Q.One Instrument

Quantum Nuclear Magnetic Resonance Spectrometer

Script Programming

Q.One Instrument

2021.05.15

1. Introduction..... 1
2. Quick Start.....2
3. Editor Description..... 3
4. Functions..... 7
4.1 SpinSPJ.....7
4.1.1 System Functions..... 8
4.1.2 Experiment Control.....9
4.1.3 Data Acquisition.....13
4.1.4 Data Processing and Analysis.....14
4.1.5 Data Acquisition And Setting.....17
4.2 Native Python Libraries.....20
4.2.1 NumPy.....20
4.2.2 Matplotlib.....21
5. Basic Application..... 22
5.1 Instrument Control..... 22
5.2 Automated Experiment.....22
5.3 Baseline Correction.....23
6. Advanced Application..... 24
6.1 Deep Learning.....24
6.2 Optimization Algorithm..... 26
6.3 PCA.....27
7. spinspj Command List.....30

1. Introduction

Scripting system is an important tool to extend the functionalities of NMR software, which allows users to write custom programs with a more flexible style. Scripts are written in a certain programming language, conventional scripting language are: JavaScript, VBScript, SQL, Python, etc. Python is one of the most popular modern computer programming language. Since the 1990s, Python has achieved rapid development due to its simplicity, legibility and extendibility. There are many different implementations for Python, such as Jython, PyPy, CPython, etc. Compared to other Python implementations, CPython has advantage due to its well-established ecosystem and most widely used. All subsequent references to Python in this manual refer to CPython.

SpinStudioJ provides a user friendly programming environment based on the Python scripting language for users. Users can not only call Python's conventional algorithms and machine learning libraries such as ***NumPy***, ***Pandas***, ***scikit-learn***, ***Tensorflow***, custom and commonly used NMR instrument control and data processing methods are also available for users. For basic Python syntax and related knowledge, please visit the following website:

（1） https://docs.python.org/3/tutorial/index.html

（2） https://docs.python.org/3/

Python editor of SpinStudioJ provides a simple and friendly interface for users. You can write Python scripts through the editor and achieve more effective numerical computation and function expansion of SpinStudioJ (the NMR software of Q.One Instrument).

The scripting system can offer the following functionalities:

（1） Instrument control

（2） Data processing

（3） Data analysis

(4) Call the Python native library

2. Quick Start

Click ***Tools>Python Editor*** on the main interface of SpinStudioJ to enter the Python editor interface. The scripting editor interface is shown in Figure 2.1.

Figure 2.1 Scripting editor interface

In the scripting editor, users can write conventional Python scripts as well as call instrument control and data processing functions by importing module “spinspj”. The Figure 2.2 shows the import of “spinspj” module.

Figure 2.2 Edit script code

After writing the Python scripts, there are two ways to run the script:

1) Click the button  to run the script.

2) The steps are shown in the following Figure 2.3.

   Step1: Save script file to path ***system\data\pyexample*** under the installation directory of NMR software .

   Step 2: Enter the corresponding script file name (suffix.py) in the ***NMR Console*** to call the corresponding script.

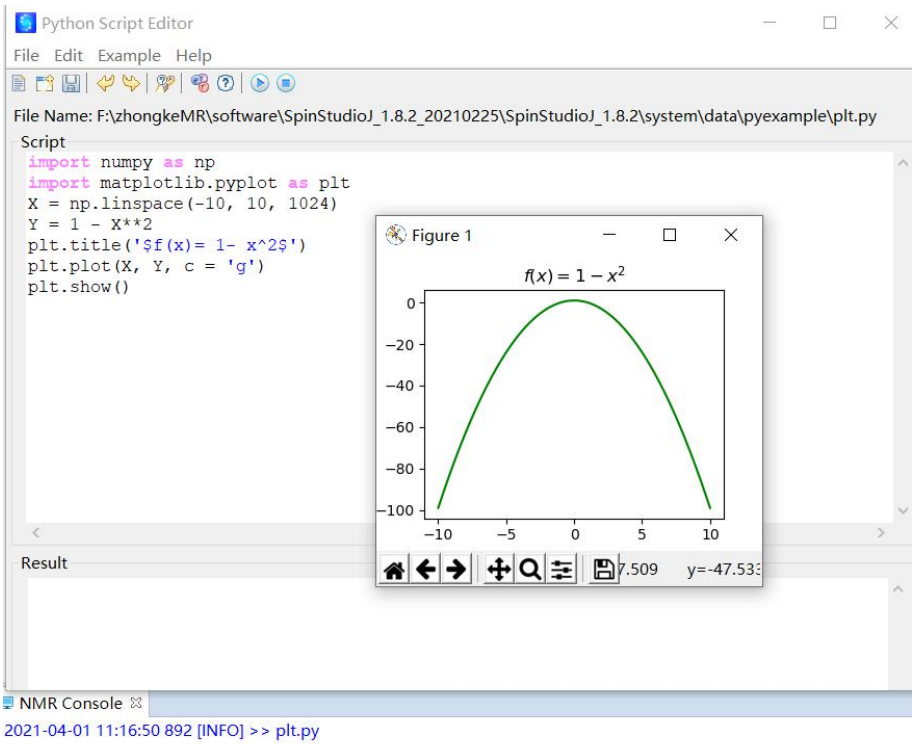

Figure 2.3 An example to execute the scripts

3. Editor Description

The Python editor is divided into 5 parts, namely: menu bar, toolbar, file path, script editing area, and result display area. The main interface of the scripting editor is shown in Figure 3.1.

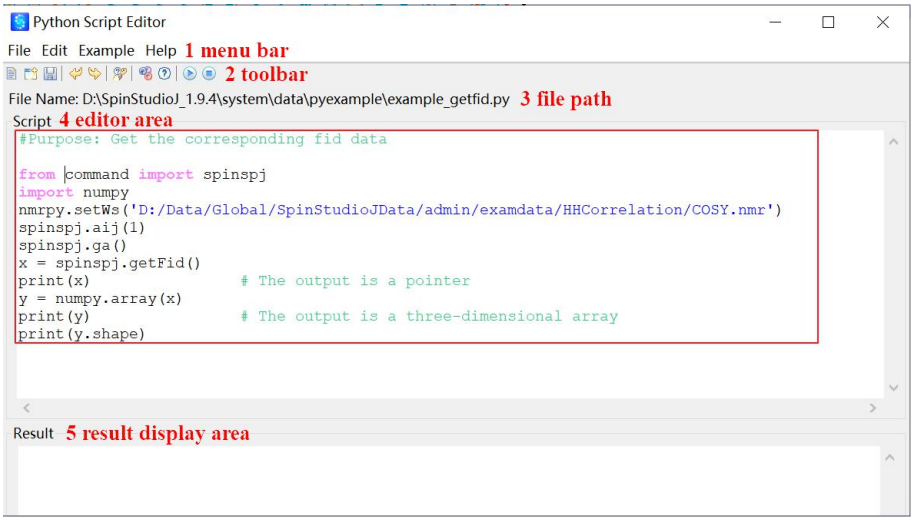

Figure 3.1 The main interface of the scripting editor

1) Menu Bar

There are 4 main menus on the menu bar: ***File***, ***Edit***, ***Example***, ***Help***。

1.1) File

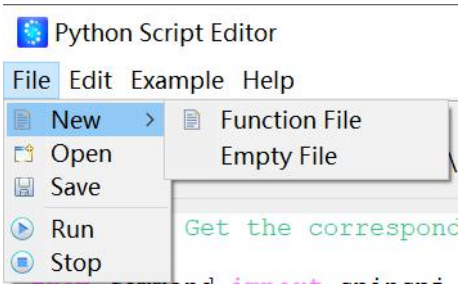

Figure 3.2 Items in menu File

The items contains in the ***File*** is shown in Figure 3.2.

