Cambridge CS

Compressed sensing reconstruction for multidimensional NMR spectroscopy

User manual

Mark J. Bostock, Robert Tovey, Wayne Boucher and Daniel Nietlispach
Department of Biochemistry, University of Cambridge, UK

May 2017

Contents

1 Overview 2

2 Installation 3

3 Quick Start 4

4 Using Cambridge CS 6

4.1 Concept 6

4.2 Loading data 6

4.2.1 Data Parameters 6

4.2.1.1 Data size 8

4.2.1.2 RQD-params 9

4.2.2 NMRPipe 9

4.3 Pre-processing (direct dimension processing) 12

4.4 Indirect dimension reconstruction 15

4.5 Post-processing 17

5 Spectrum viewer 24

Appendices 28

A Lookup table for Bruker Avance data 29

B Test Data 30

B.0.1 2D, ^{15}N HSQC, ubiquitin 30

B.0.1.1 NMRPipe 31

B.0.2 3D ^{13}C NOESY, sensory rhodopsin II (pSRII) 33

B.0.2.1 NMRPipe 34

B.0.3 3D SOFAST (Best) [^{15}N,^{1}H] TROSY HNCO 35

B.0.3.1 NMRPipe 36

B.0.4 4D, HCCH NOESY, pSRII 36

C Citation and License 38

References 39

Chapter 1: Overview

Methods for improving the time-efficiency of NMR data acquisition have been popular since the early days of multidimensional NMR [1]. Such methods have focussed around recording a subset of the full data matrix (non-uniform sampling, NUS) and coupled with a variety of processing methods due to the restriction of the Fourier transform to using regularly sampled data [1–7]. NUS approaches are typically used to improve resolution, sensitivity or reduce the recording time for spectra, or a combination of all three [4, 8]. Compressed sensing (CS) was pioneered in the field of information theory [9–11] and quickly found applications in a number of fields including MRI [12] and more recently NMR as a new method for processing NUS data [13–16]. CS has been demonstrated to be a powerful method to reconstruct undersampled spectra with high fidelity, even for weak peaks in spectra with a high dynamic range [16]. Consequently CS has rapidly gained popularity in NMR spectroscopy. The aim of this software is to provide a simple means of processing NUS data with a variety of available CS algorithms.

Chapter 2: Installation

The software comes as a single *.tar.gz* package. In order to simplify installation the package includes miniconda3 containing python3 and relevant libraries. This will hopefully avoid problems with different python/python package versions. Miniconda is subject to the Anaconda End User Licence Agreement see (`https://docs.continuum.io/anaconda/eula`). The package also contains Azara (see `http://www2.ccpn.ac.uk/azara/`) which is used to provide postprocessing commands.

The current version has been tested on 64 bit Linux distributions. A separate download is available for 64 bit Mac OS X 10.11 and later. To install:

1. Uncompress and extract the download.

   ```
   tar zxvf cambridgecs.tar.gz
   ```

2. The uncompressed folder *CambridgeCS* contains the following folders: *azara-2.8*, *bin*, *binBuild*, *miniconda3*, *testData*. Change into the *binBuild* directory and run the command:

   ```
   bash buildCython
   ```

   This will produce *.so* files for all the *.c* files in *binBuild*.

3. The code is called using the scripts in the *bin* directory. These should be added to the system path e.g. to your *.bashrc* file. `cambridgecs` calls the reconstruction code, whilst `graphPlot` calls the simple spectrum viewer independently.

Chapter 3: Quick Start

This section is designed to give a very brief overview of running the code in simple cases, with a particular view to running the example scripts contained in the folder *testData* and to adapting these to other similar situations. Detailed information on running 'Cambridge CS' and the various commands therein is given in chapter **??**.

CambridgeCS can be called using the following command:

```
cambridgecs [OPTION] [Script file]
```

```
-a --aut
Run in automatic mode. Does not launch GUI and requirements for user input are
suppressed. Requires a script file.
-h --help
Show help text
-g --gui
Launch the GUI. The GUI maybe launched without a script file in which case a script
can be built within the GUI. Alternatively an existing script file maybe loaded
into the GUI and amended or run.
```

To load one of the test scripts (e.g. *550_cs.scr*), change to the relevant directory and run the command:

```
cambridgecs -g 550_cs.scr
```

The example chosen is for a 3D H-detected HNCO experiment. The GUI will load the commands contained in the script file. Basic information about the dataset is shown in the 'Data parameters' section. Below this is the 'Pre-processing script' which carries out processing of the direct dimension. Additional commands can be added using the green crosses. Red crosses remove commands. The data reconstruction is carried out using the 'Reconstruction script' section. Here the choice of reconstruction algorithm can be made along with phasing and weighting functions for the indirect dimensions. Finally post processing is specified in the 'Post-processing script'. Typically this involves additional commands e.g. reversing dimensions or adding additional baseline corrections. As with the direct dimension, processing commands can be added and removed using the green and red crosses. Since post-processing uses Azara (packaged with the code), the dimensions are specified using script_com_n for the nth dimension.

In the 'Data Parameters' and 'Reconstruction Script' sections of the code the dropdown 'Extra Params' menus contain additional, but non-essential commands which may be used or needed in certain situations.

The script can be viewed with the button 'Open script editor'. This loads the 'Script editor' window where the script can also be edited and synced with the GUI. It is hoped that this will help users to become familiar with the scripting style. The script can be saved with the button 'Save file' although the script will also be automatically saved once the code is run.

To run the code, press the button 'Process current script'. A terminal window will appear to show the progress of the reconstruction. At the end of the reconstruction the simple viewer will open automatically (assuming post processing commands are set). This allows quick visual inspection of the reconstruction. Changes can be made in the GUI or the 'Script editor' and the code re-run as necessary. Each run opens a new simple viewer window. GUI windows can be closed with the keys 'q' or 'Esc'.

The *testData* folder contains other examples of 2, 3 and 4D data sets all of which can be reconstructed using CambridgeCS (see section B for details).

Whilst the main use of Cambridge CS is for CS processing of NUS data, it is also possible to process fully-sampled data. This can be done in two ways - either by treating the data as if it is an NUS experiment by providing a suitable points list (nuslist). This is essential if the data is recorded in NUS order i.e. quadrature detection is performed before dimension incrementation. If the experiment is recorded as a standard fully-sampled experiment the 'Uniform sampling flag' can be set in 'Data Parameters'. It is then necessary to specify 'max_points' for each indirect dimension under the 'Reconstruction script' so that the code knows how to correctly reshape the data. An example of this is given in the script `102_ft_full.scr`.

Chapter 4: Using Cambridge CS

4.1 Concept

Data processing may be carried out using either a script-based approach, or using the GUI. We recommend that initially the GUI is used in order to become familiar with the scripting approach. The button 'Open Raw Script' opens a text window which will be populated with commands as these are selected in the GUI. Commands may also be typed in the scripting window and synced with the GUI via the 'Sync text' button. The scripts and GUI have four sections:

Data parameters
Pre-processing script
Reconstruction script
Post-processing script

'Data parameters' contains information about the data i.e. file size, location, data type etc. Data parameters must be set for all processing scripts. More details are given in the section 4.2.

'Pre-processing' parameters contains the commands needed for processing the direct (acquisition) dimension of multidimensional experiments. It is assumed that this dimension is always fully-sampled. More details on the pre-processing functions are given in section 4.3. This section is optional if pre-processing is carried out in another software package. Explicit support is available for data pre-processed via the NMRPipe suite (section 4.2.2).

Reconstruction parameters contains the options for NUS data reconstruction of the indirect dimensions either via the FFT, or using a variety of compressed sensing algorithms. The output from this may either be reconstructed time-domain data, or reconstructed frequency-domain data.

'Post-processing'; this section is optional. If post-processing commands are set in the script, processing will be carried out using the Azara software package (W. Boucher, unpublished). Both time-domain or frequency-domain output from the reconstruction may be processed. Alternatively other processing packages may be used. Further details on Azara post-processing are given in section 4.5. Note that the post-processing section must be set for the simple viewer to open automatically.

4.2 Loading data

4.2.1 Data Parameters

Various options and flags are available to specify data import parameters. These are detailed below. Essential options are visible in the GUI. Extra options may be selected from the dropdown menus.

Directory
directory </Path/To/Directory>
Specify the file path to the directory containing the input data. Output data will be written to this directory. See also **Raw data directory** below

Data Input
dataInput </Path/To/Data>
Specify the file path to the input data if this is not the raw ser or fid file (Bruker and Varian respectively).

Raw data directory
rawDirectory <Directory>
Raw data may be stored in a subdirectory within Directory. If Raw data directory is specified, the software will look for raw data in </Path/To/Directory/rawDirectory>. Output data will still be saved in </Path/To/Directory>.

