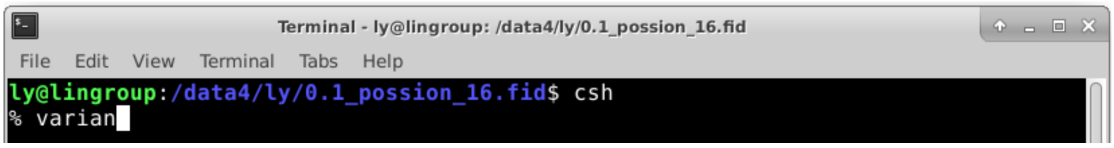
Instructions for using JTF-Net to reconstruct spectra

1. 2D NMR spectra

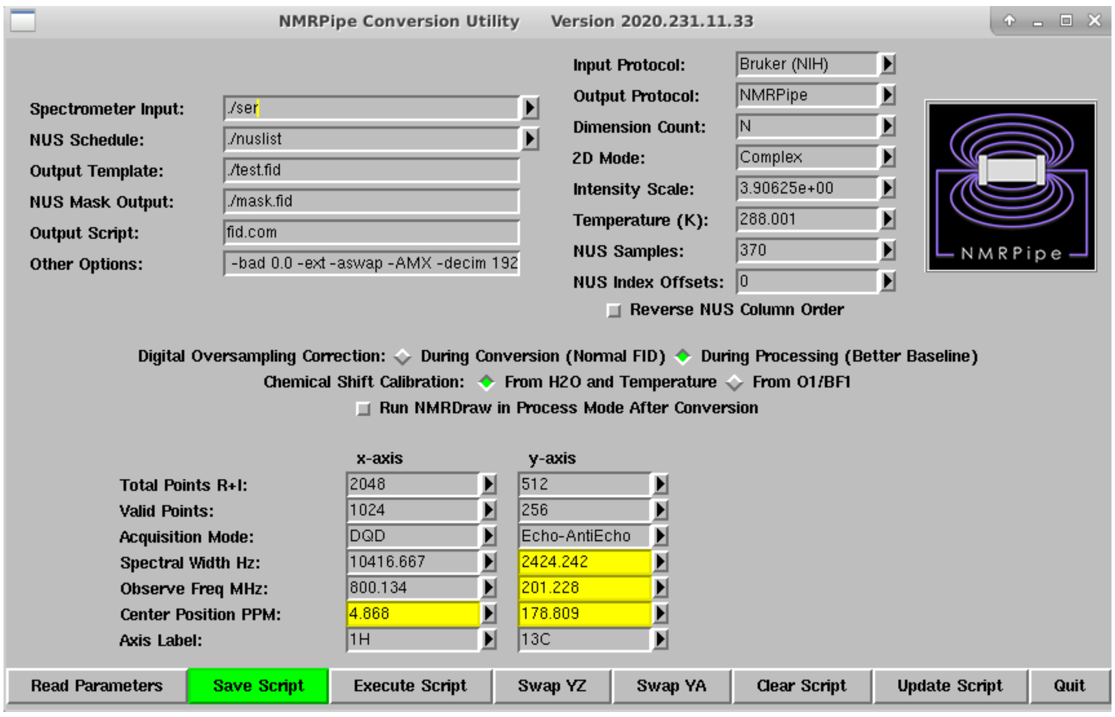
1.1 Test

1.1.1 Preprocess

1) Firstly, copy the NUS data and its sampling scheme to the NUS_data/2D/. Use NMRPipe to automatically read NUS data. Enter "varian" or "bruker" (depending on the data type) in the NMRPipe terminal, as shown in Figure R1. Select the corresponding NUS data and undersampling scheme. Click "Read Parameters" and "Save Script" to automatically generate a script named fid.com. After running fid.com, you can obtain NUS data with unsampled points replaced by 0, which will be saved as "test.fid" in NMRPipe format.



```
ly@lingroup:/data4/ly/0.1_possion_16.fid$ csh
% varian
```



Spectrometer Input:	./ser
NUS Schedule:	./nuslist
Output Template:	./test.fid
NUS Mask Output:	./mask.fid
Output Script:	fid.com
Other Options:	-bad 0.0 -ext -aswap -AMX -decim 192

Input Protocol:	Bruker (NIH)
Output Protocol:	NMRPipe
Dimension Count:	N
2D Mode:	Complex
Intensity Scale:	3.90625e+00
Temperature (K):	288.001
NUS Samples:	370
NUS Index Offsets:	0

Reverse NUS Column Order

Digital Oversampling Correction: During Conversion (Normal FID) [✓ During Processing (Better Baseline)]

Chemical Shift Calibration: [✓ From H2O and Temperature] From O1/BF1

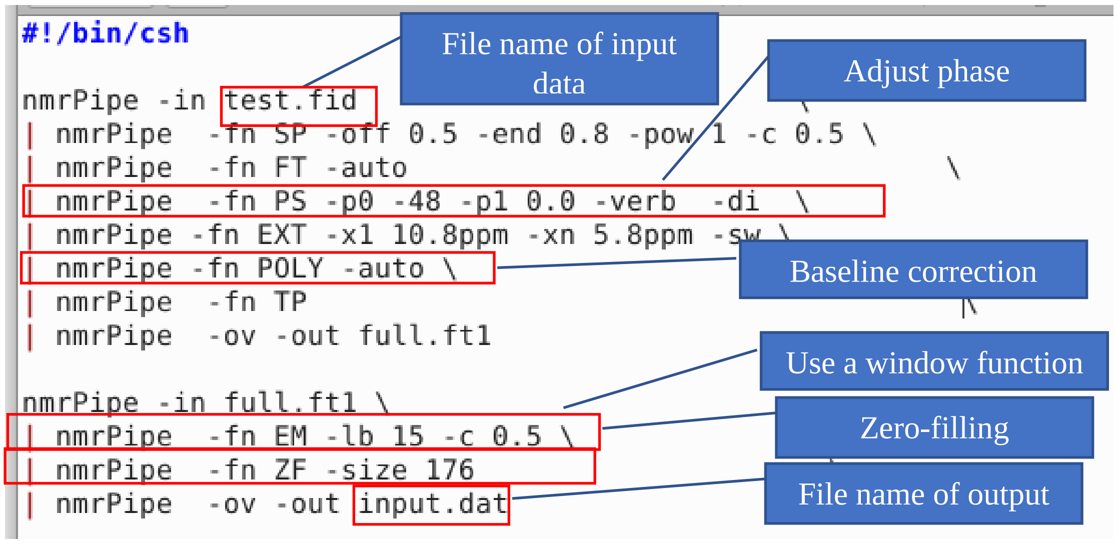
Run NMRDraw in Process Mode After Conversion

