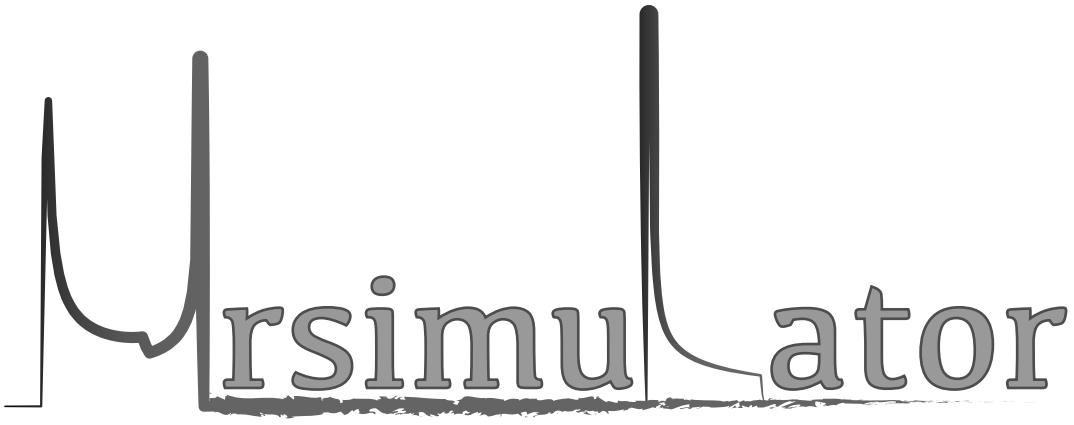


Mrsimulator Documentation

Release 0.6.1

The Mrsimulator Developers

Oct 10, 2021

GETTING STARTED

LIST OF FIGURES

LIST OF TABLES

LIST OF CODE BLOCKS

About

`mrsimulator` is an open-source python package for fast simulation and analysis of multi-dimensional solid-state magnetic resonance (NMR) spectra of crystalline and amorphous materials.

Why use mrsimulator?

- It is open-source and free.
- It is a fast and versatile multi-dimensional solid-state NMR spectra simulator including, MAS and VAS spectra of nuclei experiencing chemical shift (nuclear shielding) and quadrupolar coupling interactions.
- It includes simulation of weakly coupled nuclei experiencing J and dipolar couplings.
- It is fully documented with a stable and simple API and is easily incorporated into your python scripts and web apps.
- It is compatible with modern python packages, such as scikit-learn, Keras, etc.
- Packages using mrsimulator -
  - mrinversion

Features

The `mrsimulator` package offers the following

- **Fast simulation** of one/two-dimensional solid-state NMR spectra. See our *Performance benchmark* (page 271) results.
- **Simulation of coupled and uncoupled spin system**
  - for spin $I = \frac{1}{2}$, and quadrupole $I \geq \frac{1}{2}$ nuclei,
  - at arbitrary macroscopic magnetic flux density,
  - at arbitrary rotor angles, and
  - at arbitrary spinning frequency.
- **A library of NMR methods,**
  - 1D Bloch decay spectrum,
  - 1D Bloch decay central transition spectrum,
  - 2D Multi-quantum Variable Angle Spinning (MQ-VAS),
  - 2D Satellite-transition Variable Angle Spinning (ST-VAS),
  - 2D Dynamic Angle Spinning (DAS),
  - 2D isotropic/anisotropic sideband correlation spectrum (e.g. PASS and MAT),
  - 2D Magic Angle Flipping (MAF).
- **Models for tensor parameter distribution in amorphous materials.**
  - Czjzek
  - Extended Czjzek

Warning: The package is currently under development. We advice using with caution. Bug report are greatly appreciated.