NUS list
nusList <name>
Specify the name of the list file containing the acquired points in the indirect dimension(s). This may be specified in terms of increments (i.e. starting at 0) or in terms of points (i.e. starting at 1). The file format may be a column vector, where a single point/increment is specified by ndim lines where ndim is the number of dimensions, or as a matrix of the size [npts_indirect, ndim] where npts_indirect is the total number of points in the indirect dimensions. The software detects the format automatically. The NUS list is assumed to be in the experiment directory (either **Directory**, or **Raw data directory**). The default is *vclist* (Bruker convention)

Original NUS list
nusListOriginal <name>
Used in combination with <nusList> to artificially further undersample an experiment. <nusListOriginal> specifies the nusList which describes the recorded data. <nusList> is then used to specify a second list which which specifies a new undersampled schedule. For example, if a fully-sampled experiment is recorded and the user wishes to test the effect of 10% sampling, <nusList> would be set to a 10% sampling list while <nusListOriginal> would be a fully-sampled list. The data is then read-in using <nusListOriginal> and masked according to <nusList>. The output of the reconstruction will be equivalent to the 10% sampled spectrum. If the original experiment is already undersampled, care must be taken to avoid specifying points that are not present in <nusListOriginal>.

Data type
int, float
Used to specify the data type of raw data: either 32-bit integer or 32-bit float. Note that this option is only available for raw (unprocessed data) which is being processed using the pre-processing commands. If data is pre-processed in other software, it is assumed the data-type is 32-bit float.

Endian
big_endian, little_endian

Used to specify the endianess of raw data. Note this option is only available for raw (unprocessed data) which is being processed using the pre-processing commands. If data is pre-processed in other software, it is assumed that the data-type is 32-bit float.

Data input flag
nmrPipe, varian
Used to specify alternative data input formats (NMRPipe or Varian).

Shuffle
shuffle21, shuffle32, shuffle5724
Shuffles the input data ordering by switching the locations of real (r) and imaginary (i) points in the indirect dimensions as shown below:

Dimension	Default order		Shuffled order	
2D	r	1	i	2
	i	2	r	1
3D	rr	1	rr	1
	ir	2	ri	3
	ri	3	ir	2
	ii	4	ii	4
4D	rrr	1	rrr	1
	irr	2	rri	5
	rir	3	rir	3
	iir	4	rii	7
	rri	5	irr	2
	iri	6	iri	6
	rii	7	iir	4
	iii	8	iii	8

Threads
threads <number>
Specify the number of cpu threads used to reconstruct the data. The default is 1. If set to 0, the maximum number of available cores will be used.

4.2.1.1 Data size

All data is imported as a pseudo-2D.

Direct-dim
npts npts <points>
The total number of points in the direct dimension (real and imaginary).

block block <block size>
The block size for the direct dimension. This option can be found in the drop-down menu 'Extra Params'.

Indirect-dims
npts npts <points>
The total number of points in all the indirect dimension (real and imaginary) i.e. product of the indirect dimension sizes.

ndim ndim <number>

The total number of indirect dimensions, i.e. $n-1$ for an n-dimensional experiment.

block block <block size>

The block size for the indirect dimensions. This option can be found in the drop-down menu 'Extra Params'.

interlace interlace <dimension>

Used to convert P and N type data e.g. from Echo-Antiecho/Rance-Kay data acquisition to cosine/sine type data needed for processing to generate an absoprtion mode lineshape. Note interlacing is only used if a pre-processing script is included. Dimension assumes that $n = 1$ is the direct (acquisition dimension).

acquired_points acquired_points <number>

Used in cases where the acquisition was not completed to specify the number of data points (real and imaginary) which were recorded in the indirect dimensions as opposed to the total size of the indirect dimensions which should still be specified.

4.2.1.2 RQD-params

More information forthcoming

4.2.2 NMRPipe

Cambridge CS may be be used in conjunction with NMRPipe processing. A typical scenario would be to use NMRPipe for pre- and post-processing with CS reconstruction carried out in the indirect dimensions using the algorithms available in Cambridge CS. In this case the CS processing script should not contain the sections

```
# Pre-processing parameters:

# Azara post-processing parameters
```

Cambridge CS is able to extract parameter information from NMRPipe binary files, and write out to NMRPipe format for further processing/viewing in the NMRPipe software suite. One significant difference between NMRPipe and Cambridge CS is that Cambridge CS uses the Azara convention of specifying dimensions in numerical order and processing in situ, rather than the NMRPipe convention of transposing the required dimension to the direct dimension for processing. Thus a typical NMRPipe pre-processing script should not include the final transpose statement.

```
|nmrPipe -fn TP \
```

A sample NMRPipe pre-processing script for a 2D TROSY is shown below:

Conversion from Bruker format to pipe format:

<fid.com>

```
#!/bin/csh -f
#
bruk2pipe -in ./ser -bad 0.0 -DMX -noswap  \
         -decim 2000 -dspfvs 20 -grpdly 67.9862518310547 \
-xN      2048       -yN     300           \
-xT      1024       -yT     250           \
-xMODE   Complex    -yMODE  Echo-AntiEcho \
-xSW     10000      -ySW    1700.680      \
-xOBS    800.130    -yOBS   81.076        \
-xCAR    4.708      -yCAR   115.100       \
-xLAB    1H         -yLAB   15N           \
-ndim    2          -aq2D   States        \
-out data.fid -ov -verb
```

Processing of the direction dimension, keeping the data in the form xy:

<ft1.com>

```
nmrPipe -in data.fid                              \
| nmrPipe -fn POLY -time                          \
| nmrPipe -fn SP -off 0.5 -end 0.98 -pow 2 -c 1.0 \
| nmrPipe -fn ZF -size 4096                       \
| nmrPipe -fn FT                                  \
| nmrPipe -fn PS -p0 -173 -p1 180 -di             \
| nmrPipe -fn POLY -auto -xn 5.0ppm -ord 1        \
| nmrPipe -fn EXT -xn 5.7ppm -x1 10.95 -sw -verb  \
-verb -ov -out test.nus
```

Subsequent CS processing requires specification of a header flag <input_flag> with the value 'nmrPipe', as well as the size of the NMRPipe header <head> (the default value is 512 which should work). Other parameters are specified as normal. An example script is shown below:

```
# General properties:

    directory         /home/user/path/to/pipe/input/file
    dataInput         /home/user/path/to/pipe/input/file/test.nus
rawDirectory Bruker_directory_number
    outputFile        pipe_output
    experimentNumber  1
    nmrPipe
    head              512

    direct-dim:
        npts                1714
end_script

    indirect-dims:
        npts                300
        ndim                1
    end_script

# Reconstruction parameters:
    reconstruction_script:
        reconstruction_flag IHT
        stop_flag           l2

        dim 2:
            zerofill        2
            phase           -90.0    -360.0
            sinebell        60.0
        end_script
    end_script
```

After processing in the indirect dimensions, a small NMRPipe script <$outputFile.com> is written automatically to convert the data back to a format which can be read by NMRPipe, using the NMRPipe programme bin2pipe as shown below.

```
#!/bin/csh
#
bin2pipe \
   -in /home/user/path/to/pipe/input/file/pipe_output_recon.spc \
   -bad 0.0 -noswap \
-xN 1714 \
-xT 1714 \
-xFT Freq \
-xMODE Real \
-xSW 4184.570312 \
-xORIG 4560.469238 \
-xOBS 800.130005 \
-xLAB 1H \
-yN 2048 \
-yT 2048 \
-yFT Freq \
-yMODE Real \
-ySW 1700.680054 \
-yORIG 8492.844727 \
-yOBS 81.075996 \
-yLAB 1H \
-ndim 2 \
|nmrPipe -fn TP -auto \
|nmrPipe -fn REV \
|nmrPipe -fn TP -auto \
-out /home/user/path/to/pipe/input/file/pipe_output.pipe.ft2
     -ov -verb
```

The various parameters shown in this script are extracted from the header file of the input NMRPipe file automatically. Further examples of using CambridgeCS in combination with NMRPipe are given in Appendix B

4.3 Pre-processing (direct dimension processing)

Pre-processing of NUS data involves processing the directly acquired dimension which is fully-sampled. This may be done using the software of choice of the user e.g. Bruker Topspin or NMRPipe. Loading such data is described in section 4.2. Within the CS-NMR software, we provide direct-dimension processing modelled on the Azara software but coded in Python (W. Boucher, unpublished). Common processing functions, described below, are available and operate in the same way as described in the Azara help pages (`<http://www2.ccpn.ac.uk/azara/azara_docs/process.html>`), which may be useful for further information. The direct dimension processing commands should be contained between the flag-lines `pre_processing_script:` and `end_script`. An example pre-processing script is provided below:

```
pre_processing_script:
    complex
    brukerGroupPhase
    sinebell        60
    gaussian_sw     16.4   1.1   10000
    zerofill        2
    fft
    phase           7     -72
    real
    range           1400   2048
end_script
```

The output from the pre-processing script is used directly for further processing, but the data processed in dimension 1 is also written out directly as **_prepro_f1.spc* along with **_prepro_f1.spc.par*, a parameter file containing information on the dimension and block sizes in the Azara format, which may be of use for further processing in other software packages or to analyse the direct dimension further e.g. using the Azara plot2 software. Only information on the number of points and block sizes are correct in the parameter file. Values for spectral width (sw), spectral frequency (sf), reference point (refpt), reference ppm (refppm) and nucleus (nuc) are arbitrary and would need to be edited by the user if correct referencing is needed. Most likely these files will not be used by the average user.