	x-axis	y-axis
Total Points R+I:	2048	512
Valid Points:	1024	256
Acquisition Mode:	DQD	Echo-AntiEcho
Spectral Width Hz:	10416.667	2424.242
Observe Freq MHz:	800.134	201.228
Center Position PPM:	4.868	178.809
Axis Label:	1H	13C

Figure R1. Read NUS data by NMRPipe

2) Copy the NMRPipe_Code/2D/preprocess.com (available on GitHub) to NUS_data/2D/, and run preprocess.com to preprocess the NUS data. This includes

processing steps such as direct dimension Fourier transform, using a window function, zero filling, etc (Figure R2). The preprocessed data will be saved as “input.dat” in NMRPipe format.



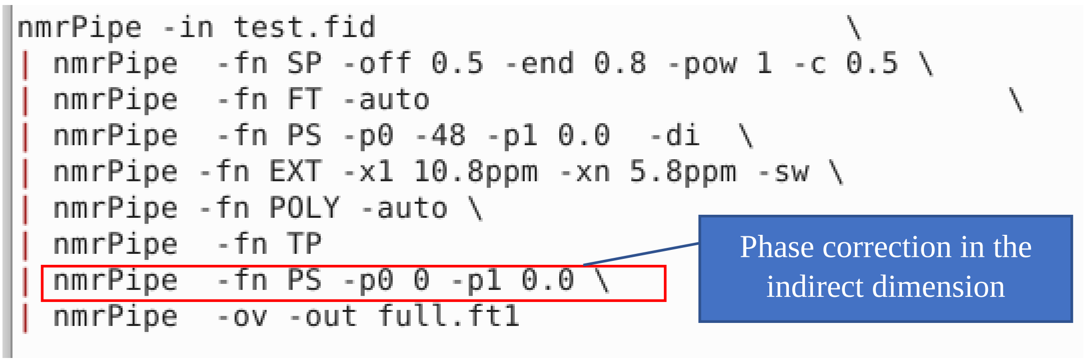
```
#!/bin/csh

nmrPipe -in test.fid
| nmrPipe  -fn SP -off 0.5 -end 0.8 -pow 1 -c 0.5 \
| nmrPipe  -fn FT -auto                                  \
| nmrPipe  -fn PS -p0 -48 -p1 0.0 -verb  -di   \
| nmrPipe -fn EXT -x1 10.8ppm -xn 5.8ppm -sw \
| nmrPipe -fn POLY -auto \
| nmrPipe  -fn TP
| nmrPipe  -ov -out full.ft1

nmrPipe -in full.ft1 \
| nmrPipe  -fn EM -lb 15 -c 0.5 \
| nmrPipe  -fn ZF -size 176
| nmrPipe  -ov -out input.dat
```

Figure R2. Part of NMRPipe_Code/preprocess.com

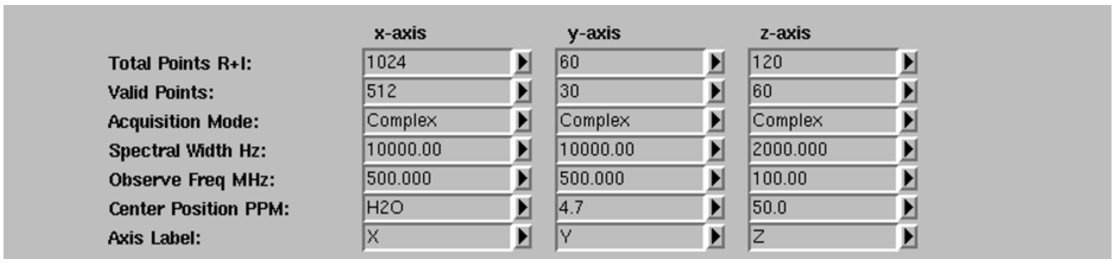
In addition, you can also perform phase correction in the indirect dimensions. For 2D NMR spectra, the NMRPipe_code/2D/preprocess.com can be used to perform phase correction in the indirect dimension. This script starts with preprocessing the direct dimension. After completing the preprocessing of the direct dimension, the phase of the indirect dimension can be adjusted using the code in the red rectangle in Figure R3



```
nmrPipe -in test.fid                             \
| nmrPipe  -fn SP -off 0.5 -end 0.8 -pow 1 -c 0.5 \
| nmrPipe  -fn FT -auto                           \
| nmrPipe  -fn PS -p0 -48 -p1 0.0  -di  \
| nmrPipe -fn EXT -x1 10.8ppm -xn 5.8ppm -sw \
| nmrPipe -fn POLY -auto \
| nmrPipe  -fn TP
| nmrPipe  -fn PS -p0 0 -p1 0.0 \
| nmrPipe  -ov -out full.ft1
```

