assignments for the data to be processed, where each variable should be set to a member of the `dimension_assignment` enum. Apodization information should be supplied by setting the `F1_apod`, `F2_apod`, `F3_apod`, and `F4_apod` member variables to instances of the `dimension_apod` class populated with the apodization information for the corresponding dimensions.

Input Data

Input data should be provided using an `NMRData` class instance from the *nmrdata* library. Note that `NMRData` class instances are *not* copyable, and it is frequently advantageous to store them on the heap with a smart pointer. To obtain an `NMRData` class instance for an existing file on disk, use a call to the static member function `NMRData::GetObjForFile`:

```
NMRData::GetObjForFile( input_filename, open_read_only );
```

where `input_filename` is the path and filename to an NMR spectrum file in one of the supported formats, where the file extension can be used to determine the format, and `open_read_only` is `true` to open the file in read-only mode or `false` to open it in writeable mode. It is recommended to surround calls to this function with error-handling code such as the following:

```
boost::scoped_ptr< NMRData > input_p;   //  A smart pointer
try
{
    //  GetObjForFile returns a pointer to a heap-allocated NMRData instance;
    //  save in the smart pointer
    input_p.reset( NMRData::GetObjForFile( input_filename, open_read_only ) );
}
catch( Exceptions::FileTypeNotRecognized &e )
{
    //  Report error: file type could not be recognized from the extension
}
catch( Exceptions::CantOpenFile &e )
{
    boost::filesystem::path input_file_path( input_filename );
    if( !boost::filesystem::exists( input_file_path ) )
    {
        //  Report error: the input file could not be found
    }
    else
    {
        //  Report error: the input file could not be opened
    }
}
catch( Exceptions::FileError &e )
{
    //  Report generic error
}
NMRData &input = *input_p;  //  Get a reference so that reference syntax
                            //  can be used instead of pointer syntax.
```

Output Data

An `NMRData` instance should also be created to hold the output data from the SCRUB or CLEAN calculation. In general, you create the instance for the correct file type using either the factory function `NMRData::GetObjForFileType` or the constructor for a specific `NMRData` derived class supporting a specific file format (e.g. `NMRData_nmrPipe` for NMRPipe files). One sets the dimensionality using the `SetNumOfDims` member function and configures each dimension using calls to

`SetDimensionSize` and related member functions. One can also copy the dimensionality, dimension sizes, and calibration parameters from another `NMRData` instance using `CopyParameters`. The file is then created using a call to `CreateFile`.

Input and Output Without Files

One can also create `NMRData` instances in memory, without associated disk files. This could be useful in a program with its own routines for accessing spectrum files, where an interface layer could transfer the data from its original source into a temporary, memory-only `NMRData` and return by the same route.

To do this, create an `NMRData` instance using the default constructor, and configure it using the same function calls described under Output Data above, except followed by `BuildInMem` instead of `CreateFile`.

Setting Up a SCRUB Calculation

When the sampling pattern has been opened, the dimension map has been configured, and `NMRData` instances for input and output have been prepared, the calculation itself can be carried out using the `scrubber` class.

Supply the sampling pattern and dimension map in the constructor. The function-call operator is used to invoke the calculation. Several overloads are available:

- To process the entire input spectrum in place:

  ```
  void operator()( BEC_NMRData::NMRData &input )
  ```

  The input data must be opened in writeable mode.

- To process the entire input spectrum and save the output to the supplied output spectrum:

  ```
  void operator()( const BEC_NMRData::NMRData &input, BEC_NMRData::NMRData &output )
  ```

  The input and output spectra must have matching dimensionalities and dimension sizes.

- To process only selected positions in the input and to save the results back to the input spectrum, overwriting the data at those positions:

  ```
  void operator()( BEC_NMRData::NMRData &input, std::vector< std::vector< int > > position_list )
  ```

  The `position_list` is a vector of vectors of ints, the outer vector specifying the positions and each inner vector listing the coordinates (index dimension positions) for a single position. Since most spectra have a single index dimension, the inner vectors normally have only a single element each.

- To process only selected positions in the input and to save the results to the provided output spectrum:

  ```
  void operator()( const BEC_NMRData::NMRData &input, BEC_NMRData::NMRData &output, std::vector< std::
      vector< int > > position_list )‘
  ```

- To process only selected positions in the input and to save the results to separate output spectra, provided as a collection of `NMRData` instances:

  ```
  void operator()( const BEC_NMRData::NMRData &input, BEC_Misc::ptr_vector< BEC_NMRData::NMRData >
      outputs, std::vector< std::vector< int > > position_list )‘
  ```

  The `ptr_vector` class is a smart pointer container available from the `bec_misc.h` file included in the distribution.