Part I

Getting Started

CHAPTER

ONE

INSTALLATIONS

1.1 Package dependencies

`mrsimulator` depends on the following packages:

Required packages

- NumPy>=1.17
- openblas
- cython>=0.29.14
- typing-extensions>=3.7
- matplotlib>=3.3.3 for figures and visualization,
- monty>=2.0.4
- csdmpy>=0.4
- pydantic>=1.0
- monty>=2.0.4

Other packages

- pytest>=4.5.0 for unit tests.
- pre-commit for code formatting
- sphinx>=2.0 for generating the documentation
- sphinxjp.themes.basicstrap for documentation.
- breathe==4.26 for generating C documentation
- sphinx-copybutton

1.2 For the users

1.2.1 Strict Requirements

`mrsimulator` has the following strict requirements:

- Python 3.6 or later
- Numpy 1.17 or later

See *Package dependencies* (page 5) for a full list of requirements.

Make sure you have the required version of python by typing the following in the terminal,

---

Tip: You may also click the copy-button located at the top-right corner of the code cell area in the HTML docs, to copy the code lines without the prompts and then paste it as usual. Thanks to Sphinx-copybutton

---

```
$ python --version
```

For *Mac* users, python version 3 is installed under the name *python3*. You may replace *python* for *python3* in the above command and all subsequent python statements.

For *Windows* users, we recommend the Anaconda or miniconda distribution of python>3.6. Anaconda distribution for python comes with popular python packages that are frequently used in scientific computing. Miniconda is a minimal installer for conda. It is a smaller version of Anaconda that includes conda, Python, and the packages they depend on, along with other useful packages such as pip.

See also:

If you do not have python or have an older version of python, you may visit the Python downloads or Anaconda websites and follow their instructions on how to install python.

1.2.2 Installing `mrsimulator` using pip

On Google Colab Notebook

Colaboratory is a Google research project. It is a Jupyter notebook environment that runs entirely in the cloud. Launch a new notebook on Colab. To install the mrsimulator package, type

```
!pip install mrsimulator
```

in the first cell, and execute. All done! You may now start using the library, or proceed to section *Introduction to Spin Systems* (page 15) to continue the tutorial.

On Local machine (Using pip)

PIP is a package manager for Python packages and is included with python version 3.4 and higher. PIP is the easiest way to install python packages.

Linux

Mac OSX

Windows

For *Linux* users, we provide the binary distributions of the mrsimulator package for python versions 3.6-3.9. Install the package using pip as follows,

```
$ pip install mrsimulator
```

For *Mac* users, we provide the binary distributions of the mrsimulator package for python versions 3.6-3.9. Install the package using pip as follows,

```
$ pip install mrsimulator
```

If the above statement didn't work, you are probably using mac OS system python, in which case, use the following,

```
$ python3 -m pip install mrsimulator --user
```

Note: We currently do not provide binary distributions for windows. You'll need to compile and build the mrsimulator library from source. The following instructions are one-time installation only. If you are upgrading the package, see the *Upgrading to a newer version* (page 7) sub-section.

Install conda

Skip this step if you already have miniconda or anaconda for python>=3.6 installed on your system. Download the latest version of conda on your operating system from either miniconda or Anaconda websites. Make sure you download conda for python 3. Double click the downloaded .exe file and follow the installation steps.

OpenBLAS and FFTW libraries

Launch the `Anaconda prompt` (it should be located under the start menu). Within the anaconda prompt, type the following to install the package dependencies.

```
$ conda install -c conda-forge openblas fftw
```

Install a C/C++ compiler

Because the core of the mrsimulator package is written in C, you will require a C-compiler to build and install the package. Download and install the Microsoft Visual C++ compiler from Build Tools for Visual Studio 2019.

Install the package.

From within the `Anaconda Prompt`, build and install the mrsimulator package using pip.

```
$ pip install mrsimulator
```

If you get a `PermissionError`, it usually means that you do not have the required administrative access to install new packages to your Python installation. In this case, you may consider adding the `--user` option at the end of the statement to install the package into your home directory. You can read more about how to do this in the pip documentation.