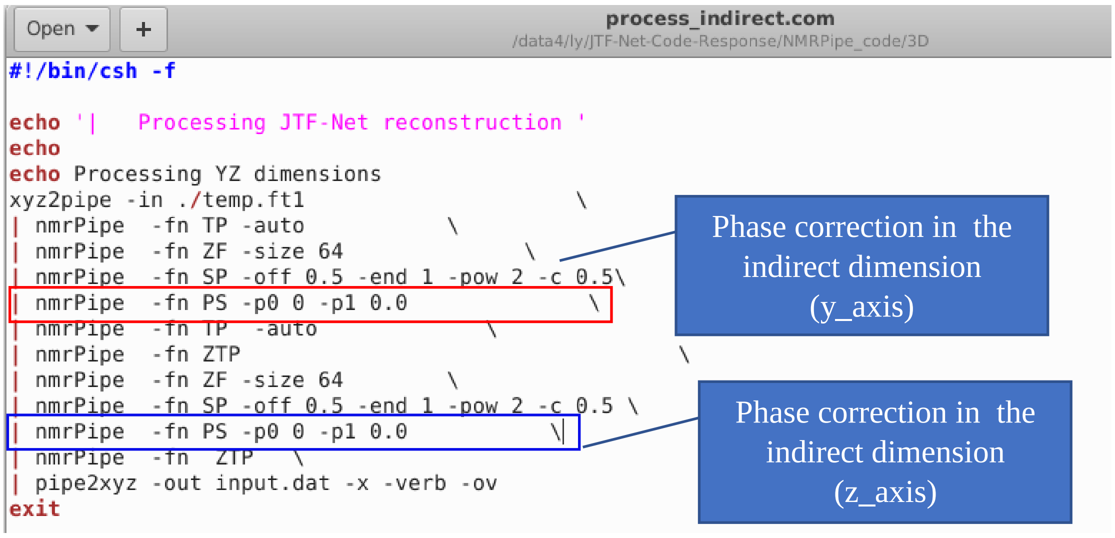
Figure R3. Part of NMRPipe_code/2D/preprocess.com.

For 3D spectra, the NMRPipe_code/3D/process_indirect.com can be used to perform phase correction in the indirect dimensions. When NMRPipe loads 3D NMR data, the two indirect dimensions are assigned to the y-axis and z-axis (Figure R4). The phase of the indirect dimension on the y-axis can be adjusted using the code in the red rectangle in Figure R5, while the phase of the indirect dimension on the z-axis is adjusted using the code in the blue rectangle in Figure R5.



	x-axis	y-axis	z-axis
Total Points R+I:	1024	60	120
Valid Points:	512	30	60
Acquisition Mode:	Complex	Complex	Complex
Spectral Width Hz:	10000.00	10000.00	2000.000
Observe Freq MHz:	500.000	500.000	100.00
Center Position PPM:	H2O	4.7	50.0
Axis Label:	X	Y	Z

Figure R4. Part of NMRPipe terminal when loading a NUS data.

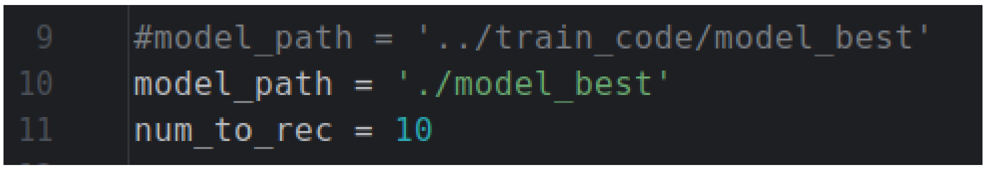

Open
+
process_indirect.com
/data4/ly/JTF-Net-Code-Response/NMRPipe_code/3D
Phase correction in the indirect dimension (y_axis)
Phase correction in the indirect dimension (z_axis)

```
#!/bin/csh -f

echo '|   Processing JTF-Net reconstruction '
echo
echo Processing YZ dimensions
xyz2pipe -in ./temp.ft1                         \
| nmrPipe  -fn TP -auto              \
| nmrPipe  -fn ZF -size 64                 \
| nmrPipe  -fn SP -off 0.5 -end 1 -pow 2 -c 0.5\
| nmrPipe  -fn PS -p0 0 -p1 0.0                     \
| nmrPipe  -fn TP  -auto                 \
| nmrPipe  -fn ZTP                                      \
| nmrPipe  -fn ZF -size 64           \
| nmrPipe  -fn SP -off 0.5 -end 1 -pow 2 -c 0.5 \
| nmrPipe  -fn PS -p0 0 -p1 0.0                \
| nmrPipe  -fn  ZTP    \
| pipe2xyz -out input.dat -x -verb -ov
exit
```

Figure R5. Part of NMRPipe_code/3D/process_indirect.com

1.1.2 Run Python_code/2D/test_code/test_JTF-Net.py and input the necessary details as instructed. Before running the script, note that on lines 9 and 10 (Figure R6), two model paths are provided: "./model_best" refers to the model we have trained, which you can download from the link provided in README in Github, while "../train_code/model_best" refers to the model trained by the reader (training details can be found in 1.2). Additionally, "num_to_rec" represents the number of times for reconstructing a single spectrum using JTF-Net (JTF-Net requires multiple reconstructions to get epistemic uncertainty), with the default set to 10.



```
 9    #model_path = '../train_code/model_best'
10    model_path = './model_best'
11    num_to_rec = 10
```