New: (1)Select ***Empty File*** to create a new empty file. (2)Select ***Function File*** to create a new File for the Function definition.

Open: Open an existing script file with a ***.py*** suffix.

Save: Save the current script to a file. If there is no corresponding file in the disk, a save dialog will pop up for the user to select the save path. If already has a corresponding file in the disk, the file will be saved directly.

Run: Run the current script.

Stop: Stop running the current script.

1.2) Edit

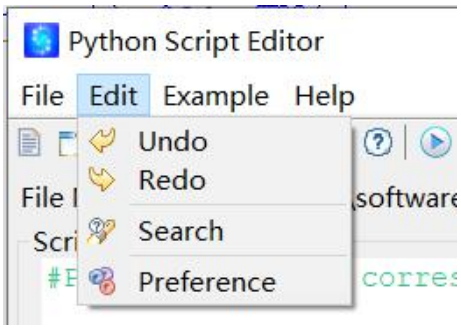

Figure 3.3 Items in menu Edit

The items contains in the ***Edit*** is shown in Figure 3.3.

Undo: Undo the previous operation.

Redo: Redo the previous operation.

Search: Open the Find dialog box to find or replace the script content.

The ***Search*** interface is shown in Figure 3.4.

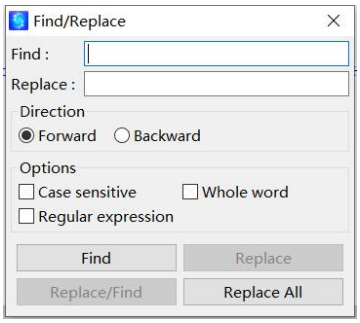

Figure 3.4 Find/Replace item in the menu

Preference: Opens the ***Preference*** dialog box. You can choose the Python Home path or the font color of the configuration script. If you change the Python path, you will need to restart the SpinStudioJ. The ***Preference*** interface is shown in Figure 3.5.

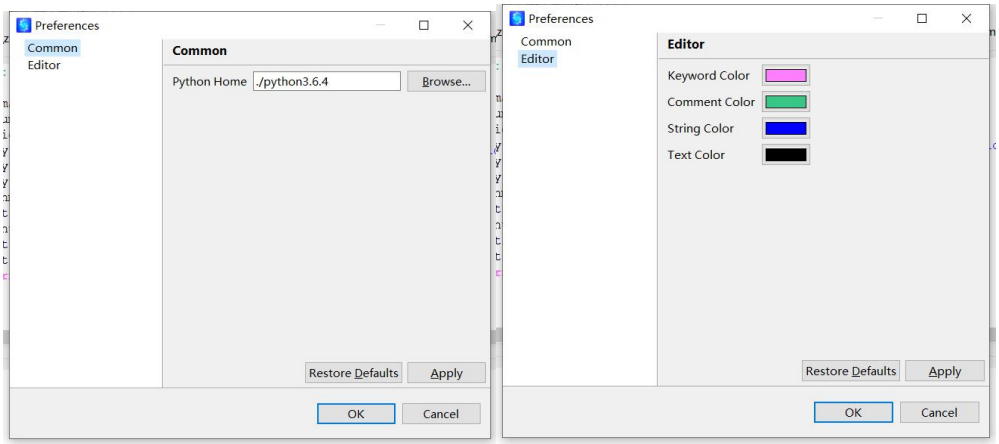

Figure 3.5 Preferences in the menu

1.3) Example

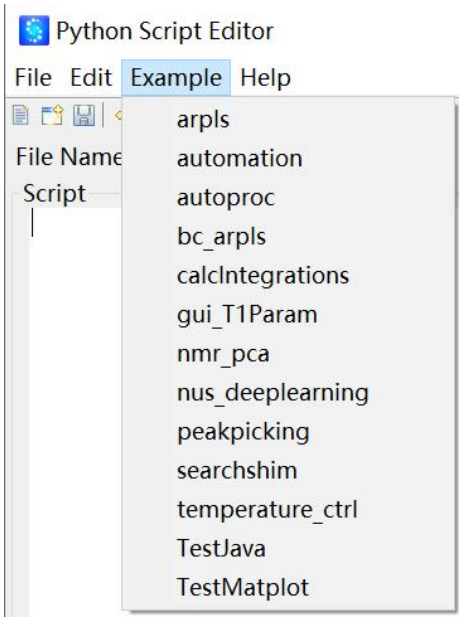

Figure 3.6 Examples in the menu

The examples in the Python editor of SpinStudioJ is shown in Figure 3.6. It mainly contains examples of Fourier transform, Hilbert transform and other operations on the spectrum by using Python editor.

1.4) Help

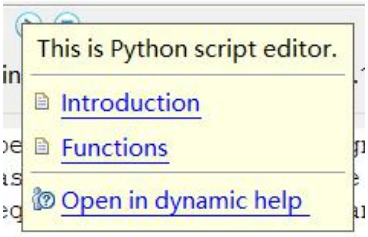

Figure 3.7 Help in the menu

The items contains in ***Help*** is shown in Figure 3.7.

Open ***Help*** to find the corresponding editor introduction document and function description document.

2) Toolbar

There are 10 buttons in the toolbar, corresponding to ***Function File***, ***Open***, ***Save***, ***Undo***, ***Redo***, ***Search***, ***Preferences***, ***Help***, ***Run***, ***Stop***. Move the mouse to a certain

button, a floating window will prompt the corresponding function name.

3) File Path

The ***File Path*** is used to display the local file path corresponding to the current script.

4) Editor Area

In this area, users can write and modify python scripts.

5) Result Display Area

Use to display the result of a script run or an error message.

4. Functions

Python is a dynamic, object-oriented scripting language, which contains a large number of functions to help users achieve various requirements. The Python commands in SpinStudioJ are divided into two categories:

1. ***SpinSPJ*** module, including custom NMR instrument control and data processing methods;
2. Native Python libraries, such as ***NumPy***, ***Matplotlib***, etc.

This manual focuses on the self-developed ***SpinSPJ*** module, and lists the usages and parameter descriptions of each command in detail. Example will be given for a few complex commands. The Python native library is abundant, and the manual provides a brief introduction to the ***NumPy*** and ***Matplotlib***.

4.1 SpinSPJ

The ***SpinSPJ*** module contains common commands for instrument control, NMR data analysis and NMR data processing.

Note: 1. Before writing the script, you need to import the ***SpinSPJ*** module, otherwise the command will not take effect. The import of ***SpinSPJ*** module and the commons contain in ***SpinSPJ*** are shown in Figure 4.1.