Processing commands available are listed below with a brief describtion of their usage and syntax, taken from the Azara manuel. For more information, see the Azara help section `<http://www2.ccpn.ac.uk/azara/azara_docs/process.html>`. In the GUI, commands may be selected by clicking on the green cross and selecting the relevant command. Any necessary parameters should be typed in the text boxes. Commands may be removed by clikcing the red cross

complex
complex
Specify data is complex.

real
real
Specify data is real.

conjugate
conjugate
Take the complex conjugate of data. Data must be complex, i.e. the complex flag (above) must previously have been set.

brukerGroupPhase
brukerGroupPhase <GRPDLY>
This does the first order phase correction required for Bruker AV data where <GRPDLY> is given in the Bruker *acqus* file. The phase correction is 360×<GRPDLY>. For Bruker Avance data the parameters DSPFVS and DECIM, given in the Bruker *acqus* file can be converted to a lookup value using the lookup table (see **??** and converted to <GRPDLY> using the formula groupDelay=0.5×lookup/DECIM).

If no argument is specified GRPDLY will be read directly from the *acqus* file assuming this is in the experiment directory.

The brukerGroupPhase command applies the following functions:

```
fftn
phase 0 <360.0*GRPDLY>
ifftn
```

where `fftn` and `ifftn` are the normalised Fourier and inverse Fourier transforms respectively, phase is as specified below, and upper cuts the upper point of the data to the initial number of points (npts1) minus the group delay value, with int($GRPDLY) retaining the integer part of group delay.

conv_sine
conv_sine <half width>
This convolves the data with a sine function with the given <half width>, and subtracts the result from the data. Typically this is done on the acquisition fid before anything else is done, in order to remove a water signal that occurs at zero frequency. A typical value with 1024 data points would be `conv_sine 8`.

decay_sw
decay_sw <line broadening> <spectral width>
This multiplies the data by a decaying exponential exp($-\pi LBt$) where LB is the line-broadening (Hz).

sinebell
sinebell <angle>
This multiplies the data by a sine function with the given <angle> (specified in degrees) at point 1, and, if there are n (complex/real) points, with angle 180 degrees at point n+1 (*not* point n).

sinebell2
sinebell2 <angle>
This multiplies the data by a sine function squared with the given <angle> (specified in degrees) at point 1, and, if there are n (complex/real) points, with angle 180 degrees at point n+1 (*not* point n).

gaussian_sw
gaussian_sw <line broadening (Hz)> <sharpening factor> <spectral width (Hz)>

This multiplies the data by the gaussian $\exp(a + bt + ct^2)$. If LB is the entered line broadening (in Hz) and s is the sharpening factor, then $a = -\ln 2/s^2$, $b = \pi \times LB$ and $c = -(\pi \times LB \times s)^2/4\ln 2$. Note that LB must be **positive** for normal use, which is the **opposite** of the Bruker convention. The function converts a Lorentzian line (or multiplet) of width FWWH equal to LB to a Gaussian line of width FWWH equal to $s \times LB$. The maximum value of the multiplying function is 1, which is obtained at the fraction of the acquired time $t/T = 2\ln 2/(LB \times \pi \times T \times s^2)$. The spectral width (SW) is needed by the program in order to translate between point number, x, and the time value, t, used in the function, by $t = x/(2 \times SW)$. If a value of $SW = 0.0$ is given, the program uses the value for SW which was entered in the par file (or the default). The suggested range of values for s are from 1.3 to 0.7 (possibly 0.5).

zerofill
zerofill <n>

This zerofills the data so that its final size is 2^n times the original size. For data that is not already of size 2^n, the data is first zero padded to the nearest 2^n before further zero filling.

fft
fft

This does a complex Fourier transform.

phase
phase <phase0> <phase1> <pivot, default=1>

This phases (complex) data with specified parameters <phase0> and <phase1> (both specified in degrees) and <pivot> which assumed by default to be the first point, point 1. The data must be complex.

reduce
reduce

This converts complex data to real data by discarding the imaginary part.

upper
upper <upper bound>

This truncates the data so that the last point is now at <upper bound>. The first point is point 1.

lower
lower <lower bound>

This shifts data so that the origin is now at <lower bound>. The first point is point 1.

range
range <lower bound> <upper bound>

This shifts and truncates data so that the origin is now at <lower bound> and so that the last point is <upper bound>. The first point is point 1.

4.4 Indirect dimension reconstruction

'Reconstruction script' contains the commands for processing the indirect dimensions of NUS acquired data. 'Reconstruction type' provides options for using various different algorithms as well as the FFT for processing the indirect dimensions. 'Extra Params' relevant to the different algorithms are accessed using the dropdown menu. Parameters specific to the different indirect dimensions should be set. The number of available dimensions is governed by the 'ndim' parameter in the 'Data parameters' section.

Reconstruction type
General options

resume
Resumes reconstruction from saved data. The code will attempt to load previous files using default names; if the expected files are not found, the reconstruction will restart afresh. Can be used, for example if max_iter was not set high-enough and further iterations are needed to improve the reconstruction quality.

saveCompleteData
Saves reconstructed time domain data for subsequent use in post-processing software. Use with the post-processing flag 'Input Data useTimeData' (see section 4.5).

record_l2

Write out values of the l2_norm for the reconstruction. The output file is **_l2.out.*

record_noise

Write out values of the noise for the reconstruction. The output file is **_noise.out.*

Algorithm specific options
The options below are used by some or all of the algorithms. Details are given here. The options available for each algorithm are given in the algorithms section.

max_iter
noise
l2_stop
threshold
global_threshold
k

stop_flag

FT
No additional options

IHT
max_iter <default=1000>
noise <noise-level, default=0>
l2_stop <default=0>
threshold <default=0.75>
global_threshold <default=0.01>
stop_flag <None/l2/noise>

IST

max_iter <default=1000>
noise <noise-level, default=0>
l2_stop <default=0>
threshold <default=0.75>
global_threshold <default=0.01>
stop_flag <None/l2/noise>

IRLS1

max_iter <1000>
noise <noise-level>
l2_stop <default=0>
k <default=0.015>
stop_flag <None/l2/noise>

Dimension commands

scaleT0

scaleT0 <scaling factor>

Scales the first point of the relevant indirect dimension by <scaling factor>. Used to correct the first point of certain experiments. Assumes the data is complex.

conjugate

conjugate

Take the complex conjugate of data. Data is assumed to be complex.

exchange

exchange

Exchange the real and imaginary parts of complex data.

mask

mask_ppmm

Apply ppmm mask (++−) to the relevant indirect dimension i.e. the sign of points 3 and 4 and every third and fourth point equivalently is changed. Needed for some Bruker data.

mask_pm

Apply pm mask (+-) i.e. the sign of every other point is changed. Equivalent to the conjugate command.

extend

extend <no. of complex points>

Extend data by specified number of complex points.

decay

decay <end value>

Multiplies the data by a decaying exponential which is 1 at point 1 and <end value> at point n.

sinebell

sinebell <angle>

This multiplies the data by a sine function with the given <angle> (specified in degrees) at point 1, and, for n complex points, with angle 180 degrees at point n+1 (*not* point n).

sinebell2
sinebell2 <angle>

This multiplies the data by a sine function squared with the given <angle> (specified in degrees) at point 1, and, if there are n (complex/real) points, with angle 180 degrees at point n+1 (*not* point n).

gaussian_sw
gaussian_sw <line broadening (Hz)> <sharpening factor> <spectral width (Hz)>

This multiplies the data by the gaussian $\exp(a + bt + ct^2)$. If LB is the entered line broadening (in Hz) and s is the sharpening factor, then $a = -\ln 2/s^2$, $b = \pi \times LB$ and $c = -(\pi \times LB \times s)^2/4\ln 2$. Note that LB must be **positive** for normal use, which is the **opposite** of the Bruker convention. The function converts a Lorentzian line (or multiplet) of width FWWH equal to LB to a Gaussian line of width FWWH equal to $s \times LB$. The maximum value of the multiplying function is 1, which is obtained at the fraction of the acquired time $t/T = 2\ln 2/(LB \times \pi \times T \times s^2)$. The spectral width (SW) is needed by the program in order to translate between point number, x, and the time value, t, used in the function, by $t = x/(2 \times SW)$. If a value of $SW = 0.0$ is given, the program uses the value for SW which was entered in the par file (or the default). The suggested range of values for s are from 1.3 to 0.7 (possibly 0.5).