Figure R6. Part of test_JTF-Net.py

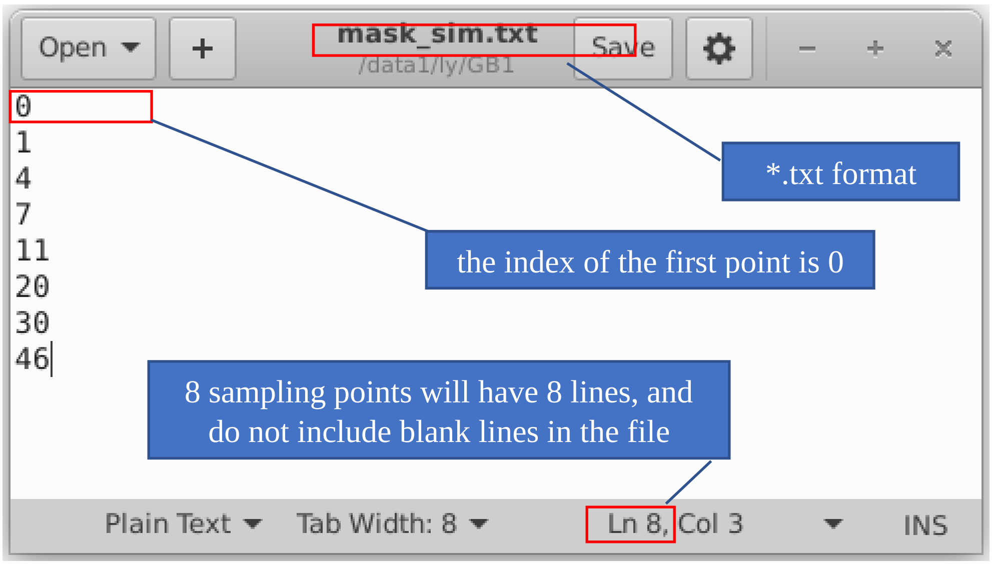
Run test_JTF-Net.py, and these instructions (in black italics) will be displayed in the program running windows:

Please input the size of the direct dimension: (The direct dimension size of input.dat)

Please input the size of the indirect dimension: (The indirect dimension size of input.dat)

Whether to change the SNR of the original spectrum (Y/N): (Since addition of artificial noise is required to change the SNR when validating REQUIRER in this paper, this option is provided.)

⋆If you input "N", it will prompt:	⋆If you input "Y" for additional artificial noise, it will prompt:
Please input the mask path: (You need to provide the path of the sampling scheme, which should be saved as a *.txt file, and ensure that the index of the first point is 0 and the file does not include blank lines, see Figure R7) *Please input the data path:* (Input the path of the "input.dat", which is generated by NMRPipe_Code/2D/preprocess.com in 1.1.1, and then press Enter to start the reconstruction.)	*Please input the noise factor:* (a float number) *Please input the mask path:* *Please input the data path:* The noise factor controls the noise level. Once you input the data path, the SNR will be displayed, and you can adjust the noise factor to change the SNR.


Open
mask_sim.txt
/data1/ly/GB1
Save
*.txt format
the index of the first point is 0
8 sampling points will have 8 lines, and do not include blank lines in the file
Plain Text
Tab Width: 8
Ln 8, Col 3
INS

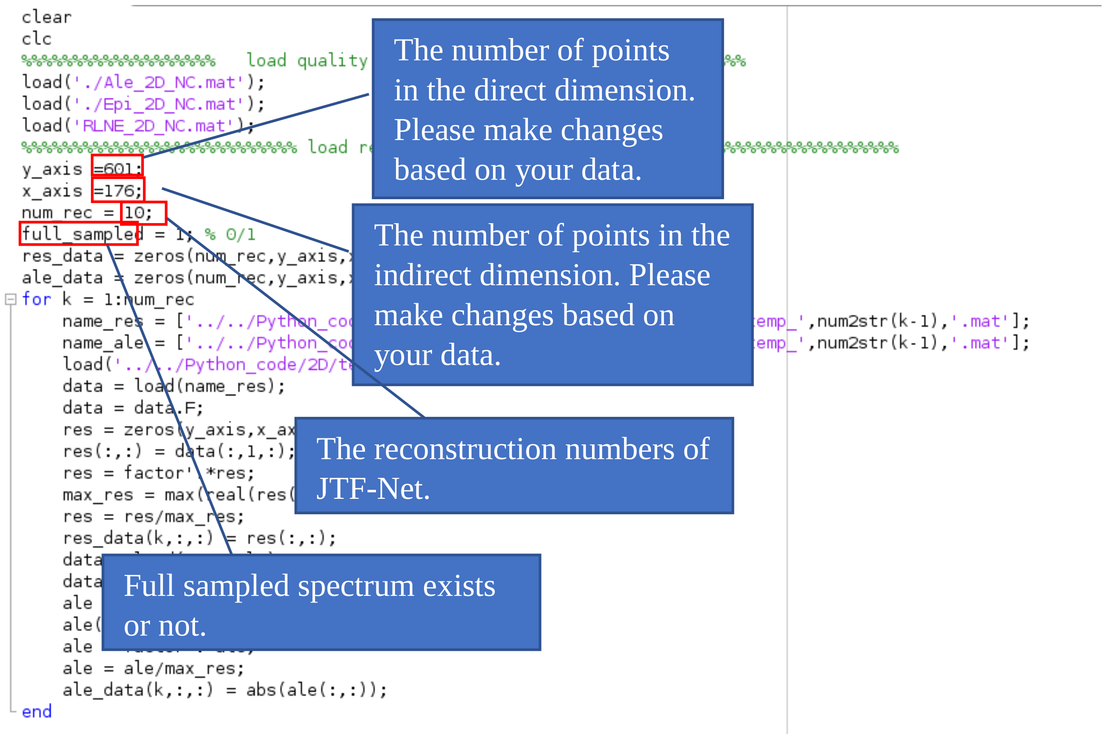
```
0
1
4
7
11
20
30
46
```

Figure R7. Details of the file of the sampling scheme

1.1.3 When the reconstruction is complete, the reconstructed spectra are automatically saved in the folder Python_code/2D/test_code/temp_data/rec_temp (this folder will be created automatically). If the "num_to_rec" is set to 10, the 10 reconstructed spectra and aleatoric uncertainties will be saved in Python_code/2D/test_code/rec_temp.