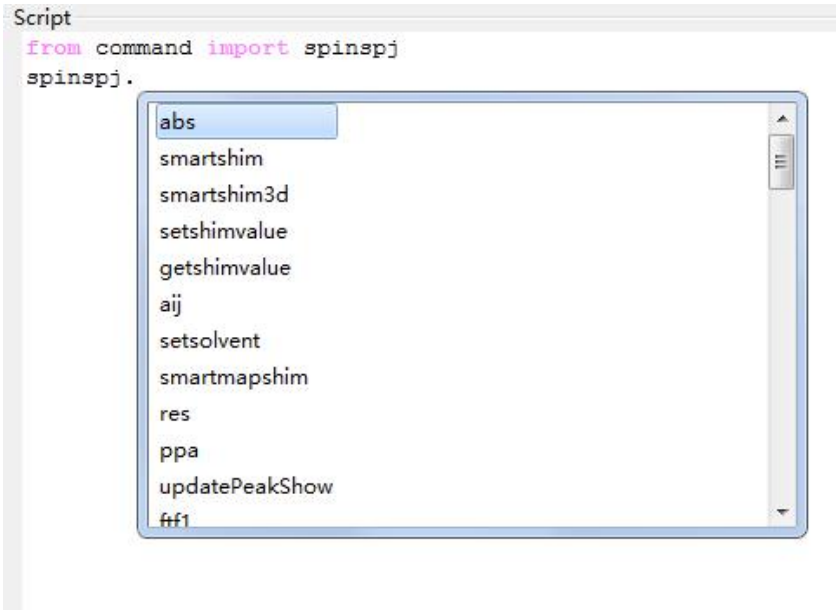

Figure 4.1 sub menu for prompting

2. When transforming the spectrum or FID, use the ***updateSpecShow*** or ***updateFidShow*** command to update display of the spectrum or FID. An example for transforming and updating spectrum is shown in Figure 4.2.

```
Script
#purpose: Acquire, and transform the obtained NMR data

from command import spinspj
spinspj.go('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.setWs('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.wft()
spinspj.aph()
spinspj.abs()
spinspj.updateSpecShow()#Update the display of spectrum data
```

Figure 4.2 An example for transforming and updating spectrum

4.1.1 System Functions

1) openWs()

Function: Open the file in the specified path.

Usage: spinspj.openWs(String filePath).

Parameter description:

filepath: file path.

2) setActiveWs()

Function: Set the current active workspace as the Python workspace.

Usage: spinspj.setActiveWs().

Parameter description: None.

3) setWs()

Function: Set the specified workspace as the Python workspace.

Usage: spinspj.setWs(String filePath).

Parameter description:

filepath: file path.

4) updateFidShow()

Function: Update the FID display.

Usage: spinspj. updateFidShow(boolean... checkProcessState).

Parameter description:

The default value is False, which means the status of FID display is not checked.

5) updateSpecShow()

Function: Update the spectrum display.

Usage: spinspj. updateSpecShow(boolean... checkProcessState).

Parameter description:

The default value is False, which means the status of spectrum display is not checked.

4.1.2 Experiment Control

4.1.2.1 Inject

1) aij()

Function: Automatic injection.

Usage: spinspj.aij(n, time).

Parameter description:

n: sample number;

time: maximum waiting time(unit :s).

```
Script
from command import spinspj
spinspj.setWs('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.aij(1, 60)   #Inject the sample 1, and the maximum waiting time is 60s
```

Figure 4.3 An example for injecting the sample

4.1.2.2 Temperature Control

1) vartemp()

Function: Temperature control.

Usage: spinspj.vartemp(targetTemp, time).

Parameter description:

targetTemp: target temperature (unit: ℃);

time: maximum waiting time(unit :s).

4.1.2.3 Spin

1) spin()

Function: Spin the sample by a specified speed.

Usage: spinspj.spin(target, time).

Parameter description:

target: speed (unit: Hz);

time: maximum waiting time (unit: s).

4.1.2.4 Lock

1) alock()

Function: Automatic lock.

Usage: spinspj.alock(time).

Parameter description:

time: maximum waiting time(unit: s).

4.1.2.5 Tuning

1) stm()

Function: Smart tuning and matching.

Usage: spinspj.stm('str', Boolean, time).

Parameter description:

str: nucleus;

Boolean: True or False, whether to ^{1}H-^{19}F decouple;

Time: maximum waiting time(unit: s).

An example for smart tuning and matching is shown in Figure 4.4.

```
Script
from command import spinspj
spinspj.setWs('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.stm('F19',True,55) #F19: nucleus, True: F19 tuning in broadband, used when doing H1{F19} decoupling experiment, 55: Maximum waiting time
```

Figure 4.4 An example for smart tuning and matching

2) stm4wks()

Function: Smart tuning and matching for the designated workspace.

Usage: spinspj.stm4wks(path, time)

Parameter description:

path: file path;

time: maximum waiting time(unit: s).

4.1.2.6 Shimming

1) simplexshim()

Function: Search shimming with simplex.

Usage: spinspj. simplexshim("str1", "str2", iteration, time).

Parameter description:

str1: evaluation standard;

str2: shim coil;

iteration: number of iteration;

time: maximum waiting time (unit: s).

An example for searching shimming with simplex algorithm is shown in Figure 4.5.

```
Script
from command import spinspj
spinspj.setWs('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.simplexshim('FIDArea','z1_z2_Z3',15,1000)
```

Figure 4.5 An example for searching shimming with simplex algorithm

2) smartmapshim()

Function: One-dimensional gradient field map + shimming.

Usage: spinspj.smartmapshim(time).

Parameter description:

time: maximum waiting time (unit: s).

3) smartmapshim3d()

Function: Three-dimensional smart field map + shimming.

Usage: spinspj.smartmapshim3d("str", nx, ny, time).

Parameter description:

str: nucleus;

nx: x phase value;

ny: y phase value;

time: maximum waiting time (unit: s).

4) smartshim()

Function: One-dimensional gradient shimming.

Usage: spinspj.smartshim(time).

Parameter description:

time : maximum waiting time (unit: s).

5) smartshim3d()

Function: Three-dimensional smart shim.

Usage: spinspj.smartshim3d('str',time).

Parameter description:

str: nucleus;

time: maximum waiting time (unit: s).

6) tuningshim()

Function: Search shimming with tuning.

Usage: spinspj.tuningshim('str1', 'str2', 'iteration', 'str3', time).

Parameter description:

str1: evaluation standard;

str2: shim coil;

iteration: number of iteration;

str3: step;

time: maximum waiting time (unit: s).

An example for searching shimming with tuning algorithm is shown Figure 4.6.

```
Script
from command import spinspj
spinspj.setWs('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.tuningshim('FIDArea','z1_z2_Z1','4_4_4','120_300_60',1000)
```

Figure 4.6 An example for searching shimming with tuning algorithm

4.1.3 Data Acquisition

1) ga()

Function: Acquisition and perform Fourier transform.

Usage: spinspj.ga(), sampling and Fourier transform in the current workspace.

spinspj.ga(path), sampling and Fourier transform in the specified workspace.

Parameter description:

path: file path.

2) go()

Function: Sample.