Upgrading to a newer version

If you are upgrading to a newer version of `mrsimulator`, you have all the prerequisites installed on your system. In this case, type the following in the terminal/Prompt

```
$ pip install mrsimulator -U
```

All done! You may now start using the library, or proceed to section *Introduction to Spin Systems* (page 15) to continue the tutorial.

1.2.3 Building `mrsimulator` from the source

Prerequisites

You will need a C-compiler suite and the development headers for the BLAS and FFTW libraries, along with development headers from Python and Numpy, to build the `mrsimulator` library from source. The mrsimulator package utilizes the BLAS and FFTW routines for numerical computation. To leverage the best performance, we recommend installing the BLAS and FFTW libraries, which are optimized and tuned for your system. In the following, we list recommendations on how to install the c-compiler (if applicable), BLAS, FFTW, and building the mrsimulator libraries.

Obtaining the Source Packages

Stable packages

The latest stable source package for `mrsimulator` is available on PyPI and Github release. Download and extract the *.tar.gz* file.

OS-dependent prerequisites

Note: Installing OS-dependent prerequisites is a one-time process. If you are upgrading to a newer version of mrsimulator, skip to *Building and Installing* (page 9) section.

Linux

Mac OSX

Windows

OpenBLAS and FFTW libraries

On Linux, the package manager for your distribution is usually the easiest route to ensure you have the prerequisites to building the mrsimulator library. To build from source, you will need the OpenBLAS and FFTW development headers for your Linux distribution. Type the following command in the terminal, based on your Linux distribution.

For (Debian/Ubuntu):

```
$ sudo apt-get install libopenblas-dev libfftw3-dev
```

For (Fedora/RHEL):

```
$ sudo yum install openblas-devel fftw-devel
```

Install a C/C++ compiler

The C-compiler comes with your Linux distribution. No further action is required.

OpenBLAS/Accelerate and FFTW libraries

You will require the `brew` package manager to install the development headers for the OpenBLAS (if applicable) and FFTW libraries. Read more on installing brew from homebrew.

Step-1 Install the FFTW library using the homebrew formulae.

```
$ brew install fftw
```

Step-2 By default, the mrsimulator package links to the openblas library for BLAS operations. Mac users may opt to choose the in-build Apple's Accelerate library. If you opt for Apple's Accelerate library, skip to *Step-3*. If you wish to link the mrsimulator package to the OpenBLAS library, type the following in the terminal,

```
$ brew install openblas
```

Step-3 If you choose to link the mrsimulator package to the OpenBLAS library, skip to the next section, *Building and Installing* (page 9).

(a) You will need to install the BLAS development header for Apple's Accelerate library. The easiest way is to install the Xcode Command Line Tools. Note, this is a one-time installation. If you have previously installed the Xcode Command Line Tools, you may skip this sub-step. Type the following in the terminal,

```
$ xcode-select --install
```

(b) The next step is to let the mrsimulator setup know your preference. Open the `settings.py` file, located at the root level of the mrsimulator source code folder, in a text editor. You should see

```
# -*- coding: utf-8 -*-
# BLAS library
use_openblas = True
# mac-os only
use_accelerate = False
```

To link the mrsimulator package to the Apple's Accelerate library, change the fields to

```
# -*- coding: utf-8 -*-
# BLAS library
use_openblas = False
# mac-os only
use_accelerate = True
```

Install a C/C++ compiler

The C-compiler installs with the Xcode Command Line Tools. No further action is required.

Install conda

Skip this step if you already have miniconda or anaconda for python>=3.6 installed on your system. Download the latest version of conda on your operating system from either miniconda or Anaconda websites. Make sure you download conda for python 3. Double click the downloaded .exe file and follow the installation steps.

OpenBLAS and FFTW libraries

Launch the `Anaconda prompt` (it should be located under the start menu). Within the anaconda prompt, type the following to install the package dependencies.

```
$ conda install -c conda-forge openblas fftw
```

Install a C/C++ compiler

Because the core of the mrsimulator package is written in C, you will require a C-compiler to build and install the package. Download and install the Microsoft Visual C++ compiler from Build Tools for Visual Studio 2019.