Before running Matlab_code/2D/prob_predic.m (Figure R8), set the "full_sampled" to 0 when only the NUS data is available. Then run prob_predic.m to get REQUIRER and the reconstructed spectrum. In this case, only the REQUIRER will be output.

If the full sampled spectrum is available, the actual RLNE indicator can be output. Before running Matlab_code/2D/prob_predic.m, it is necessary to preprocess the fully-sampled spectrum. The same script NMRPipe_code/2D/preprocess.com can be used to preprocess the fully sampled spectrum. The name of the output file should be set to "label.dat" and this file should be moved to the Pyhton_code/2D/test_code/label_path. Then, run the script Python_code/2D/test_code/trans_label_to_mat.py to convert the fully-sampled spectrum to the *.mat format, and save it in the Python_code/2D/test_code/label_path. Set the "full_sampled" variable to 1 and run Matlab_code/2D/prob_predic.m. In this case, the script will load the fully sampled spectrum in the Python_code/2D/test_code/label_path and output both the REQUIRER and the actual RLNE.


The number of points in the direct dimension. Please make changes based on your data.
The number of points in the indirect dimension. Please make changes based on your data.
The reconstruction numbers of JTF-Net.
Full sampled spectrum exists or not.

```
clear
clc
%%%%%%%%%%%%%%%%%%%%   load quality                                %%
load('./Ale_2D_NC.mat');
load('./Epi_2D_NC.mat');
load('RLNE_2D_NC.mat');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% load re                                %%%%%%%%%%%%%%%%%%
y_axis =601;
x_axis =176;
num_rec = 10;
full_sampled = 1; % 0/1
res_data = zeros(num_rec,y_axis,
ale_data = zeros(num_rec,y_axis,
for k = 1:num_rec
    name_res = ['../../Python_cod                                temp_',num2str(k-1),'.mat'];
    name_ale = ['../../Python_cod                                temp_',num2str(k-1),'.mat'];
    load('../../Python_code/2D/t
    data = load(name_res);
    data = data.F;
    res = zeros(y_axis,x_ax
    res(:,:) = data(:,1,:);
    res = factor'.*res;
    max_res = max(real(res(
    res = res/max_res;
    res_data(k,:,:) = res(:,:);
    data
    data
    ale
    ale(
    ale
    ale = ale/max_res;
    ale_data(k,:,:) = abs(ale(:,:));
end
```

Figure R8. Part of Matlab_code/2D/prob_predic.m

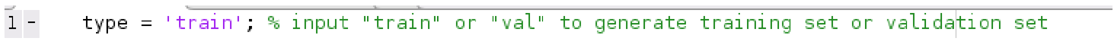
1.2 Network training (optional)

If you choose to use the trained model we provide, you can skip the step. If you choose to train the model yourself, you can choose the dataset we provide, or you can choose to use the code in Matlab_code to generate the datasets.

1.2.1 Generate training and validation datasets

Matlab_code/2D/main_pos.m (available on GitHub) can be used to generate training and validation datasets. Before running it, set the “type” in the first line of main_pos.m to “train” to generate the training dataset. Then set the type to “val” to generate the validation dataset (Figure R9). The training and validation datasets will be saved in Dataset/2D/ (This folder will be automatically generated when running main_pos.m for the first time, and there is no need to create it manually).

Or, you can download the training and validation datasets form the link in Github README.



```
1 -    type = 'train'; % input "train" or "val" to generate training set or validation set
```

Figure R9. The part of main_pos.m

1.2.2 Train JTF-Net

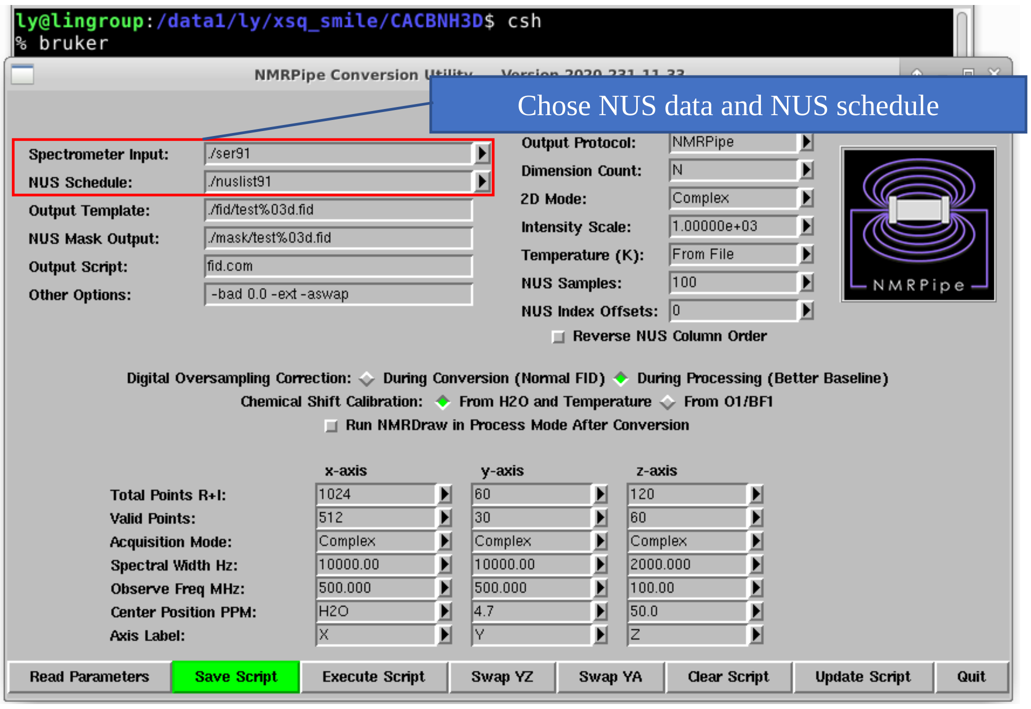
Run Python_code/2D/train_code/run.py to train JTF-Net. The training process is based on an early stopping strategy, which automatically stops when the validation loss does not improve for 10 epochs. The model with the lowest validation loss will be saved in the Python_code/2D/train_code/model_best folder.