4.5 Post-processing

Post-processing is available using the Azara processing package. Data input may be time-domain data or frequency domain data from the output of the reconstruction script. This is specified via the flag 'Input Data', using the options *useTimeData* and *useFTData*, respectively. A selection of commonly-used Azara commands can be specified from within the CS-processing script between the flag-lines `post-processing script:` and `end_script`. Within this, dimensions can be specified using the Azara nomenclature `script_com <dimension>`, `end_script`. An example post-processing script is given below:

```
# Azara post-processing script:
    post_processing_script:

        useFTData

        script_com 2
                base_poly 6 0
        end_script

        script_com 3
                base_poly 4 0
        end_script

        script_com 1
                base_poly2 8 1 1 400
        end_script
    end_script
```

The CS-script uses the commands listed under `post_processing_script:` to write an Azara script file **_azara.scr*, which is then run using the Azara programme (W. Boucher, unpublished; `http://www2.ccpn.ac.uk/azara/`). This has the advantage that the commands can be re-run or modified at any time by changing the Azara script and calling with `process *_azara.scr`. The following functions are available from CS-processing using either the GUI or the script-based approach, grouped by category as found in the GUI. A brief description of their usage and syntax is given below, taken from the Azara manual. For more information, see the Azara help section `<http://www2.ccpn.ac.uk/azara/azara_docs/process.html>`.

Real/complex

complex
complex
Specify data is complex.

real
real
Specify data is real.

reduce
reduce
Reduce complex data to real data by throwing away the imaginary part.

Data manipulation

exchange
exchange
Swap the real and imaginary parts of complex data.

conjugate
conjugate
Take the complex conjugate of complex data.

shift
shift <shift amount>
This shifts the data by the positive amount <shift amount>.

reverse
reverse
This reverses the data.

set
set <first point> <last point> <value>
Sets each point between <first point> and <last point> (inclusive) to <value>. Applies only to real data. For complex data use **set2**.

set2
set2 <first point> <last point> <real value> <imaginary value>
Sets each point between <first point> and <last point> (inclusive) to <real value> +i*<imaginary value>. Applies only to complex data. For real data use **set**.

scale
scale <first point> <last point> <value>
Multiply data by <value> from <first point> to <last point> (inclusive). The data must be real. For complex data, use scale2, below.

scale2
scale2 <first point> <last point> <real value> <imaginary value>
Multiply complex data by <real value> + i*<imaginary value> between <first point> to <last point> (inclusive).

mask_ppmm
mask_ppmm
Apply ppmm mask (++–) i.e. the sign of points 3 and 4 and every third and fourth point equivalently is changed. Needed for some Bruker data.

mirror_zero
mirror_zero
Mirrors the data for zero dwell time by taking the complex conjugate and reversing the data, excluding the first point. Requires complex data.

mirror_half
mirror_half
Mirrors the data for zero dwell time by taking the complex conjugate and reversing the data, including the first point. Requires complex data.

cycle
cycle <cycle amount>
This cycles the data by <cycle amount>. If the data is complex, <cycle amount> should be specified in terms of complex points.

Data range

lower
lower <lower bound>
This shifts data so that the origin is now at <lower bound>. The first point is point 1.

upper
upper <upper bound>
This truncates data so that its last point is <upper bound>. The first point is point 1.

range
range <lower bound> <upper bound>
This shifts and truncates data so that the origin is now at <lower bound> and so that the last point is <upper bound>. The first point is point 1.

fft

zerofill
zerofill <n>
Data is padded with zeros giving a final size of 2^n times the original size where $n \leq 4$

fft
fft
Complex Fourier transform. The data must be complex. If the input data is not a power of two, it is zero-padded to the nearest power of two.

ifft
ifft
Inverse complex Fourier transform. The data must be complex. If the input data is not a power of two, it is zero-padded to the nearest power of two.

hft
hft
Hilbert Fourier transform. The data must be real. If the input data is not a power of two, it is zero-padded to the nearest power of two.

fftn
fftn
Normalised complex Fourier transform. The data must be complex. If the input data is not a power of two, it is zero-padded to the nearest power of two.

ifftn
ifftn
Normalised complex inverse Fourier transform. The data must be complex. If the input data is not a power of two, it is zero-padded to the nearest power of two.

Apodisation

decay
decay <end value>
Multiplies the data by a decaying exponential which is 1 at point 1 and <end value> at point n.

decay_sw
decay_sw <line broadening (Hz)> <spectral width (Hz)>
Multiplies the data by a decaying exponential, $\exp(-\pi * LB * t)$, where LB is <line broadening (Hz)>. Spectral width is used to convert between points, x, and time values, t, using the function $t = \frac{x}{(2*SW)}$.

sinebell
sinebell <angle>
This multiplies the data by a sine function with the given <angle> (specified in degrees) at point 1, and, if there are n (complex/real) points, with angle 180 degrees at point n+1 (*not* point n).

sinebell2
sinebell2 <angle>
This multiplies the data by a sine function squared with the given <angle> (specified in degrees) at point 1, and, if there are n (complex/real) points, with angle 180 degrees at point n+1 (*not* point n).

gaussian_sw
gaussian_sw <line broadening (Hz)> <sharpening factor> <spectral width (Hz)>

This multiplies the data by the gaussian $\exp(a + bt + ct^2)$. If LB is the entered line broadening (in Hz) and s is the sharpening factor, then $a = -\ln 2/s^2$, $b = \pi \times LB$ and $c = -(\pi \times LB \times s)^2/4\ln 2$. Note that LB must be **positive** for normal use, which is the **opposite** of the Bruker convention. The function converts a Lorentzian line (or multiplet) of width FWWH equal to LB to a Gaussian line of width FWWH equal to $s \times LB$. The maximum value of the multiplying function is 1, which is obtained at the fraction of the acquired time $t/T = 2\ln 2/(LB \times \pi \times T \times s^2)$. The spectral width (SW) is needed by the program in order to translate between point number, x, and the time value, t, used in the function, by $t = x/(2 \times SW)$. If a value of $SW = 0.0$ is given, the program uses the value for SW which was entered in the par file (or the default). The suggested range of values for s are from 1.3 to 0.7 (possibly 0.5).

Phase

phase phase <phase0> <phase1>

Phase complex data using the zero order correction, <phase0>, and first order correction, <phase1> (both specified in degrees), with the pivot at point 1, where point 1 is the first point in Azara nomenclature.

phase2
phase2 <phase0> <phase1> <pivot>

Phase complex data using the zero order correction, <phase0>, and first order correction, <phase1> (both specified in degrees), with pivot at point <pivot>, where the first point is point 1.

Baseline correction

base_const
base_const <half width>

This fits the baseline of a spectrum (not fid) by first finding baseline points using a shifting window of half width <half width> and then fitting a constant to those baseline points.

base_const2
base_const2 <half width> <first point> <last point>

This fits the baseline of a spectrum (not fid) by first finding baseline points using a shifting window of size <half width> and then fitting a constant to those baseline points.
Only the points from <first point> to <last point> (inclusive) are fitted.

base_poly
base_poly <half width> <degree>

This fits the baseline of a spectrum (not fid) by first finding baseline points using a shifting window of size <half width> and then fitting a polynomial of degree (order) <degree> to those baseline points.

base_poly2
base_poly2 <half width> <degree> <first point> <last point>

This fits the baseline of a spectrum (not fid) by first finding baseline points using a shifting window of size <half width> and then fitting a polynomial of degree (order) <degree> to those baseline points. Only the points from <first point> to <last point> (inclusive) are fitted.

base_trig
base_trig <half width> <order>

This fits the baseline of a spectrum (not fid) by first finding baseline points using a shifting window of size <half width> and then fitting trig functions of order <order> to those baseline points. In fact, the number of functions used to do the fitting is 2*<order> + 1, the 2 being because cosines and sines are used, and the 1 being because a constant is also used. This fit only makes sense if the original data set contains the entire recorded spectral width.

base_trig2
base_trig2 <half width> <order> <first point> <last point>

This fits the baseline of a spectrum (not fid) by first finding baseline points using a shifting window of size <half width> and then fitting trig functions of order <order> to those baseline points. In fact, the number of functions used to do the fitting is 2*<order> + 1, the 2 being because cosines and sines are used, and the 1 being because a constant is also used. Only the points from <first point> to <last point> (inclusive) are fitted. This fit only makes sense if the original data set contains the entire recorded spectral width.

base_points
base_points <base points file>

This uses the points in the <base points file> to define the baseline points for subsequent baseline correction routines (by default the baseline is determined automatically by the program). There must be a matching end_base_points before the next use of base_points or before a resumption of the automatic baseline determination.

end_base_points
end_base_points

This matches the most recent base_points and must occur before the next use of base_points or before a resumption of automatic baseline determination (the default).

base_subtract
base_subtract <base subtract file>

This subtracts the values given in <base subtract file> from the data. If there are n points then there must be n values in <base subtract file> (in free format).

base_subtract2
base_subtract2 <base subtract file> <first point> <last point>

This subtracts the values given in <base subtract file> from the data, from <first point> to <last point> (inclusive). If n = <last point> - <first point> + 1, then there must be n values in <base subtract file> (in free format).