Usage: spinspj.go(), sampling in the current workspace.

spinspj.go(path), sampling in the specified workspace.

Parameter description:

path: file path.

4.1.4 Data Processing and Analysis

4.1.4.1 1D Data Processing and Analysis

1) abs()

Function: Automatic baseline correction.

Usage: spinspj.abs().

Parameter description: None.

2) dc()

Function: Zero-order polynomial baseline correction (full spectrum).

Usage: spinspj.dc().

Parameter description: None.

3) aph()

Function: Automatic phase correction.

Usage: spinspj.aph().

Parameter description: None.

4) aref()

Function: Automatic calibration.

Usage: spinspj.aref().

Parameter description: None.

An example for automatic reference is shown in Figure 4.7.

```
Script
from command import spinspj
spinspj.setActiveWs()
spinspj.aref()
spinspj.updateSpecShow()
```

Figure 4.7 An example for automatic reference

5) dsnmax()

Function: Calculate the maximum signal-to-noise ratio.

Usage: spinspj.dsnmax(Float... noiseWidth).

Parameter description: The default value is 200Hz.

6) ft()

Function: Fourier transform.

Usage: spinspj.ft ().

Parameter description: None.

7) ftabs()

Function: Fourier transform and display as absolute value spectrum.

Usage: spinspj.ftabs().

Parameter description: None.

8) res()

Function: Calculate the resolution of a one-dimensional spectrum.

Usage: print(spinspj.res()).

Parameter description: None.

9) wft()

Function: Windowed Fourier transform.

Usage: spinspj.wft ().

Parameter description: None.

10) wftabs()

Function: Windowed Fourier transform and display in absolute value spectrum.

Usage: spinspj.wftabs().

Parameter description: None.

11) wftpwr()

Function: Windowed Fourier transform and display in power spectrum.

Usage: spinspj.wftpwr()

Parameter description: None.

4.1.4.2 2D Data Processing and Analysis

1) ftf2()

Function: Fourier transform for F2 dimension.

Usage: spinspj.ftf2().

Parameter description: None.

2) ftf3()

Function: Fourier transform for F3 dimension.

Usage: spinspj.ftf3().

Parameter description: None.

3) ftpwr()

Function: Fourier transform and display in power spectrum.

Usage: spinspj.ftpwr().

Parameter description: None.

4) ftf1()

Function: Fourier transform for F1 dimension.

Usage: spinspj.ftf1()

Parameter description: None.

5) wftf1()

Function: Windowed Fourier transform for F1 dimension.

Usage: spinspj.wftf1().

Parameter description: None.

6) wftf2()

Function: Windowed Fourier transform for F2 dimension.

Usage: spinspj.wftf2().

Parameter description: None.

7) wftf3()

Function: Windowed Fourier transform for F3 dimension.

Usage: spinspj.wftf3().

Parameter description: None.

8) tilt()

Function: Spectrum tilt.

Usage: spinspj.tilt().

Parameter description: None.

9) sym()

Function: Symmetrical COSY spectrum.

Usage: spinspj.sym().

Parameter description: None.

10) symj()

Function: Symmetrical J spectrum.

Usage: spinspj.symj().

Parameter description: None.

11) ppa()

Function: Find peaks in the full spectrum.

Usage: spinspj.ppa().

Parameter description: None.

4.1.5 Data Acquisition And Setting

1) getFid()

Function: Get the FID data of the specified path.

Usage: spinspj.getFid(String filePath).

Parameter description:

filepath: file path.

An example for getting FID data is shown in Figure 4.8.

Script

```
#Purpose: Get the corresponding fid data

from command import spinspj
import numpy
spinspj.setWs('D:/Data/Global/SpinStudioJData/admin/1.nmr')
spinspj.aij(1)
spinspj.ga()
x = spinspj.getFid()
print(x)                # The output is a pointer
y = numpy.array(x)
print(y)                # The output is a three-dimensional array
print(y.shape)
```

Figure 4.8 An example for getting FID data

2) getPara()

Function: Get parameter value.

Usage: spinspj.getPara(String paraName).

Parameter description:

paraName: parameter name.

3) getshimvalue()

Function: Get the value of the shim coil.

Usage: spinspj.getshimvalue('str').

Parameter description:

str: shim coil.

4) getSpec()

Function: Get the spectrum data of the specified path.

Usage: spinspj. getSpec(String filePath).

Parameter description:

filepath: file path.

5) setFid()

Function: Set the FID data of the specified path.

Usage: spinspj.setFid(String filePath, ArrayList<?> realData, ArrayList<?> imagData).

Parameter description:

filepath: file path;

realData: real data of FID,;

imagData: imaginary data of FID.

An example for setting and getting FID data is shown in Figure 4.9.

```
Script
#Get fid data

print('begin read fid')
result = spinspj.getFid()
array =  numpy.asarray(result)
fid = array[:,0,:]+1j*array[:,1,:]
print(fid.ndim)
print(fid.shape)

#set fid data

spinspj.setWs('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.setFid(fid.real.tolist(),fid.imag.tolist())
```

Figure 4.9 An example for setting and getting FID data

6) setPara()

Function: Set parameter value.

Usage: spinspj.setPara(String paraName, Object paraValue).

Parameter description:

p*araName*: parameter name;

paraValue: parameter value.

7) setshimvalue()

Function: Set the shim value.

Usage: spinspj.setshimvalue('str1', shimvalue).

Parameter description:

str: shim coil;

shimvalue: the current value of the shim coil.

8) setSpec()

Function: Set spectrum data of the specified path.

Usage: spinspj.setSpec(String filePath, ArrayList<?> realData, ArrayList<?> imagData).

Parameter description:

filepath: file path;

realData: real data of spectrum;

imagData: imaginary data of spectrum.

An example for setting and getting spectrum data is shown in Figure 4.10.

```
Script
#Get spectrum data

print('begin read fid')
result = spinspj.getSpec()
array =  numpy.asarray(result)
fid = array[:,0,:]+1j*array[:,1,:]
print(spec.ndim)
print(spec.shape)

#set spectrum data

spinspj.setWs('D:\Data\Global\SpinStudioJData\admin\1.nmr')
spinspj.setSpec(spec.real.tolist(),spec.imag.tolist())
```

Figure 4.10 An example for setting and getting spectrum data

4.2 Native Python Libraries

The native library of Python is rich and powerful. Users can import Python libraries through the ***import*** command as needed for further operations. Here's a quick look at two libraries commonly used in Python.

4.2.1 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. The NMR data is large in volume and high in dimension, and the ***NumPy*** library can be used for data calculation and data conversion efficiently.

The example in Figure 4.11 shows how to convert FID data to an array using the ***array*** method from the ***NumPy*** library and output the result. ***NumPy*** creates an array

with a shape property that returns the number of dimensions of each dimension. The output results show that the FID data is a 3D array composed of 512 2D arrays with 2 rows and 1024 columns.