Building and Installing

Use the terminal/Prompt to navigate into the directory containing the package (usually, the folder is named mrsimulator),

```
$ cd mrsimulator
```

From within the source code folder, type the following in the terminal to install the library.

```
$ pip install .
```

If you get an error that you don't have the permission to install the package into the default `site-packages` directory, you may try installing with the `--user` options as,

```
$ pip install . --user
```

1.2.4 Test your build

If the installation is successful, you should be able to run the following test file in your terminal. Download the test file here.

```
$ python test_file.py
```

The above statement should produce the following figure.

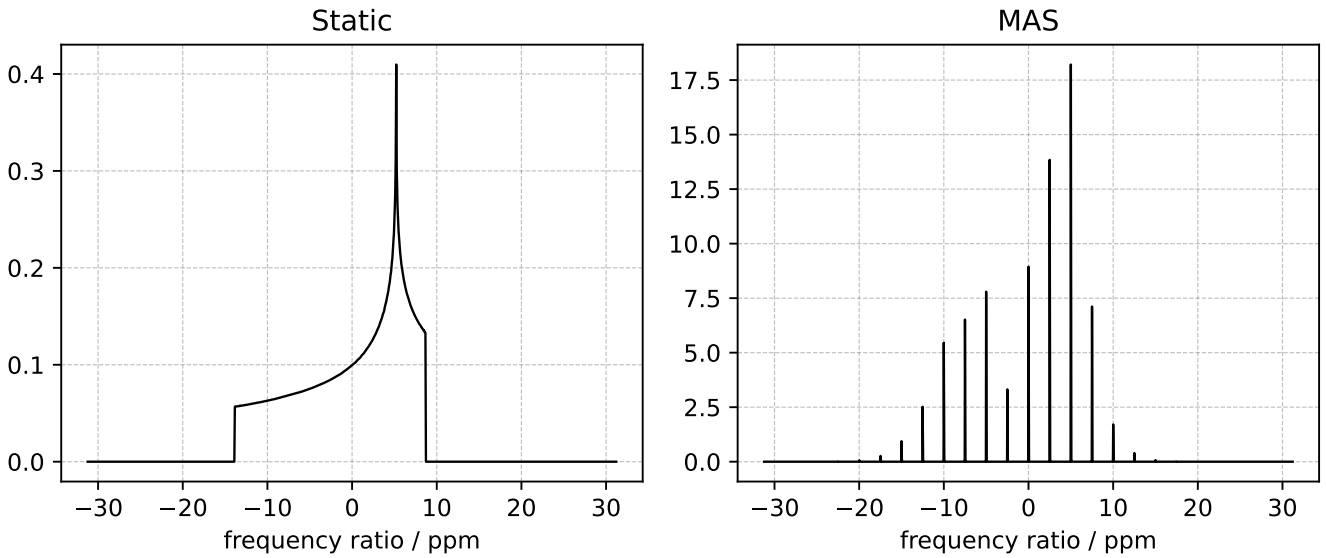


Figure 1.1: A test example simulation of solid-state NMR spectrum.

1.3 For developers and contributors

1.3.1 Make your own copy of mrsimulator on GitHub

Making a copy of someone's code on GitHub is the same as making a *fork*. A fork is a complete copy of the code and all of its revision history.

1. Log into a GitHub account.
2. Go to the mrsimulator Github home page.
3. Click on the *fork* button.

You will see a short animation of Octocat scanning a book on a flatbed scanner. After that, you should find yourself at the home page for your forked copy of mrsimulator.

1.3.2 Create a development environment

It is good practice to create separate virtual python environments for packages when in developer mode. The following is an example of a Conda environment.

```
$ conda create -n mrsimulator-dev python=3.7
```

The above command will create a new python3.7 environment named *mrsimulator-dev*. To activate the environment, use

```
$ conda activate mrsimulator-dev
```

1.3.3 Make sure git is installed on your computer

Git is the name of a source code management system. It keeps track of the changes made to the code and manages contributions from several different individuals. You may read more about git at the Git Basics.