2. 3D NMR spectra

2.1 Test

2.1.1. Preprocess

Firstly, copy the NUS data and its sampling scheme to the NUS_data/3D/. Use NMRPipe to automatically read NUS data. Enter “varian” or “bruker” (depending on the data type) in the NMRPipe terminal, as shown in Figure R10. Select the NUS data and corresponding undersampling scheme, and click “Read Parameters” and “Save Script” to automatically generate a script named fid.com. After running fid.com, you can obtain NUS data with unsampled points replaced by 0, which will be saved in NMRPipe format. If the data is in Bruker format, it will be saved at the path ./fid/test%03d.fid. If the data is in Varian format, it will be saved at the path ./data/test%03d.fid.



```
ly@lingroup:/data1/ly/xsq_smile/CACBNH3D$ csh
% bruker
```

Spectrometer Input: ./ser91
NUS Schedule: ./nuslist91
Output Template: ./fid/test%03d.fid
NUS Mask Output: ./mask/test%03d.fid
Output Script: fid.com
Other Options: -bad 0.0 -ext -aswap

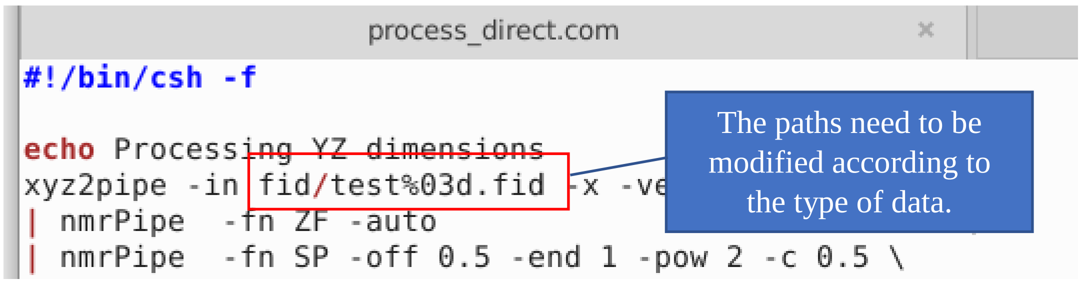
Output Protocol: NMRPipe
Dimension Count: N
2D Mode: Complex
Intensity Scale: 1.00000e+03
Temperature (K): From File
NUS Samples: 100
NUS Index Offsets: 0
Reverse NUS Column Order

Digital Oversampling Correction: During Conversion (Normal FID) [✓ During Processing (Better Baseline)]
Chemical Shift Calibration: [✓ From H2O and Temperature] From O1/BF1
Run NMRDraw in Process Mode After Conversion

	x-axis	y-axis	z-axis
Total Points R+I:	1024	60	120
Valid Points:	512	30	60
Acquisition Mode:	Complex	Complex	Complex
Spectral Width Hz:	10000.00	10000.00	2000.000
Observe Freq MHz:	500.000	500.000	100.00
Center Position PPM:	H2O	4.7	50.0
Axis Label:	X	Y	Z

Figure R10. Generate an NMRPipe script for NUS data.

2.1.2 Copy the 3D/NMRPipe_Code/process_direct.com and process_indirect.com (available on GitHub) to NUS_data/3D/, set the path in process_direct.com (Figure R11) to fid/test%03d.fid for Varian data (or data/test%03d.fid for Bruker data), and run process_direct.com and process_indirect.com in turn to preprocess the NUS data. The preprocessed data is saved as “input.dat”.



process_direct.com
The paths need to be modified according to the type of data.

```
#!/bin/csh -f

echo Processing YZ dimensions
xyz2pipe -in fid/test%03d.fid -x -ve
| nmrPipe  -fn ZF -auto
| nmrPipe  -fn SP -off 0.5 -end 1 -pow 2 -c 0.5 \
```

Figure R11. Part of process_direct.com

2.1.3 Before run Python_code/3D/test_code/test_JTF-Net.py. Like 1.1.2, two model paths are provided: "./model_best" refers to the model we have trained, which you can download from the link provided in README, while "../train_code/model_best" refers to the model trained by the reader (training details can be found in 2.2).

Run Python_code/3D/test_code/test_JTF-Net.py, and these instructions (in black italics) will be displayed in the program running windows:

Please input the size of the indirect dimension (z-axis):

Please input the size of the indirect dimension (y-axis):

Please input the size of the direct dimension:

Whether to change the SNR of the original spectrum (Y/N): (Since the addition of artificial noise is required to change the SNR when validating REQUIRER in this paper, this option is provided.)