Script

```
#Purpose: Get the corresponding fid data

from command import spinspj
import numpy
spinspj.setWs('D:/Data/Global/SpinStudioJData/XL/examdata/HHCorrelation/COSY.nmr')
data = spinspj.getFid()
#print(data)                    # The output is a pointer
result = numpy.array(data)
print(result)                   # The output is a three-dimensional array
print(result.shape)
```

Result

```
...

[[  0.   0.   0. ...  845. -1810. -1294.]
 [  0.   0.   0. ... 1647.  1417.  -826.]]

[[  0.   0.   0. ... 1209.  1187. -1992.]
 [  0.   0.   0. ... -1024.  1619.  1608.]]

[[  0.   0.   0. ...  437.  1414.   953.]
 [  0.   0.   0. ... -813.  -945.  1685.]]]
(512, 2, 1024)
```

Figure 4.11 An example for library Numpy

4.2.2 Matplotlib

Matplotlib is a plotting library for Python, which can be used with ***NumPy*** to draw scatter plots, histograms, line graphs, etc.

Users can read the data file and draw as needed. The Figure 4.12 shows an example for library Matplotlib.

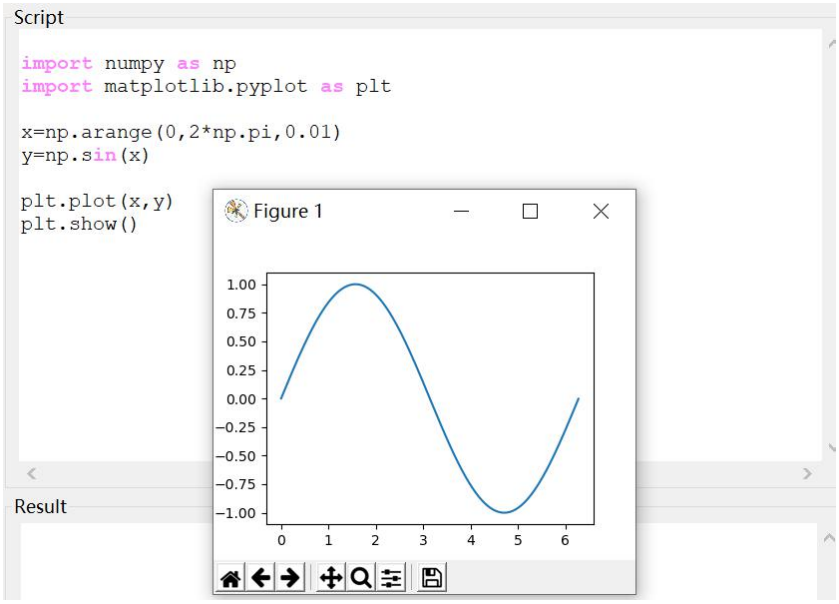

Figure 4.12 An example for library Matplotlib

5. Basic Application

5.1 Instrument Control

Instrument control is an indispensable part of NMR experiment and Python provides convenience for users to achieve temperature control, sample spin, inject and so on. An example of temperature control is shown in Figure 5.1.

```
Script
from command import spinspj

spinspj.satActiveWs()  #set active workspace
spinspj.vartem(30)  #the target temperature is 30℃
spinspj.go()  #acquisition
spinspj.updateFidShow() #update the display of spectrum
spinspj.stopvartem() #stop the temperature control
```

Figure 5.1 An example for library temperature control

5.2 Automated Experiment

Python in SpinStudioJ provides great convenience for experimentation. Users can write Python scripts according to requirements to achieve inject, control instrument temperature, automatic data processing and so on. At the same time, users

can save the corresponding script file for the next call to the script without rewriting the scripts. An example of automatic experiment is shown in following Figure 5.2.

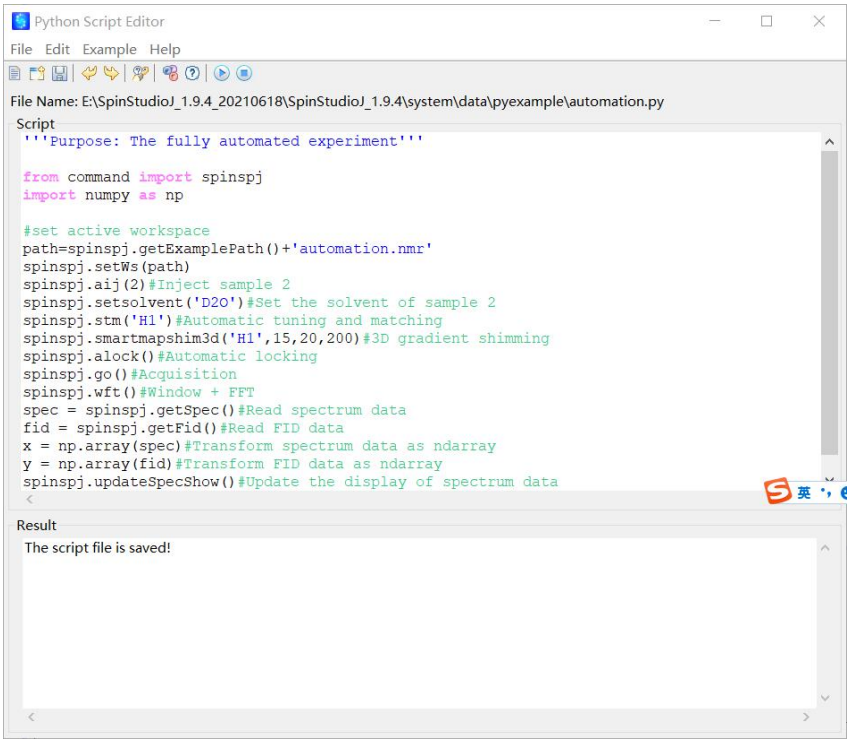

Figure 5.2 An example for automated experiment

5.3 Baseline Correction

Baseline correction is a common spectrum pretreatment method and an important data processing method for NMR experiments. With Python, users can use common baseline correction methods for data processing to obtain a high quality spectrum. The following Figure 5.3 shows an example of baseline correction with arPLS.

```
def arpls(data,lamda,ratio):
    N = data.shape[0]
    D = speyediff(N)
    H = D.T.dot(D) * lamda
    w = np.ones(N, dtype=float)
    while True:
        col = np.arange(N)
        row = np.arange(N)
        W = csc_matrix((w, (row, col)), shape=(N, N))
        sol = W + H
        z = splu(sol).solve(w * data)
        d = data - z
        dn = d[d < 0]
        m = np.mean(dn)
        s = np.std(dn)
        wt = 1 / (1 + np.exp((2 * (d - (2 * s - m)) / s)))
        if np.linalg.norm(w - wt) / np.linalg.norm(w) < ratio: break
        w = wt
    return z
lamda=pow(10,10)
ratio=0.01
BaseLine=arpls(Re,lamda,ratio)
```

Figure 5.3 The script for baseline correction

The output is shown in Figure 5.4. The red line is the baseline and the blue is the actual spectrum.