Linear prediction

lp_extend
lp_extend <number predicted> <length of sequence> <cutoff>

When <number predicted> is positive, this uses linear prediction to extend the existing data from the end by <number predicted>.

When <number predicted> is negative, this does a linear prediction by <|number predicted|> points inserted at the beginning of the existing data.

Data is assumed to be complex with <number predicted> and <length of sequence> specified in complex points. <cutoff> takes values between 0 and 1; singular values below cutoff are set to 0. Recommended values for <length of sequence> are $\min(\frac{n}{4} \text{ to } \frac{n}{3}, 15 \text{ to } 20)$ where n is the number of complex points prior to linear prediction. The recommended value for <cutoff> is 0.0001. For more details see `<http://www2.ccpn.ac.uk/azara/azara_docs/process.html>` and Zhu and Bax, 1992, J. Magn. Reson. 100, 202-207.

lp_forward
lp_forward <number predicted> <number of poles>
Predicts <number predicted> points from the end of the existing data. For complex data, specify in terms of complex points. Real and imaginary points are fitted separately. <number of poles> is an upper bound for the number of oscillators in a given row. A typical value is 5 or 10.

lp_backward
lp_forward <number predicted> <number of poles>
Predicts the first <number predicted> points, overwriting the existing first <number predicted> points. For complex data, specify in terms of complex points. Real and imaginary points are fitted separately. <number of poles> is an upper bound for the number of oscillators in a given row. A typical value is 5 or 10.

lp_first
lp_first <number predicted> <length of sequence> <number of sequences> <cutoff>
Linear prediction of the first <number predicted> points, using <number of sequences> sequences of length <length of sequence>, by overwriting the first <number predicted> points. Specify in terms of complex points. Real and imaginary points are fitted separately. <number of sequences> must be ≥ <length of sequence>. Singular value decomposition is used and <cutoff> (between 0 and 1) determines which singular values are significant.

lp_first2
lp_first2 <number predicted> <length of sequence> <number of sequences> <cutoff>
Linear prediction of the first <number predicted> points, using <number of sequences> sequences of length <length of sequence>, by overwriting the first <number predicted> points. Specify in terms of complex points. For complex data, real and imaginary points are fitted as one complex point. <number of sequences> must be ≥ <length of sequence>. Singular value decomposition is used and <cutoff> (between 0 and 1) determines which singular values are significant.

lp_last
lp_last <number predicted> <length of sequence> <number of sequences> <cutoff>
Linear prediction of the last <number predicted> points, using <number of sequences> sequences of length <length of sequence>, by extending the existing data. Specify in terms of complex points. Real and imaginary points are fitted separately. <number of sequences> must be ≥ <length of sequence>. Singular value decomposition is used and <cutoff> (between 0 and 1) determines which singular values are significant.

lp_last2
lp_last2 <number predicted> <length of sequence> <number of sequences> <cutoff>
Linear prediction of the last <number predicted> points, using <number of sequences> sequences of length <length of sequence>, by extending the existing data. Specify in terms of complex points. For complex data, real and imaginary points are fitted as one complex point. <number of sequences> must be ≥ <length of sequence>. Singular value decomposition is used and <cutoff> (between 0 and 1) determines which singular values are significant.

Chapter 5: Spectrum viewer

A simple spectrum viewer is included with the software to aid visualisation of processed data. The spectrum viewer can be opened in two ways. If the code is run directly from the GUI, the spectrum viewer will open automatically once the reconstruction is completed (assuming post-processing is selected). Alternatively, the viewer can be launched stand-alone using the script *graphPlot* in the *bin* directory.

The spectrum viewer reads Azara-style *.spc.par* files (see below) and can display 2, 3 and 4D datasets.

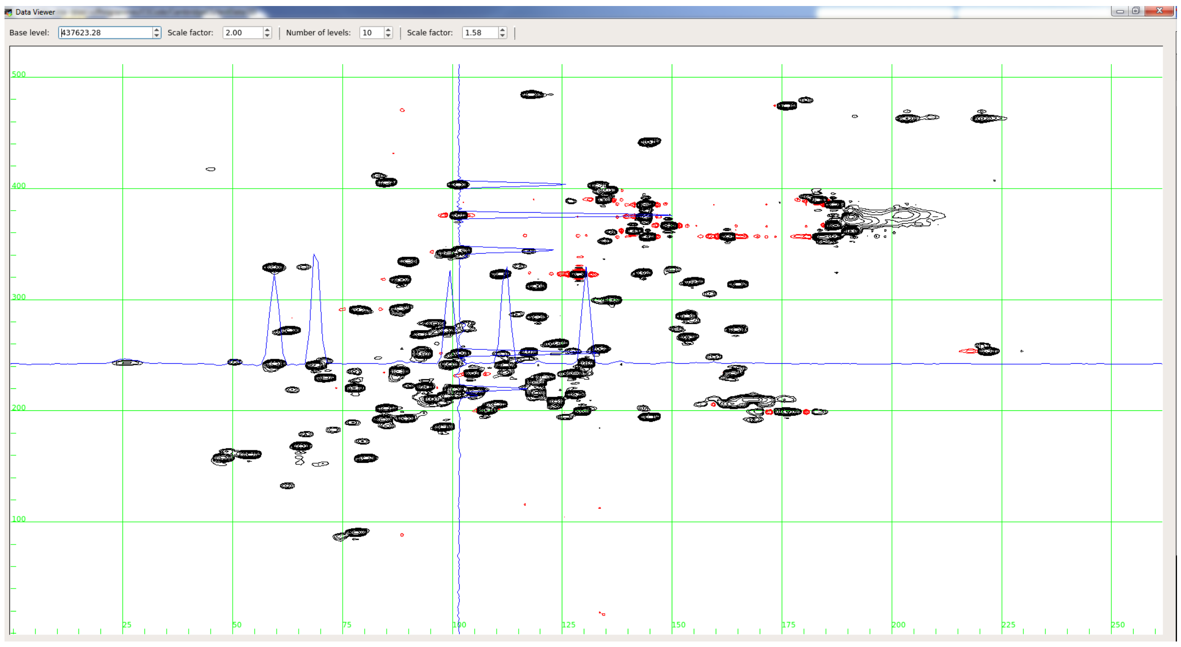
```
ndim 2
file 102_ft_full_postpro_block.spc

dim 1
npts 262
block 64
sw 1000.000
sf 500.000
refppm 0.0000
refpt 1.0000
nuc 1H

dim 2
npts 256
block 64
sw 1000.000
sf 500.000
refppm 0.0000
refpt 1.0000
nuc 1H
```

In each dimension the ‘dim’, ‘npts’ and ‘block’ options are essential. Other parameters contain default values which may be read by Azara plot2. The spectrum viewer is designed for simple visual analysis of the processed spectrum; currently shift referencing is not supported. An example window for a 2D HSQC spectrum is shown in Figure 5.1

1D traces are shown in blue, positive contours in black and negative contours in red. The base level and number of displayed contour levels can be set using the ‘Base level’ and ’Number of levels’ options at the top of the window. In each case a scale factor can be set. For the contour levels, this is multiplicative and determines the change between each contour level for the given number of levels. For the base level, this is also multiplicative and determines how much the base level changes on increasing or decreasing. Some useful commands are given below:



Base level: 437623.28 Scale factor: 2.00 | Number of levels: 10 | Scale factor: 1.58

Figure 5.1: 2D spectrum display window.