If you are using anaconda/miniconda, you probably have git pre-installed. To check, type in terminal

```
$ git --version
# if git is installed, will get something like: git version 2.30.2
```

If git is not installed, install it.

Basic git configuration:

Follow the instructions at Set Up Git at GitHub to configure:

- Your user name and email in your copy of git.
- Authentication, so you don't have to type your GitHub password every time you need to access GitHub from the command line.

1.3.4 Copy your fork of mrsimulator from GitHub to your computer

Unless you plan on always editing the code using the online Github editor, you may need to copy the fork of mrsimulator from your GitHub account to your computer. Make a complete copy of the fork with

```
$ git clone --recursive https://github.com/your-user-name/mrsimulator.git
```

Insert *your-user-name* with your GitHub account username. If there is an error at this stage, it is probably an error in setting up authentication.

You now have a copy of the mrsimulator fork from your GitHub account to your local computer into a mrsimulator folder.

1.3.5 Understanding *Remotes*

In git, the name for another location of the same repository is *remote*. The repository that contains the latest "official" development version is traditionally called the *upstream* remote. You can read more about remotes on Git Basics.

At this point, your local copy of mrsimulator doesn't know where the *upstream* development version of mrsimulator is. To let git know, change into the mrsimulator folder you created in the previous step, and add a remote:

```
cd mrsimulator
git remote add mrsimulator git://github.com/DeepanshS/mrsimulator.git
```

You can check that everything is set up properly so far by asking git to show you all of the remotes it knows about for your local repository of mrsimulator with `git remote -v`, which should display something like

```
mrsimulator   git://github.com/DeepanshS/mrsimulator.git (fetch)
mrsimulator   git://github.com/DeepanshS/mrsimulator.git (push)
origin     git@github.com:your-user-name/mrsimulator.git (fetch)
origin     git@github.com:your-user-name/mrsimulator.git (push)
```

1.3.6 Build the development version of mrsimulator

OS-dependent prerequisites

Note: Installing OS-dependent prerequisites is a one-time process. If you are upgrading to a newer version of mrsimulator, skip to next section.

Linux

Mac OSX

Windows

OpenBLAS and FFTW libraries

On Linux, the package manager for your distribution is usually the easiest route to ensure you have the prerequisites to building the mrsimulator library. To build from source, you will need the OpenBLAS and FFTW development headers for your Linux distribution. Type the following command in the terminal, based on your Linux distribution.

For (Debian/Ubuntu):

```
$ sudo apt-get install libopenblas-dev libfftw3-dev
```

For (Fedora/RHEL):

```
$ sudo yum install openblas-devel fftw-devel
```

Install a C/C++ compiler

The C-compiler comes with your Linux distribution. No further action is required.

OpenBLAS/Accelerate and FFTW libraries

You will require the `brew` package manager to install the development headers for the OpenBLAS (if applicable) and FFTW libraries. Read more on installing brew from homebrew.

Step-1 Install the FFTW library using the homebrew formulae.

```
$ brew install fftw
```

Step-2 By default, the mrsimulator package links to the openblas library for BLAS operations. Mac users may opt to choose the in-build Apple's Accelerate library. If you opt for Apple's Accelerate library, skip to *Step-3*. If you wish to link the mrsimulator package to the OpenBLAS library, type the following in the terminal,

```
$ brew install openblas
```

Step-3 If you choose to link the mrsimulator package to the OpenBLAS library, skip to the next section, *Building and Installing* (page 9).