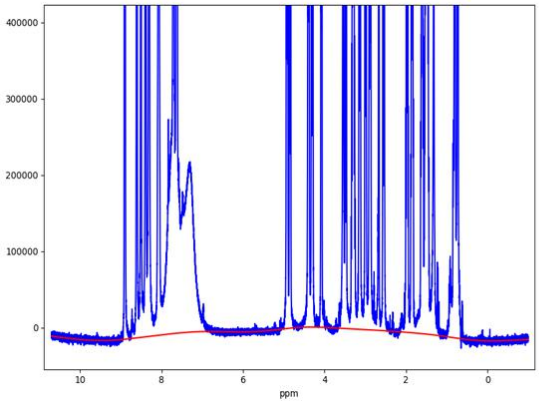

Figure 5.4 An example for baseline correction

6. Advanced Application

6.1 Deep Learning

Deep learning is a type of machine learning that trains a computer to perform human-like tasks, such as recognizing speech, identifying images or making predictions. Instead of organizing data to run through predefined equations, deep learning sets up basic parameters about the data and trains the computer to learn on its

own by recognizing patterns using many layers of processing.

In SpinStudioJ, deep learning is applied to non-uniform sampling (NUS), which greatly reduces the sampling time. Reconstructing a spectrum from NUS data is equivalent to mapping of the input under sampled FID signal to the target spectrum. First, the spectrum artifacts introduced by NUS are removed with dense convolutional neural network (CNN, a kind of deep learning) and then the reconstructed spectra are further refined to maintain the data consistency to the sampled signal.

As illustrated in following Figure 6.1, a 3D HNCO NMR data of Azurin (molecular weight is 14kDa) with full sampling was downloaded from the MddNMR website http://mddnmr.spektrino.com. A fair comparison between fully and 20% sampling in respect to the spectral quality are shown in (a)-(f). In Figure 6, (a) and (c) are the sub-regions of the projections on the planes of ^{15}N-^{1}H and ^{15}N-^{13}C for the fully sampled 3D spectrum, which is reconstructed by fast Fourier Transform. (b) and (d) are the corresponding reconstruction results of a 20% sampling rate, which is reconstructed by a DLNMR method. (e) gives the correlation coefficient of the peak intensities of (a) and (b), (f) gives the correlation coefficient of the peak intensities of (c) and (d). The correlation coefficients of peak intensities between DLNMR reconstructed and fully sampled spectra are greater than 0.99, which indicates excellent fidelity of the two spectra. For the 3D NMR, the achieved acceleration factor of 5 in NUS implies that the experimental time can be reduced from 22.4 hours to 4.48 hours. In addition, the computation time for the 3D reconstruction is 9.66 seconds (data size: 732*60*60, GPU: Nvida Tesla K40M), which demonstrates the method achieves very high computational efficiency. This example shows the capability of the scripting system to implement a deep learning based NMR method by leveraging native CPython libraries.

```
#Import libraries including deep learning
import numpy as np
from r import DLNMR,reconstruction
from keras.models import Model
from command import spinspj
.....
#Load the model of deep learning
    DL_NMR = DLNMR(
        model_kind="HyperComplex",
        weight_path=weight_path,
        GPU_index="0",
        verbosity=2,)
.....
#Set the related parameters
    dimension = 3  #dimension
    targetFidPt = (366, 120, 120) #FID size
    fnMode = 4 #acquisition mode of indirect
                #dimension
#Reconstruct the spectrum
    DL_NMR_rec = reconstruction(
        DL_NMR, data_nus,  dimension,
        targetFidPt, acquiredPtF1, fnMode,
        number_split=32, verbosity=2,)
```

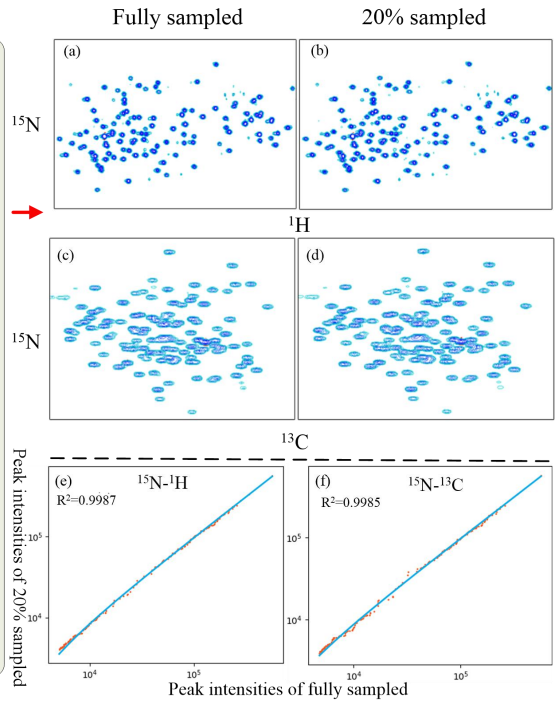

Figure 6.1 An example for NUS by deep learning

6.2 Optimization Algorithm

In practical applications, finding the optimal solution is a common requirement, but when the amount of data is large or the problem model is very complex, solving the optimal solution of the problem requires a large amount of calculation. The optimization algorithm relies on the iterative idea to search for optimization, which improves the efficiency of solving optimization problems. In the NMR experiment process, optimization ideas can be used to find better shimming values, tuning parameters and etc. Users can call methods and write scripts according to requirement, and combine optimization algorithms to solve problems.

The ***scipy.optimize*** submodule for Python provides functions for minimizing (or maximizing) objective functions, possibly subject to constraints. It includes solvers for nonlinear problems (with support for both local and global optimization algorithms), linear programing, constrained and nonlinear least-squares, root finding, and curve fitting.

An example of searching shim is shown in Figure 6.2. Taking the area of FID as

the measurement standard, and the simplex method is used to search for the minimum value of 1/FIDArea to achieve the purpose of searching for better shimming.

Figure 6.2 The script for searching shimming by algorithm simplex

6.3 PCA

Principal Component Analysis (PCA) is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process. The main aim of PCA is to overcome the dimensionality of the problem. The reduction of dimensionality should be such that when dropping higher dimensions, the loss of data is minimum.

NMR experimental data can be pre-processed by PCA to retain high-latitude data, remove some noise and unimportant features, so as to achieve the purpose of increasing the data processing speed. An example of using PCA for NMR data preprocessing is shown in Figure 6.3.

Script

```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
origin_data = np.loadtxt('.\system\data\pyexample\ST000101_gone.txt')
data = np.array(origin_data)
data = np.transpose(data)
rownum = data.shape[0]
colnum = data.shape[1]
for k in range(rownum):
    m=np.mean(data[k,:])
    stddata = np.std(data[k,:])
    data[k,:] = (data[k,:] - m)/stddata
data2 = np.transpose(data)
pca1=PCA(n_components=8)
newData=pca1.fit(data2)
print(pca1.explained_variance_ratio_)
print(pca1.explained_variance_)
eig=pca1.explained_variance_
xcomp = [1, 2, 3, 4, 5, 6, 7, 8]
plt.figure(1)
plt.legend()
plt.bar(xcomp, pca1.explained_variance_ratio_[0:8], label='', color='blueviolet')
plt.xlabel('principal component')
plt.ylabel('R2X')
plt.title('principal component analysis')
plt.show()
```

Figure 6.3 The script for principle component analysis

As illustrated in the following figure, the data are from the web site "metabolomics workbench", and the corresponding study ID is "ST000101" [35] which presents an NMR analysis of synthetic mixtures. Each of total 10 samples has 20 synthetic metabolites. All of the samples can be divided into two groups because the quantity of the 10 metabolites is quite different. After a Java based automatic phase, baseline correction, and integration of binned spectra, the data set is analyzed by a PCA that is implemented with the proposed spinspj scripting system. The script for the PCA includes reading the original data, calculating the average and standard deviation, conducting a PCA, and displaying the columnar and scattered data. CPython libraries of NumPy, scikit-learn[36], and matplotlib are helpful to implement above requirements. The histogram shown in Figure 6.4 gives the result of the 1^{st}~8^{th} principal components and their contribution percentages. Among all of the components, the first principal component can explain 58.02% of the information in the total samples.