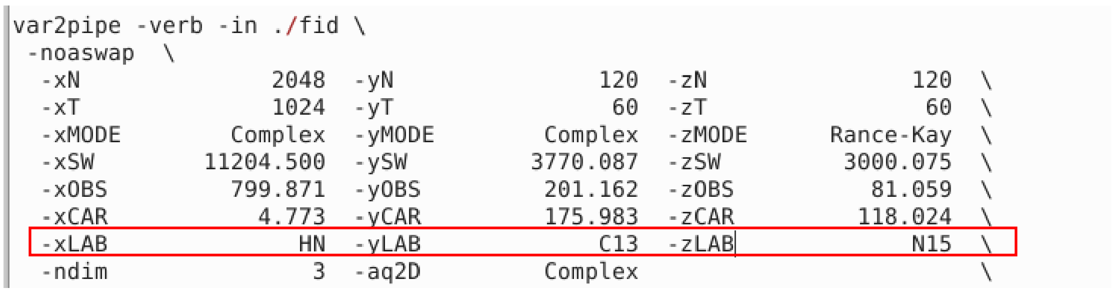
⁎If you input "N", it will prompt:	⁎If you input "Y", it will prompt:
Please input the mask path: (You need to provide the path to the sampling scheme, which should be saved as a *.txt file, and ensure that the index of the first sampling point is 0.) *Please input the data path:* (Input the path of the "input.dat", which is generated by NMRPipe_Code/3D/ process_indirect.com, and then press Enter to start the reconstruction.)	*Please input the noise factor:* (a float number) *Please input the mask path:* *Please input the data path:* The noise factor controls the noise level. Once you input the data path, the SNR will be displayed, and you can adjust the noise factor to change the SNR.

When the reconstruction is complete, the reconstructed spectra are automatically saved in the folder Python_code/3D/test_code/rec_data (this folder will be created automatically). If the "num_to_rec" is set to 10, the 10 reconstructed spectra and aleatoric uncertainties will be saved in Python_code/3D/test_code/rec_data.

2.1.4. Projections are required to observe the 3D spectrum when the reconstruction is completed. NMRPipe can generate projections on different planes for different types of spectra. For example, in the case of an HNCO spectrum, it can be seen from fid.com (Figure R12) that the three axes of this 3D spectrum are HN, C13, and N15. When the spectrum is projected using NMRPipe, three projection plane files are generated, including HN.C13.dat, HN.N15.dat, and C13.N15.dat, as shown in Figure R13.

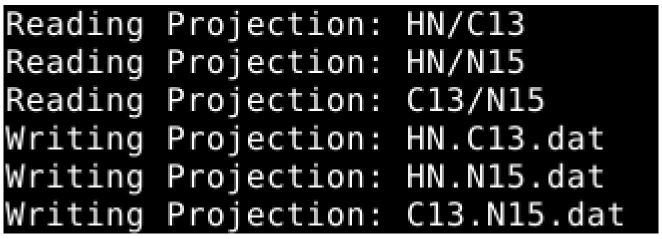
The script NMRPipe_code/3D/recFT.com is used to post-process and project the reconstructed spectra saved in Python_code/3D/test_code/rec_data. Since the indirect dimensions of these spectra are in the time domain, the script first applies operations like zero-filling, Fourier transforms, and windows functions before performing the projections. Before running recFT.com, ensure that the projection plane names are correctly set in the script according to your observation needs (Figure R14).

Run recFT.com, and the processed reconstructed spectra will be saved in the NMRPipe_code/3D/res_nmrpipe/ directory, named res3D1.dat, res3D2.dat, ..., res3Dn.dat, and the corresponding projections of these spectra will be saved as res_proj1.dat, res_proj2.dat, ..., res_projn.dat.



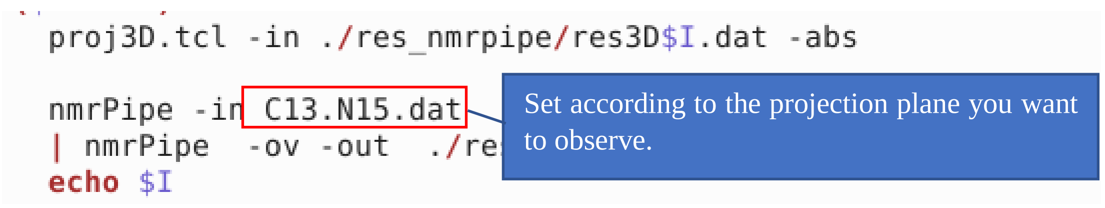
```
var2pipe -verb -in ./fid \
 -noaswap  \
  -xN              2048  -yN              120  -zN              120  \
  -xT              1024  -yT               60  -zT               60  \
  -xMODE        Complex  -yMODE       Complex  -zMODE      Rance-Kay  \
  -xSW        11204.500  -ySW        3770.087  -zSW        3000.075  \
  -xOBS         799.871  -yOBS        201.162  -zOBS         81.059  \
  -xCAR           4.773  -yCAR        175.983  -zCAR        118.024  \
  -xLAB              HN  -yLAB            C13  -zLAB            N15  \
  -ndim               3  -aq2D        Complex                        \
```

Figure R12. Part of a fid.com



```
Reading Projection: HN/C13
Reading Projection: HN/N15
Reading Projection: C13/N15
Writing Projection: HN.C13.dat
Writing Projection: HN.N15.dat
Writing Projection: C13.N15.dat
```

Figure R13. Information output by NMRPipe during projection



```
proj3D.tcl -in ./res_nmrpipe/res3D$I.dat -abs

nmrPipe -in C13.N15.dat
| nmrPipe  -ov -out  ./re
echo $I
```