(a) You will need to install the BLAS development header for Apple's Accelerate library. The easiest way is to install the Xcode Command Line Tools. Note, this is a one-time installation. If you have previously installed the Xcode Command Line Tools, you may skip this sub-step. Type the following in the terminal,

```
$ xcode-select --install
```

(b) The next step is to let the mrsimulator setup know your preference. Open the `settings.py` file, located at the root level of the mrsimulator source code folder, in a text editor. You should see

```
# -*- coding: utf-8 -*-
# BLAS library
use_openblas = True
# mac-os only
use_accelerate = False
```

To link the mrsimulator package to the Apple's Accelerate library, change the fields to

```
# -*- coding: utf-8 -*-
# BLAS library
use_openblas = False
# mac-os only
use_accelerate = True
```

Install a C/C++ compiler

The C-compiler installs with the Xcode Command Line Tools. No further action is required.

Install conda

Skip this step if you already have miniconda or anaconda for python>=3.6 installed on your system. Download the latest version of conda on your operating system from either miniconda or Anaconda websites. Make sure you download conda for python 3. Double click the downloaded .exe file and follow the installation steps.

OpenBLAS and FFTW libraries

Launch the `Anaconda prompt` (it should be located under the start menu). Within the anaconda prompt, type the following to install the package dependencies.

```
$ conda install -c conda-forge openblas fftw
```

Install a C/C++ compiler

Because the core of the mrsimulator package is written in C, you will require a C-compiler to build and install the package. Download and install the Microsoft Visual C++ compiler from Build Tools for Visual Studio 2019.

Build and install

Before building the development version of mrsimulator, install the development requirement packages with pip. In the directory where your copy of mrsimulator is, type:

```
$ pip install -r requirements-dev.txt
$ pip install -e .
```

As always, if you get an error that you don't have the permission to install the package into the default site-packages directory, you may try installing by adding the `--user` option.

1.3.7 Note for the developers and contributors

Before commits: Mrsimulator follows python community standards for writing code and documentation. To help guide the developers and contributors towards these standards, we have created a *.pre-commit-config.yaml* file, that when used with `pre-commit`, will inspect the code and document for issues. Type `pre-commit run` before git commits to inspect the changes.

You can also set up the git hook script to automatically run *pre-commit* on git commits with the `pre-commit install`. Read more about pre-commit.

Running tests: For unit tests, we use the pytest module. At the root directory of the mrsimulator package folder, type

```
$ pytest
```

which will run a series of tests.

Building docs: We use the sphinx python documentation generator for building docs. Navigate to the *docs* folder within the mrsimulator package folder, and type,

```
$ make html
```

The above command will build the documentation and store the build at *mrsimulator/docs/_build/html*. Double click the *index.html* file within this folder to view the offline documentation.

CHAPTER

TWO

INTRODUCTION TO SPIN SYSTEMS

At the heart of any `mrsimulator` calculation is the definition of a **SpinSystem** object describing the sites and couplings within a spin system. We begin by examining the definition of a **Site** object.

2.1 Site

A site object holds single site NMR interaction parameters, which includes the nuclear shielding and quadrupolar interaction parameters. Consider the example below of a JSON serialization of a **Site** object for a deuterium nucleus.

Listing 2.1: An example 2H site in JSON representation.

```
{
    "isotope": "2H",
    "isotropic_chemical_shift": "4.1 ppm",
    "shielding_symmetric": {
        "zeta": "12.12 ppm",
        "eta": 0.82,
        "alpha": "5.45 rad",
        "beta": "4.82 rad",
        "gamma": "0.5 rad"
    },
    "quadrupolar": {
        "Cq": "1.47 MHz",
        "eta": 0.27,
        "alpha": "0.212 rad",
        "beta": "1.231 rad",
        "gamma": "3.1415 rad"
    }
}
```

The *isotope* key holds the spin isotope, here given a value of *2H*. The *isotropic_ chemical_ shift* is the isotropic chemical shift of the site isotope, ^{2}H, here given as *4.1 ppm*. We have additionally defined an optional *shielding_ symmetric* key, whose value holds a dictionary with the components of the second-rank traceless symmetric nuclear shielding tensor. We parameterize this tensor using the Haeberlen convention with parameters *zeta* and *eta*, defined as the shielding anisotropy and asymmetry, respectively. The Euler angle orientations, *alpha*, *beta*, and *gamma* are the relative orientation of the nuclear shielding tensor from a common reference frame.