Before starting the calculation, one can set the number of threads to use by changing the `num_of_threads` member variable (–1, the default, indicating to set the number of threads to the number of processor cores in the machine), and one can activate the log by setting the `log_file` to a valid `FILE` pointer.

A single `scrubber` can be used for multiple calculations on different input data, as long as the sampling pattern and dimension map are the same.

Monitoring a Calculation in Progress

To monitor the progress of a SCRUB calculation, create an instance of the `status_reporter_scrub` class and pass a pointer to it to the `scrubber` constructor. Each index dimension position to be calculated is dubbed a *job*, and the `get_snapshot` function can be called at any time to get the status of all the jobs for that calculation. Each job is given a sequential ID number when the calculation is started, and the `get_snapshot` function returns a `std::map` of `job_info_scrub` structs indexed by these job IDs. Each `job_info_scrub` instance has a `position` member variable providing the index dimension location for that job, `started` and `done` flags indicating whether the processing of this job has begun and finished, respectively, and information about the current state of the calculation if that calculation is running.

To cancel a calculation in progress, call the `status_reporter_scrub`'s `cancel` member function.

CLEAN

CLEAN calculations are set up in the same manner is SCRUB calculations, except using a `cleaner` class instance instead of a `scrubber`, and a `status_reporter_clean` instead of a `status_reporter_scrub`.

PSFs and Pure Components

The pure component mode and pure component calculation parameters are set using member variables of the `scrubber` or `cleaner` before starting any calculations. The pure component can be obtained by calling `get_pure_component`, which returns a pointer to an array of floats with the pure component values. The PSF can be obtained, in the same format, using `get_psf`.

One can calculate a PSF or pure component for any arbitrary sampling pattern using the `psf_calc` class.

Linking

To link *nmr_wash* into your program, set the linker search path to point to the `libs/boost/stage/lib`, `libs/cminpack-1.1.2`, and `libs/nmrdata` subdirectories of the *nmr_wash* distribution, as well as the directory in the distribution containing `libnmr_wash.a`, and then add the following libraries to your linker command line:

```
-lnmr_wash -lminpack -lboost_date_time-mt -lboost_exception-mt -lboost_filesystem-mt -lboost_program_options-
    mt -lboost_system-mt -lboost_thread-mt -lnmrdata -lm
```

as well as `-lpthreads` if you are linking on Linux.

All of the external libraries included with *nmr_wash* have permissive licenses, and it should be possible to redistribute them in your own packages in binary and/or source form without any restrictions. See their individual licenses for more information.

Acknowledgements

My colleagues Pei Zhou, Jon Werner-Allen, and Tony Yan played seminal roles in the design and testing of the SCRUB algorithm, without which this software package would not have been possible. Their extensive contributions are gratefully acknowledged.

The *nmr_wash* implementation is actually the second version of SCRUB I have produced, and it arose as the product of some very fruitful discussions during a visit to Ad Bax's group at NIH. Many thanks to Ad for inviting me, listening to what we were up to, and putting me in contact with all the right people in his group. Jinfa Ying, in particular, provided great advice on the workflow and user interface, and Frank Delaglio has been a huge help in building this new version to interface with NMRPipe. I most grateful for all of their advice and assistance.

Like every other programmer, I am dependent on beta testers to help find bugs, and I was especially fortunate in my beta testers for this package. Many thanks are due to Flemming Hansen of University College London and to Patrick Reardon of the Pacific Northwest National Laboratory for identifying some significant issues.

Finally, it will be apparent to anyone who examines this package that it relies heavily on the outstanding libraries produced by other scientists as well as the broader C++ community, and I would like to thank them for their generosity in sharing so much valuable work with the public. Many thanks to the developers of the Boost C++ libraries, MINPACK, the C MINPACK port, Blitz++, and the PHENIX Computational Crystallography Toolbox.

BRIAN E. COGGINS

Durham, North Carolina
November 2013

Legal Notices

Copyright

License Info

The *nmr_wash* suite is licensed under a relatively permissive license allowing for redistribution of the program and derivative works in both source and object form. We do require that you cite our work when publishing or presenting work based on SCRUB or derived using SCRUB. For full details, see the `LICENSE.txt` file.

Legal Notices for External Libraries

- NMRPipe source code files used in *pipewash* are included with permission from Frank Delaglio.
- The Blitz++ library is used under the terms of the Blitz++ BSD License, included in the file `COPYRIGHT` in the Blitz++ subdirectory of the distribution.
- Under the terms of the CMINPACK license, the following notice is provided: "This product includes software developed by the University of Chicago, as Operator of Argonne National Laboratory."
- Copyright notices and license files for all of the included libraries may be found in their respective subdirectories of the software distribution.