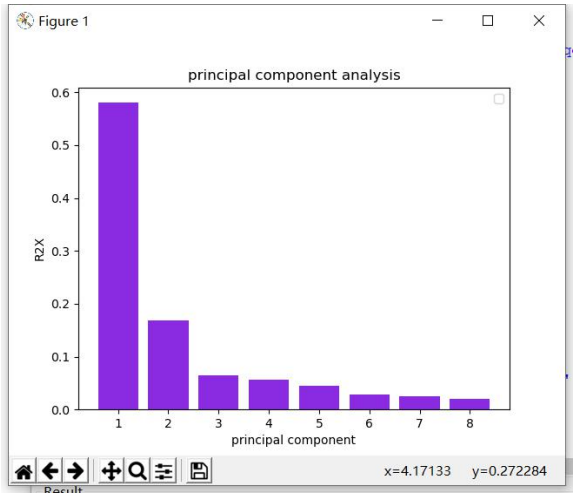

Figure 6.4 The histogram of principle component analysis

The score scatter plot in Figure 6.5 shows the score values of each sample on the first and second principal component. Obviously, on the first dimension of the PCA, the samples are divided into group A and B, which is consistent with the metabolite composition of the samples. The data of the score scatter plot is also the same with the published result of the study.

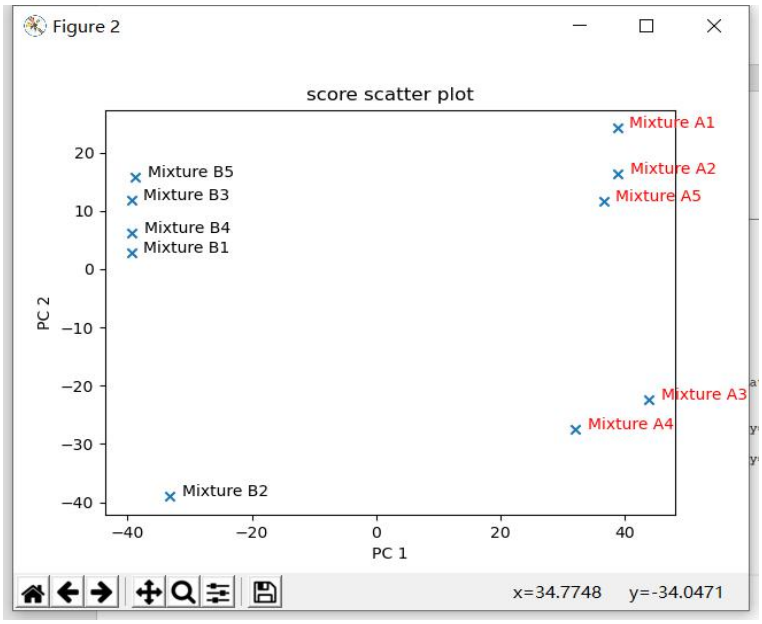

Figure 6.5 The score figure for principle component analysis

7. spinspj Command List

Python Function	*Description*
System Functions	
openWs(String filePath) *Example:* *from command import spinspj* *spinspj.openWs('D:/Data/Global /COSY.nmr')*	Open the file in the specified path. Parameter description: *filepath*: file path.
setWs(String filePath) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/ COSY.nmr')*	Set the specified workspace as the Python workspace. Parameter description: *filepath*: file path.
setActiveWs() *Example:* *from command import spinspj* *spinspj.setActiveWs()*	Set the current active workspace as the Python workspace. Parameter description: None.
updateFidShow(boolean... checkProcessState) *Example:* *spinspj.updateFidShow()*	Update the FID display. Parameter description: The default value is False, which means the status is not checked.
updateSpecShow(boolean... checkProcessState) *Example:*	Update the spectrum display. Parameter description: The default value is False, which

spinspj.updateSpecShow()	means the status is not checked.
Instrument Control	
Inject	
aij(n, time) *Example:* *from command import spinspj* *spinspj.aij(3, 40)*	Automatic injection. Parameter description: *n*: sample number; *time*: maximum waiting time(unit: s).
Temperature Control	
vartemp(targetTemp, time) *Example:* *from command import spinspj* *spinspj.vartemp(28, 40)*	Temperature control. Parameter description: *targetTemp*: target temperature; *time*: maximum waiting time(unit: s).
Spin	
spin(speed, time) *Example:* *from command import spinspj* *spinspj.spin(100, 60)*	Spin the sample by a specified speed. Parameter description: *speed*: speed(unit: Hz); *time*: maximum waiting time(unit: s).
Lock	
alock(time)	Automatic lock. Parameter description:

Example: *spinspj.alock(55)*	*time*: maximum waiting time(unit: s).
Tuning	
stm('str', Boolean, time) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/COSY.nmr')* *spinspj.stm(H1,True, 60)*	Automatic tuning. Parameter description: *str*: nucleus; *Boolean* : True or False, whether to decouple; *time*: maximum waiting time(unit: s).
stm4wks(path, time) *Example:* *from command import spinspj* *spinspj.stm4wks('D:/Data/Global/COSY.nmr',55)*	Automatically tune in the designated workspace. Parameter description: *path*: file path; *time*: maximum waiting time(unit: s).
Shimming	
simplexshim("str1", "str2", iteration, time) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/COSY.nmr')* *spinspj.simplexshim('FIDArea', 'z1_z2_z3', 15,1000)*	Search shimming with simplex. Parameter description: *str1*: evaluation standard; *str2*: shim coil; *iteration*: number of iteration; *time*: maximum waiting time(unit: s).
smartmapshim(time)	One-dimensional gradient field

Example: *from command import spinspj* *spinspj.setWs('D:/Data/Global/COSY.nmr')* *spinspj.smartmapshim(30)*	map + shimming. Parameter description: *time*: maximum waiting time(unit: s).
smartmapshim3d("str", nx, ny, time) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/COSY.nmr')* *spinspj.smartmapshim3d('H1', 30,50,100)*	Three-dimensional smart field map + shimming. Parameter description: *str*: nucleus; *nx*: x phase value; *ny*: y phase value; *time*: maximum waiting time(unit: s).
smartshim(time) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/COSY.nmr')* *spinspj.smartshim(30)*	One-dimensional gradient shimming. Parameter description: *time*: maximum waiting time(unit: s).
smartshim3d('str',time) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/ COSY.nmr')* *spinspj.smartshim3d(H1, 45)*	Three-dimensional smart shim. Parameter description: *str*: nucleus; *time*: maximum waiting time(unit: s).
tuningshim('str1', 'str2', 'iteration', 'str3', time)	Search shimming with tuning.