Since deuterium is a quadrupolar nucleus, $I > 1/2$, there also can be a quadrupolar coupling interaction between the nuclear quadrupole moment and the surrounding electric field gradient (EFG) tensor, defined in a dictionary held in the optional key *quadrupolar*. An EFG tensor is a second-rank traceless symmetric tensor, and we describe its components with *Cq* and *eta*, i.e., the quadrupolar coupling constant and asymmetry parameter, respectively. Additionally, we

see the Euler angle orientations, *alpha*, *beta*, and *gamma*, which are the relative orientation of the EFG tensor from a common reference frame.

See Table 2.2 and Table 2.4 for further information on the **Site** and **SymmetricTensor** objects and their attributes, respectively.

2.2 Coupling

A coupling object holds two site NMR interaction parameters, which includes the J-coupling and the dipolar coupling interaction parameters. Consider the example below of a JSON serialization of a **Coupling** object.

Listing 2.2: A **Coupling** object in JSON representation.

```
{
  "site_index": [0, 1],
  "isotropic_j": "15 Hz",
  "j_symmetric": {
      "zeta": "12.12 Hz",
      "eta": 0.82,
      "alpha": "2.45 rad",
      "beta": "1.75 rad",
      "gamma": "0.15 rad"
  },
  "dipolar": {
      "D": "1.7 kHz",
      "alpha": "0.12 rad",
      "beta": "0.231 rad",
      "gamma": "1.1415 rad"
  }
}
```

The *site_index* key holds a list of two integers corresponding to the index of the two coupled sites within the spin system. The value of the *isotropic_j* is the isotropic J-coupling, here given as *15 Hz*. We have additionally defined an optional *j_symmetric* key, whose value holds a dictionary with the components of the second-rank traceless symmetric J-coupling tensor. We parameterize this tensor using the Haeberlen convention with parameters *zeta* and *eta*, defined as the J-coupling anisotropy and asymmetry parameters, respectively. The Euler angle orientations, *alpha*, *beta*, and *gamma* are the relative orientation of the J-coupling tensor from a common reference frame.

Additionally, the dipolar coupling interaction between the coupled nuclei is defined with an optional *dipolar* key. A dipolar tensor is a second-rank traceless symmetric tensor, and we describe the dipolar coupling constant with the parameter D. The Euler angle orientations, *alpha*, *beta*, and *gamma* are the relative orientation of the dipolar tensor from a common reference frame.

See Table 2.3 and Table 2.4 for further information on the **Site** and **SymmetricTensor** objects and their attributes, respectively.

2.3 SpinSystem

The **SpinSystem** object is a collection of sites and couplings within a spin system.

2.3.1 Uncoupled Spin System

Using the previous 2H **Site** object example, we construct a simple single-site **SpinSystem** object, as shown below.

Listing 2.3: An example of uncoupled 2H spin system in JSON representation.

```
{
    "name": "2H uncoupled spin system",
    "description": "An optional description of the spin system",
    "sites": [
      {
        "isotope": "2H",
        "isotropic_chemical_shift": "4.1 ppm",
        "shielding_symmetric": {
            "zeta": "12.12 ppm",
            "eta": 0.82,
            "alpha": "5.45 rad",
            "beta": "4.82 rad",
            "gamma": "0.5 rad"
        },
        "quadrupolar": {
            "Cq": "1.47 MHz",
            "eta": 0.27,
            "alpha": "0.212 rad",
            "beta": "1.231 rad",
            "gamma": "3.1415 rad"
        }
      }
    ],
    "abundance": "0.148%"
}
```

At the root level of the **SpinSystem** object, we find four keywords, **name**, **description**, **sites**, and **abundance**. The value of the *name* key is the optional name of the spin system, here given a value of *2H uncoupled spin system*. The value of the description key is an optional string describing the spin system. The value of the *sites* key is a list of **Site** objects. Here, this list comprises of single **Site** object (lines 5-22) from Listing 2.1. The value of the *abundance* key is the abundance of the spin system, here given a value of *0.148%*.