Command	Function
↑ / ↓	Increase/decrease the base level by the scale factor
f / d	Scale up/down the 1D traces
shift + left mouse drag	Zoom into the selected region
ctrl + left mouse drag	Zoom into the selected region
Double-click (left-mouse buttom)	Zoom out to the full spectrum size
q	quit viewer
Esc	quit viewer

Table 5.1: Commands for use with the spectrum viewer

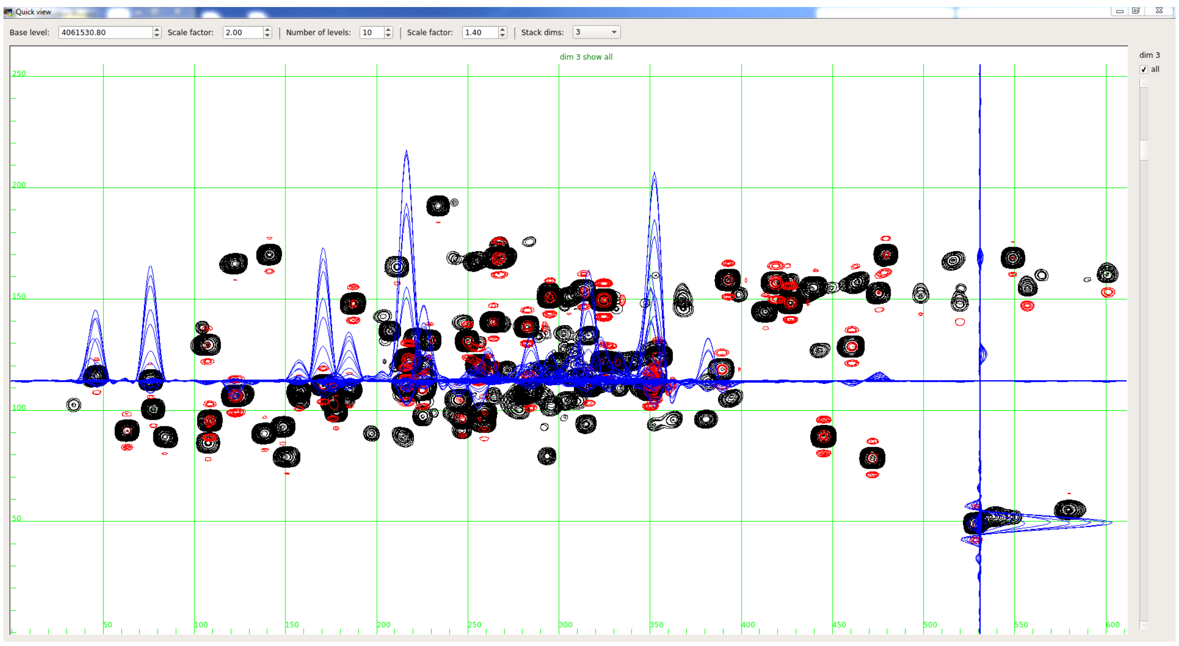

Base level: 4061530.80 | Scale factor: 2.00 | Number of levels: 10 | Scale factor: 1.40 | Stack dims: 3

Figure 5.2: 3D spectrum display window. ‘All’ (right-hand side) is selected to stack all the planes in dimension 3 (indicated by ‘Stack dims:’ on the top toolbar).

For n-dimensional spectrum with $n > 2$, two dimensions are displayed and the remaining dimensions are ‘stacked’. The choice of stacked dimension can be changed using the ’Stack dims’ dropdown menu. By default all the planes from the stacked dimension are shown as indicted by the ‘all’ checkbox on the right hand side. If this is unticked, the slider can then be used to scroll through the stacked dimension(s). The slice selected is shown at the top of the spectrum display window. An example of a 3D display window with all planes in dimension 3 stacked is shown in Figure **??**. An example with a single plane displayed is shown in Figure **??**.

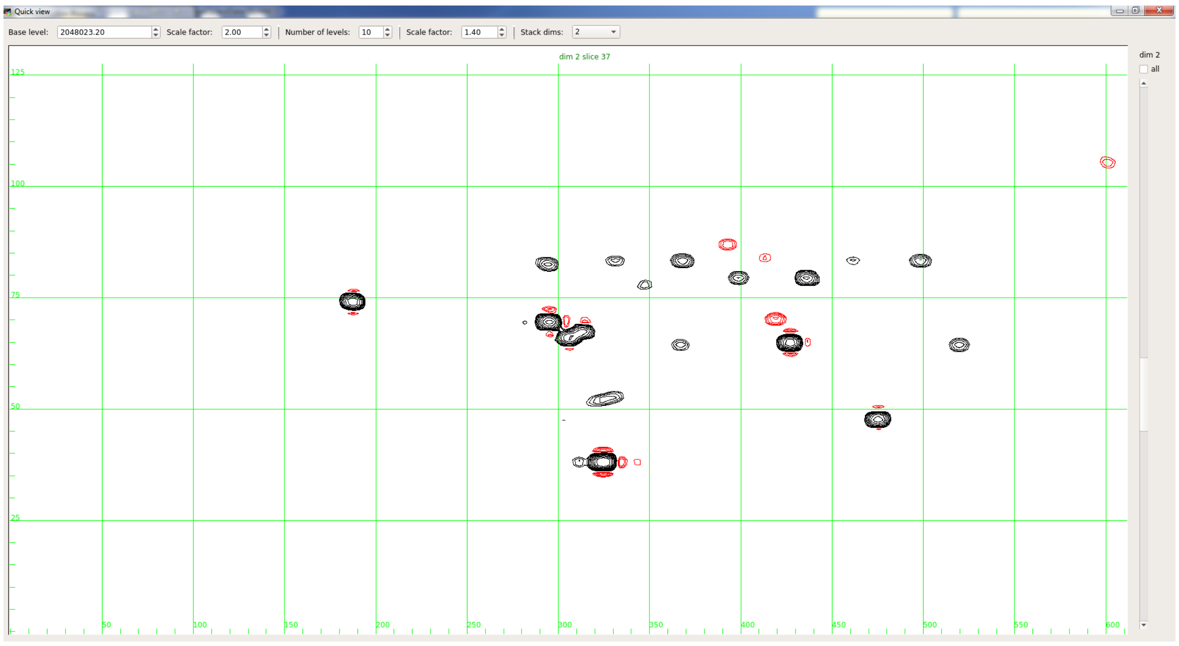

Base level: 2048023.20 Scale factor: 2.00 | Number of levels: 10 | Scale factor: 1.40 | Stack dims: 2

Figure 5.3: 3D spectrum display window for a single plane in the first indirect dimension (dimension 2, plane 37). ‘All’ (right-hand side) is now unticked and the slider used to scroll to the desired slice.

Appendices

Appendix A: Lookup table for Bruker Avance data

Lookup values are given in the main body of the table for a particular (DSPFVS, DECIM) coordinate. Using the lookup value from the table below, groupDelay=0.5×lookup/DECIM. The first order phase correction applied is 360×<GRPDLY>. For more details see the **brukerGroupPhase** command in section 4.3.

DSPFVS	10	11	12	13	20
DECIM					
2	179	184	184	11	-1
3	201	219	219	17	-1
4	533	384	384	23	-1
6	709	602	602	35	-1
8	1097	852	852	47	-1
12	1449	1668	1668	71	-1
16	2225	2312	2292	95	-1
24	2929	3368	3368	143	-1
32	4481	4656	4616	191	-1
48	5889	6768	6768	287	-1
64	8993	9344	9264	383	-1
96	11809	13568	13568	575	-1
128	18017	18560	18560	-1	-1
192	23649	27392	27392	-1	-1
256	36065	36992	36992	-1	-1
384	47329	55040	55040	-1	-1
512	72161	73856	73856	-1	-1
768	94689	110336	110336	-1	-1
1024	144353	147584	147584	-1	-1
1536	189409	220928	220928	-1	-1
2048	288737	295040	295040	-1	-1
2080	-1	-1	-1	-1	282814

Appendix B: Test Data

Test data are contained in the directory *testData*, with processing scripts included to illustrate processing methods.

B.0.1 2D, ^{15}N HSQC, ubiquitin

This data set was recorded on a Bruker DRX500 spectrometer. Change into the sub-directory *2D*. The raw data is contained in the folder *102* in this directory. There are two scripts set-up; *102_ft_full.scr* and *102_cs_nus.scr*. *102_ft_full.scr* is used to demonstrate FT processing of a fully-sampled data set. To load the processing script into the GUI for the fully-sampled HSQC, run the command

```
cambridgecs -g 102_ft_full.scr
```

Looking at the GUI display, note that the *Directory* set, *..../2D*, is the directory where output files will be stored. Change this on the computer being used by clicking *Select* and choosing the relevant directory. Since the raw data is in the directory *102*, the *Raw data directory* parameter is set. This option is available in the *Extra Parameters* drop down menu. An alternative approach would be to set the top level *Directory* to *.../2D/102*. In this case the processed data will be stored in the same directory as the raw data. Since the data is fully-sampled, the *Uniform sampling flag* is set, also available from *Extra Params*.

The pre-processing script contains commands for FT processing of the direct dimension. Since the data was recorded on a Bruker DRX machine, the parameters DECIM and DSPFVS (found in the Bruker *acqus* file) should be converted to a group delay value and used with the command brukerGroupPhase. Here DECIM = 16 and DSPFVS = 12 and the lookup value (see A) is 2292. This gives brukerGroupPhase = $0.5 \times 2292/16 = 71.625$.

'Reconstruction script' contains commands for processing the indirect dimensions. Here *FT* is selected from the dropdown menu since the data is fully-sampled and the data can be processed with the fast Fourier transform. In addition, the flag *max_points* must be set (here 256, the total number of points in the indirect dimension), since there is no NUS list for a fully-sampled data set to specify the matrix size.

Post-processing commands are used to add additional baseline corrections using Azara. The input data may be specified here as the FT data i.e. the frequency domain output from the reconstruction.

The button, 'Open script editor' shows the text version of the script, which can be easily adapted to other spectra.