Example: *from command import spinspj* *spinspj.setWs('D:/Data/Global/COSY.nmr')* *spinspj.tuningshim3d('FIDArea','z1_z2_z3','4_4_4','120_120_120',1000)*	Parameter description: *str1*: evaluation standard; *str2*: shim coil; *iteration*: number of iteration; *str3*: step; *time*: maximum waiting time(unit: s).
Acquisition	
ga() *Example*: *from command import spinspj* *spinspj.ga('D:/Data/Global/COSY.nmr')*	Sample and perform Fourier transform. Parameter description: *path*: file path.
go() *Example*: *from command import spinspj* *spinspj.go('D:/Data/Global /COSY.nmr')*	Sample. Parameter description: *path*: file path.
Data Processing and Analysis	
1D Data Processing and Analysis	
abs() *Example*: *from command import spinspj* *spinspj.setWs('D:/Data/Global/C13CPD.nmr')* *spinspj.abs()* *spinspj.updateSpecShow()*	Automatic baseline correction. Parameter description: None.

dc() *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/C13CPD.nmr')* *spinspj.dc()* *spinspj.updateSpecShow()*	Zero-order polynomial baseline correction (full spectrum). Parameter description: None.
aph() *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/C13CPD.nmr')* *spinspj.aph()* *spinspj.updateSpecShow()*	Automatic phase correction. Parameter description: None.
aref() *Example:* *from command import spinspj* *spinspj.setActiveWs()* *spinspj.aref()* *spinspj.updateSpecShow()*	Automatic calibration. Parameter description: None.
dsnmax(Float... noiseWidth) *Example:* *from command import spinspj* *spinspj.setActiveWs()* *print(spinspj.dsnmax())*	Calculate the maximum signal-to-noise ratio. Parameter description: The default value is 200Hz.

res() *Example:* *from command import spinspj* *spinspj.setActive()* *print(spinspj.res())*	Calculate the resolution of a one-dimensional spectrum. Parameter description: None.
ft() *Example:* *from command import spinspj* *spinspj.setActiveWs()* *spinspj.ft()*	Fourier transform. Parameter description: None.
ftabs() *Example:* *from command import spinspj* *spinspj.setActiveWs()* *spinspj.ftabs()* *spinspj.updateSpecShow()*	Fourier transform and display as absolute value spectrum. Parameter description: None.
wft() *Example:* *from command import spinspj* *spinspj.setActiveWs()* *spinspj.wft()* *spinspj.updateSpecShow()*	Window function & Fourier transform. Parameter description: None.
wftabs()	Window function, Fourier

Example: *from command import spinspj* *spinspj.setActive()* *spinspj.wftabs()* *spinspj.updateSpecShow()*	transform and display in absolute value spectrum. Parameter description: None.
wftpwr() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.wftpwr()* *spinspj.updateSpecShow()*	Window function, Fourier transform and display in power spectrum. Parameter description: None.
2D Data Processing and Analysis	
ftf2() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.ftf2()* *spinspj.updateSpecShow()*	Fourier transform for F2 dimension. Parameter description: None.
ftf3() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.ftf3()*	Fourier transform for F3 dimension. Parameter description: None.

spinspj.updateSpecShow()	
ftpwr() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.ftpwr()* *spinspj.updateSpecShow()*	Fourier transform and display in power spectrum. Parameter description: None.
ftf1() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.ftf1()* *spinspj.updateSpecShow()*	Fourier transform for F1 dimension. Parameter description: None.
wftf1() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.wftf1()* *spinspj.updateSpecShow()*	Window function & Fourier transform for F1 dimension. Parameter description: None.
wftf2() *Example:* *from command import spinspj* *spinspj.setActive()*	Window function & Fourier transform for F2 dimension. Parameter description: None.

spinspj.wftf2() *spinspj.updateSpecShow()*	
wftf3() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.wftf3()* *spinspj.updateSpecShow()*	Window function & Fourier transform for F3 dimension. Parameter description: None.
ppa() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.ppa()* *spinspj.updateSpecShow()*	Auto peak picking on full spectrum. Parameter description: None.
tilt() *Example:* *from command import spinspj* *spinspj.setActive()* *spinspj.tilt()* *spinspj.updateSpecShow()*	Tilt automatically along the line. Parameter description: None.
sym() *Example:* *from command import spinspj*	Symmetry for COSY-like spectrum. Parameter description: None.

spinspj.setWs('D:\Data\Global\COSY.nmr') *spinspj.sym()* *spinspj.updateSpecShow()*	
symj() *Example:* *from command import spinspj* *spinspj.setActiveWs()* *spinspj.symj()* *spinspj.updateSpecShow()*	Symmetry for J spectrum. Parameter description: None.
Data Acquisition And Setting	
getFid(String filePath) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/ COSY.nmr')* *spinspj.aij(1)* *spinspj.ga()* *data : spinspj.getFid()* *print(data)* *result : spinspj.array(data)* *print(result)*	Get the FID data of the specified path. Parameter description: *filepath*: file path.
getPara(String paraName) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/COSY.nmr')* *result : spinspj.getPara(np1)*	Get parameter value. Parameter description: *paraName*: parameter name.

print(result)	
getshimvalue('str') *Example*: *from command import spinspj* *spinspj.setWs('D:/Data/Global/ COSY.nmr')* *result : spinspj.getshimvalue('z1')* *print(result)*	Get the value of the shim coil. Parameter description: *str*: shim coil.
getSpec(String filePath) *Example*: *from command import spinspj* *spinspj.setWs('D:/Data/Global/ COSY.nmr')* *print('begin read data')* *result : spinspj.getSpec()* *array : spinspj.asarray(result)* *spec : array[:,0,:] + 1j*array[:,1,:]* *print(spec.ndim)* *print(spec.shape)*	Get the spectrum data of the specified path. Parameter description: *filepath*: file path.
setFid(String filePath , ArrayList<?> realData , ArrayList<?> imagData) *Example*: *from command import spinspj* *spinspj.setWs('D:/Data/Global/test2.nmr')* *spinspj.setFid(fid.real.tolist(), fid.imag.tolist())*	Set the FID data of the specified path. Parameter description: *filepath*: file path; *realData*: real data of FID, *imagData*: imaginary data of FID.
setPara(String paraName, Object paraValue)	Set parameter value.

Example: *from command import spinspj* *spinspj.setWs('D:/Data/Global/test2.nmr')* *spinspj.setPara(np1,12)*	Parameter description: p*araName*: parameter name; *paraValue*: parameter value .
setshimvalue('str1', shimvalue) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/test2.nmr')* *spinspj.setshimvalue('z1',100)*	Set the shim value. Parameter description: *str*: shim coil; *shimvalue*: the current value of the shim coil.
setSpec(String filePath，ArrayList<?> realData，ArrayList<?> imagData) *Example:* *from command import spinspj* *spinspj.setWs('D:/Data/Global/test2.nmr')* *spinspj.setSpec(spec.real.tolist(), sepc.imag.tolist())*	Set spectrum data of the specified path. Parameter description: *filepath*: file path; *realData*: real data of spectrum; *imagData*: imaginary data of spectrum.