Figure R14. Part of NMRPipe_code/3D/recFT.com

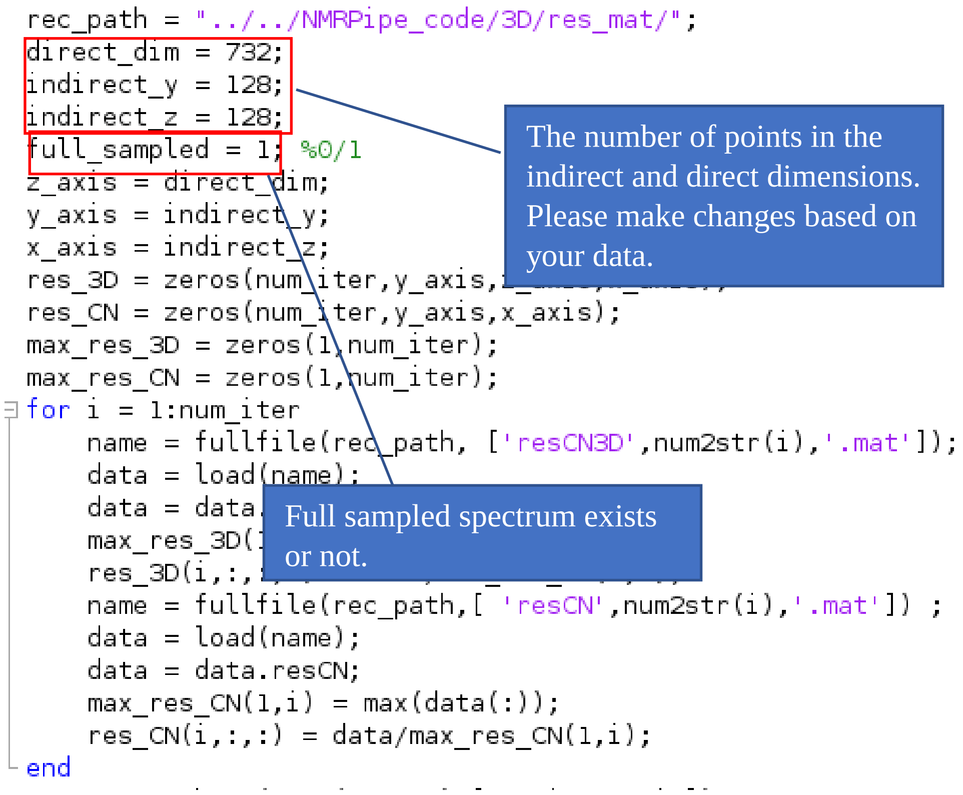
2.1.5 If the full sampled spectrum is available, the actual RLNE indicator can be output. It is necessary to preprocess the fully-sampled spectrum. Copy the fully sampled data to NMRPipe_code/3D/. The same script NMRPipe_code/3D/process_direct.com and process_indirect.com can be used to preprocess the fully sampled spectrum. As recFT.com is used for batch processing of multiple reconstruction outcomes, the script is provided for the post-processing and projection of the fully sampled spectrum. Run labelFT.com and the post-processed fully sampled spectrum and projection file are saved in the NMRPipe_code/3D/res_nmrpipe, named full_spec.ft3.dat and label_proj.dat, respectively.

Run Python_code/3D/test_code/nmrpipe_to_mat.py (available on GitHub) to convert the reconstructed spectra and their projection files in NMRPipe_code/3D/res_nmrpipe/ to *.mat for REQUIRER calculation in MATLAB. The script provides the option to convert the full sampled spectrum and its projection (if available) into *.mat. These instructions will be displayed in the program running windows:

Convert the full sampled spectrum to mat or not? (Y/N)

⁎If you input “Y”	⁎If you input “N”
It will prompt: *Please input the path of the projection file:* *Please input the path of the full sampled 3D spectrum:* These *.mat files will be saved in NMRPipe_code/3D/res_mat/.	It will only convert the reconstructed spectra and their projection files to *.mat. These *.mat files will be saved in NMRPipe_code/3D/res_mat/.

2.1.6 Run Matlab_code/3D/prob_predic.m (Figure R15). Set the “full_sampled” to 0 when only the NUS data is available. In this case, only the REQUIRER will be output. Set the full_sampled variable to 1 when you have the fully sampled spectrum. In this case, the script will output both the REQUIRER and the actual RLNE.


The number of points in the indirect and direct dimensions. Please make changes based on your data.
Full sampled spectrum exists or not.

```
rec_path = "../../NMRPipe_code/3D/res_mat/";
direct_dim = 732;
indirect_y = 128;
indirect_z = 128;
full_sampled = 1; %0/1
z_axis = direct_dim;
y_axis = indirect_y;
x_axis = indirect_z;
res_3D = zeros(num_iter,y_axis,
res_CN = zeros(num_iter,y_axis,x_axis);
max_res_3D = zeros(1,num_iter);
max_res_CN = zeros(1,num_iter);
for i = 1:num_iter
    name = fullfile(rec_path, ['resCN3D',num2str(i),'.mat']);
    data = load(name);
    data = data.
    max_res_3D(
    res_3D(i,:,
    name = fullfile(rec_path,[ 'resCN',num2str(i),'.mat']) ;
    data = load(name);
    data = data.resCN;
    max_res_CN(1,i) = max(data(:));
    res_CN(i,:,:) = data/max_res_CN(1,i);
end
```

Figure R15. Part of Matlab_code/3D/prob_predic.m

2.2 Network training (optional)

If you choose to use the trained model we provide, you can skip the step. If you choose to train the model yourself, you can choose the dataset we provide, or you can choose to use the code in Matlab_code to generate the datasets.

2.2.1 Generate the training and validation datasets

Matlab_code/3D/startGeneratingSamples.m (available on GitHub) can be used to generate training and validation datasets. The generated training and validation set will be saved in Dataset/3D/ (This folder will be automatically generated when running for the first time, and there is no need to create it manually).

Or, you can download the training and validation datasets form the link in Github README.

2.2.2 Train JTF-Net

Run Python_code/3D/train_code/run.py to train JTF-Net. Like 1.1.2, the training is also based on the early stopping strategy. The model with the lowest validation loss also be saved in the Python_code/3D/train_code/model_best folder.