The reconstruction is run by clicking the button *Process current script*. A terminal window displays the progress of the reconstruction with information about the reconstruction parameters and output files. The progress bar on the GUI tracks the progress of the reconstruction (note this does not include the Azara post processing). If Azara post-processing is set, the output spectrum will be displayed in the simple viewer.

The script *102_cs_nus.scr* demonstrates artificial undersampling of a fully-sampled data set, which may be used in test cases to determine how a particular spectrum responds to different sampling schedules and undersampling factors. The reconstruction is setup largely as above. However, the *Uniform sampling flag* and *max_points* are no longer set. Instead two NUS lists are provided; *vclist_full* and *vclist*. *vclist_full* is the *Original NUS list* which specifies the positions of points in the recorded data. *vclist* is the *NUS list* used for undersampling, in this case ∼30% sampling following an decaying exponential schedule. The reconstruction is carried out with the IHT algorithm, using the default parameters.

B.0.1.1 NMRPipe

In the folder *2D* there is a subfolder, *NMRPipe*, containing NMRPipe processing scripts. The data is prepared using the script *fid.com* which was prepared using the assistance of the 'Bruker' macro:

```
#!/bin/csh

bruk2pipe -in ../102/ser \
  -bad 0.0 -noaswap -DMX -decim 16 -dspfvs 12 -grpdly 0  \
  -xN              1024  -yN               256  \
  -xT               512  -yT               128  \
  -xMODE            DQD  -yMODE        States-TPPI  \
  -xSW        10000.000  -ySW         2000.000  \
  -xOBS          500.01 -yOBS         50.665588  \
  -xCAR           4.679  -yCAR             120  \
  -xLAB              HN  -yLAB             15N  \
  -ndim               2  -aq2D          States  \
  -out ./test.fid -verb -ov
```

Direct dimension processing is carried out using the script *nusproc.com.* Importantly, CambridgeCS follows the Azara convention of processing in place along each axis, so there is no need for a transpose statement at the end of the 1D script.

```
#!/bin/csh

nmrPipe -in ./test.fid \
| nmrPipe  -fn SOL                                    \
| nmrPipe  -fn SP -off 0.5 -end 0.98 -pow 2 -c 0.5    \
| nmrPipe  -fn ZF -auto                               \
| nmrPipe  -fn FT                                     \
| nmrPipe  -fn PS -p0 237  -p1 33 -di                 \
| nmrPipe  -fn EXT -x1 220 -xn 480 -sw -verb          \
   -ov -out test.nus
```

The output from the NMRPipe 1D processing is read into CambridgeCS for reconstruction of the indirect dimensions. In this case, there is no need to setup 'Pre-processing parameters' (pre-processing has been carried out here using NMRPipe). Reconstruction of the indirect dimension is carried out using the commands in the 'Reconstruction parameters' section. Following this, post-processing may be carried out either by selecting Azara post processing commands or using NMRPipe. Here we use NMRPipe:

```
# General properties:
    directory                          /home/path/to/CambridgeCS/testData/2D/NMRPipe
    outputFile                         102_pipe_cs_2
    dataInput                          /home/path/to/CambridgeCS/
                                                testData/2D/NMRPipe/test.nus
    rawDirectory                       ../102
    nmrPipe
    uniform_sampling
    head                               512

    direct-dim:
        npts                           261
    end_script

    indirect-dims:
        npts                           256
        ndim                           1
    end_script

# Pre-processing parameters:

# Reconstruction parameters:
    reconstruction_script:
        reconstruction_flag            FT

        dim 2:
            zerofill                   1
            phase                      180.0    -360.0
            mask_ppmm
            max_points                 256
            sinebell                   90.0
        end_script
    end_script

# Azara post processing instructions:
```

In this example 'uniform sampling' is selected as the data is fully-sampled. In addition the 'nmrPipe' flag is set along with the header flag 'head 512' since all NMRPipe files have a header of length 512 bytes. After reconstruction the processed data will be displayed in the graph viewer window, although note that this does not include any post-processing commands. In addition, a small output script is created and run which converts the data back to NMRPipe format *102_pipe_cs_2.com*:

```
#!/bin/csh
#
bin2pipe -in /home/path/to/CambridgeCS/testData/
                        2D/NMRPipe/102_pipe_cs_2_recon.spc -bad 0.0 -noswap \
-xN 261 \
-xT 261 \
-xFT Freq \
-xMODE Real \
-xSW 2548.828125 \
-xORIG 2661.812500 \
-xOBS 500.010010 \
-xLAB HN \
-yN 512 \
-yT 512 \
-yFT Freq \
-yMODE Real \
-ySW 2000.000000 \
-yORIG 5095.495605 \
-yOBS 50.665588 \
-yLAB 15N \
-ndim 2 \
-out /home/path/to/CambridgeCS/testData/2D/NMRPipe/102_pipe_cs_2.pipe.ft2 -ov
```

The final NMRPipe output file is '102_pipe_cs_2.pipe.ft2'. Further post-processing commands may be added to this or other NMRPipe scripts as needed. Note that if Azara post-processing is selection, the conversion to a NMRPipe output file is also carried out. However, the spectrum displayed in the graph viewer will be the result of the Azara post-processing.

B.0.2 3D ^{13}C NOESY, sensory rhodopsin II (pSRII)

This data set was recorded on a Bruker AV800 spectrometer equipped with a cryoprobe. As for the 2D data set discussed above *Directory* refers to the output directory where the processed data will be saved, whilst *Raw data directory* refers to the directory containing the raw data. The data here is undersampled, using a 25% exponentially decaying NUS schedule. The parameter *nuslist* points to the file *vclist*. The file will be looked for in the *Raw data directory* and this specifies the list of acquired points in the form, where x and y correspond to coordinates for the two indirect dimensions:

```
x1
y1
x2
y2
x3
y3
...
xn
yn
```

The *nuslist* option can be found under *Extra params*. This also contains the option *Threads*. This dictates how many CPU threads to use. The default is set to *1*. If this value is set to *0*, the maximum available will be used. Reconstructions of multidimensional data sets can be speeded up considerably by increasing the number of threads.

Since the data is recorded on and Bruker AV instrument, the *GRPDLY* parameter in the *acqus* file is used to correct for the first order phase error. This is done automatically with the *brukerGroupPhase* command set to *-1*.

As the data is undersampled, reconstruction is carried out using one of the compressed sensing algorithms; here iterative hard thresholding (IHT) is used. The stop flag is set to *l2*; the algorithm compares the l2 norm between successive iterations and will stop when there is no significant change in this value. The IHT may be run with default settings. In this case, the threshold has been increased to 85% of the maximum peak height (0.85). Processing parameters for the two indirect dimensions are set under *dim 2* and *dim 3* and a post-processing script with baseline corrections is also included.

B.0.2.1 NMRPipe

As previously, NMRPipe processing scripts are contained in the subdirectory 'NMRPipe'. For a 3D experiment, each indirect dimension coordinate has four FIDs representing the different quadrature detection components. Thus the y-dimensions has size 4 and the z axis has size determined by the number of coordinates in the *vclist*. The direct dimension processed data is written out as individual 2D planes, which could be used for phasing using for example NMRDraw.

```
#!/bin/csh

bruk2pipe -in ../1734/ser \
  -bad 0.0 -aswap -DMX -decim 2000 -dspfvs 20 -grpdly 67.9862518310547  \
  -xN              1024  -yN                 4  -zN               480  \
  -xT               512  -yT                 2  -zT               240  \
  -xMODE            DQD  -yMODE        Complex  -zMODE        Complex  \
  -xSW        10000.000  -ySW         2000.000  -zSW         3263.708  \
  -xOBS         800.134  -yOBS         800.134  -zOBS         201.197  \
  -xCAR           4.773  -yCAR           4.773  -zCAR          22.038  \
  -xLAB             1Hx  -yLAB              1H  -zLAB             13C  \
  -ndim               3  -aq2D          States                         \
  | pipe2xyz -x -out ./fid.nus/test%03d.fid -verb -ov -to 0
```

Further processing is carried out using the script *csnusproc.com*. Once again, no transpose command is required at the end of the 1D processing script:

```
#!/bin/csh

xyz2pipe -in fid.nus/test%03d.fid -x \
| nmrPipe -fn SOL \
| nmrPipe  -fn SP -off 0.5 -end 0.98 -pow 2  \
| nmrPipe  -fn ZF -auto                       \
| nmrPipe  -fn FT -verb                          \
| nmrPipe  -fn PS -p0 282  -p1 0 -di            \
| nmrPipe  -fn EXT -left -sw           \
 -ov -out testcs.nus -x
```

Indirect dimension processing is carried out using CambridgeCS with the script *1734_pipe_cs.scr*. Again the *nmrPipe* and *head* flags are set:

```
# General properties:
    directory                       /home/path/to/CambridgeCS/testData/
                                              3D/13CNOESY/NMRPipe
    outputFile                      1734_pipe_cs
    dataInput                       /home/path/to/CambridgeCS/testData/
                                              3D/13CNOESY/NMRPipe/testcs.nus
    rawDirectory                    ../1734
    nusList                         vclist
    nmrPipe
    head                            512

    direct-dim:
        npts                        512
    end_script

    indirect-dims:
        npts                        1920
        ndim                        2
    end_script

# Pre-processing parameters:

# Reconstruction parameters:
    reconstruction_script:
        reconstruction_flag         IHT
        stop_flag                   l2

        dim 2:
            zerofill                2
            phase                   45.0    0
            mask_ppmm
            sinebell                60.0
        end_script

        dim 3:
            zerofill                1
            phase                   0    0
            sinebell                60.0
        end_script
    end_script

# Azara post processing instructions:
```

The NMRPipe script *1734_pipe_cs.com* converts the data back to NMRPipe format.

B.0.3 3D SOFAST (Best) [^{15}N,^{1}H] TROSY HNCO

The data was recorded on a Bruker DRX500 spectrometer. The reconstruction is set up similar to the ^{13}C NOESY described above. In this case, 22.7% sampling is used with an exponentially decaying distribution in the decaying ^{13}C dimension (dim 3). The ^{15}N dimension is recorded with P/N selection; P/N data is handled in the processing by selecting the interlace option from the 'indirect-dims', 'Extra params' drop down menu. *Interlace* is set to 2 to convert dim 2 P/N points to cos/sin points. As for the 2D HSQC discussed above, the first order phase correction is determine from the DECIM and DSPFVS parameters in the *acqus* file. Here DECIM = 16 and DSPFVS = 12 and the lookup value (see A) is 2292. This gives brukerGroupPhase = $0.5 \times 2292/16 = 71.625$.

The reconstruction is carried out using the IHT algorithm. *Global_threshold* is selected from the *Extra Params* dropdown menu and set to 1×10^{-4}. This lowers the minimum threshold for the reconstruction, which may increase the accuracy of the reconstruction, but also increases the reconstruction time and may lead to reconstruction of noise components.

B.0.3.1 NMRPipe

NMRPipe processing proceeds as described above for the 3D NOESY. However, dimension 2 (the first indirect dimension (^{15}N)) is acquired as echo-antiecho data (P-/N-type). In order to avoid the phase-twist line-shape, this data must be converted to cosine-/sine-type data. This is achieved using the Rance-Kay macro provided with NMRPipe which is called using the following command:

```
| nmrPipe -fn MAC -macro $NMRTXT/ranceY.M -noRd -noWr   \
```

ranceY.m is used since dimension 2 (y-dimension) needs converting. The full script (*fid.com*) is:

```
#!/bin/csh

bruk2pipe -in ../550/ser \
  -bad 0.0 -noaswap -DMX -decim 16 -dspfvs 12 -grpdly -1  \
  -xN              1024  -yN             4     -zN              350  \
  -xT               512  -yT             2     -zT              175  \
  -xMODE            DQD  -yMODE       Real     -zMODE          Real  \
  -xSW        10000.000  -ySW     2530.364     -zSW        1666.667  \
  -xOBS         500.132  -yOBS      50.678     -zOBS        125.757  \
  -xCAR             4.7  -yCAR     120.000     -zCAR        176.000  \
  -xLAB             1Hx  -yLAB         15N     -zLAB            13C  \
  -ndim               3  -aq2D      States                           \
| nmrPipe -fn MAC -macro $NMRTXT/ranceY.M -noRd -noWr   \
| pipe2xyz -x -out ./fid/test%03d.fid -verb -ov -to 0
```

Other processing scripts are as previously described, however note that since the Echo-Antiecho data is handled by NMRPipe the *interlace* command should not be set in CambridgeCS.

B.0.4 4D, HCCH NOESY, pSRII

The data was recorded on a Bruker AV800 spectrometer, equipped with cryoprobe. The reconstruction is set up as in previous examples, with the GRPDLY value set to brukerGroupPhase = $0.5 \times 2292/16 = 71.625$.

Reconstruction of a 4D using 12 threads takes around 5 h. For a quick view of the spectrum, it is recommended to change the reconstruction type to **FT**. This will allow a quick overview to check phasing and general spectral appearance before a longer run. It is advised to run longer reconstructions in automatic mode. No user prompts are required and the process can be disconnected from the terminal and run in the backgroup. An example command is given below:

```
nohup nice -n 10 cambridgecs -a 1664_cs.scr &
```

nohup disconnects the process from the terminal ('no hangup' command will be sent if the terminal is closed). *nice -n 10* sets the 'niceness' for the process. Desktop processes receive priority with a value of 0. *cambridgecs -a* runs the process in automatic mode (*-a*) from the command line.

Appendix C: Citation and License

Please cite the following paper when presenting results processed using this software:

Mark J. Bostock, Daniel J. Holland and Daniel Nietlispach (2012), Compressed sensing reconstruction of undersampled 3D NOESY spectra: application to large membrane proteins. J. Biomol NMR, 54:15-32.

CambridgeCS (c) by Mark J. Bostock, Robert Tovey and Daniel Nietlispach, Department of Biochemistry, University of Cambridge, UK.

License:

By using CambridgeCS software, the following requirements and conditions are implicitly accepted by the user:

1. The person acquiring or using this software is authorized by his Workplace and Organisation to commit to this agreement, which is binding on this person, this person's Workplace, Organisation, and all other persons and institutions with whom this software is shared.

2. This agreement covers all CambridgeCS software in any form, but does not apply to software that is explicitly distributed under a different license provided that the alternative license is clearly reflected in the individual files that it covers.

3. CambridgeCS software is provided free of charge to academic users. A fee is applicable for commercial use.

4. CambridgeCS software, once acquired, may be shared with others working in the same Workplace as the original acquirer. The software must be accompanied by this license.

5. Except as stated in 4. CambridgeCS software, whether in whole or in part, modified or unmodified, may not be redistributed in any way.

6. CambridgeCS software is provided 'as is' and is without warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement. The authors will not be held liable for any use you make of the software.

7. Acknowledgment shall be made in all publications resulting from the use of CambridgeCS software. Please reference:

Mark J. Bostock, Daniel J. Holland and Daniel Nietlispach (2012), Compressed sensing

reconstruction of undersampled 3D NOESY spectra: application to large membrane proteins. J. Biomol NMR, 54:15-32.

References

[1] J. C. J. Barna, E. D. Laue, M. R. Mayger, J. Skilling, and S. J. P. Worrall, *J. Magn. Reson.*, 1987, **73**, 69–77.

[2] B. E. Coggins and P. Zhou, *J. Biomol. NMR*, 2008, **42**, 225–239.

[3] V. Y. Orekhov, I. V. Ibraghimov, M. Billeter, *J. Biomol. NMR*, 2001, **20**, 49–60.

[4] D. Rovnyak, D. P. Frueh, M. Sastry, Z.–Y. J. Sun, A. S. Stern, J. C. Hoch, and G. Wagner, *J. Magn. Reson.*, 2004, **170**, 15–21.

[5] P. Schmieder, A. Stern, G. Wagner, and J. Hoch, *J. Biomol. NMR*, 1993, **3**, 569–576.

[6] P. Schmieder, A. S. Stern, G. Wagner, and J. C. Hoch, *J. Biomol. NMR*, 1994, **4**, 483–490.

[7] V. Tugarinov, L. E. Kay, I. V. Ibraghimov, and V. Y. Orekhov, *J. Am. Chem. Soc.*, 2005, **127**, 2767–2775.

[8] M. R. Palmer, C. L. Suiter, G. E. Henry, J. Rovnyak, J. C. Hoch, T. Polenova, and D. Rovnyak, *J. Phys. Chem. B*, 2015, **119**, 6502–6515.

[9] B. F. Logan, Columbia University, New York, 1965.

[10] D. L. Donoho, *IEEE T. Inform. Theory.*, 2006, **52**, 1289–1306.

[11] E. J. Candès, J. Romberg, and T. Tao, *IEEE T. Inform. Theory.*, 2006, **52**, 489–509.

[12] M. Lustig, D. L. Donoho, and J. M. Pauly, *Magn. Reson. Med.*, 2007, **58**, 1182–1195.

[13] D. J. Holland, M. J. Bostock, L. F. Gladden, and D. Nietlispach, *Angew. Chem. Int. Ed.*, 2011, **50**, 6548–6551.

[14] K. Kazimierczuk and V. Y. Orekhov, *Angew. Chem. Int. Ed.*, 2011, **50**, 5556–5559.

[15] S. G. Hyberts, A. G. Milbradt, A. B. Wagner, H. Arthanari, and G. Wagner, *J. Biomol. NMR*, 2012, **52**, 315–327.

[16] M. J. Bostock, D. J. Holland, and D. Nietlispach, *J. Biomol. NMR*, 2012, **54**, 15–32.