See Table 2.1 for further description of the **SpinSystem** class and its attributes.

2.3.2 Coupled Spin System

Appending to the previous single-site spin system example from Listing 2.3, we construct a two-spin coupled spin system, as follows.

Listing 2.4: An example of coupled 2H-13C spin system in JSON representation.

```
{
    "name": "2H-13C coupled spin system",
    "description": "An optional description of the spin system",
    "sites": [
      {
        "isotope": "2H",
        "isotropic_chemical_shift": "4.1 ppm",
        "shielding_symmetric": {
            "zeta": "12.12 ppm",
            "eta": 0.82,
            "alpha": "5.45 rad",
            "beta": "4.82 rad",
            "gamma": "0.5 rad"
        },
        "quadrupolar": {
            "Cq": "1.47 MHz",
            "eta": 0.27,
            "alpha": "0.212 rad",
            "beta": "1.231 rad",
            "gamma": "3.1415 rad"
        }
      },
      {
        "isotope": "13C",
        "isotropic_chemical_shift": "-53.2 ppm",
        "shielding_symmetric": {
            "zeta": "90.5 ppm",
            "eta": 0.64
        }
      }
    ],
    "couplings": [
      {
        "site_index": [0, 1],
        "isotropic_j": "15 Hz",
        "j_symmetric": {
            "zeta": "12.12 Hz",
            "eta": 0.82,
            "alpha": "2.45 rad",
            "beta": "1.75 rad",
            "gamma": "0.15 rad"
        },
        "dipolar": {
            "D": "1.7 kHz",
            "alpha": "0.12 rad",
            "beta": "0.231 rad",
```

(continues on next page)

(continued from previous page)

```
47              "gamma": "1.1415 rad"
48          }
49        }
50      ],
51      "abundance": "0.48%"
52  }
```

In comparison to the previous example, there are five keywords at the root level of the **SpinSystem** object, **name**, **description**, **sites**, **couplings**, and **abundance**. In this example, the value of the *sites* key holds a list of two **Site** objects. At index 0 (lines 5-22) is the *2H* site from Listing 2.1, and at index 1 (lines 23-30) is a *13C* site. The value of the *couplings* key is a list of **Coupling** objects. In this example, we define a single coupling object (lines 33-49) from Listing 2.2. The value of the *site_index* key within the coupling object, line 34, corresponds to the site index from the *sites* list.

2.3.3 Table of Class Attributes

Table 2.1: The attributes of a SpinSystem object.

Attributes	Type	Description
`name`	String	An *optional* attribute with a name for the spin system. Naming is a good practice as it improves the readability, especially when multiple spin systems are present. The default value is an empty string.
`description`	String	An *optional* attribute describing the spin system. The default value is an empty string.
`sites`	List	An *optional* list of *Site* (page 302) objects. The default value is an empty list.
`couplings`	List	An *optional* list of coupling objects. The default value is an empty list.
`abundance`	String	An *optional* quantity representing the abundance of the spin system. The abundance is given as percentage, for example, `25.4 %`. This value is useful when multiple spin systems are present. The default value is `100 %`.

Table 2.2: The attributes of a Site object.

Attribute name	Type	Description
`isotope`	String	A *required* isotope string given as the atomic number followed by the isotope symbol, for example, `13C`, `29Si`, `27Al`, and so on.
`isotropic_chemical_shift`	ScalarQuantity	An *optional* physical quantity describing the isotropic chemical shift of the site. The value is given in dimensionless frequency ratio, for example, `10 ppm` or `10 µHz/Hz`. The default value is `0 ppm`.
`shielding_symmetric`	*SymmetricTensor* (page 333)	An *optional* object describing the second-rank traceless symmetric nuclear shielding tensor following the Haeberlen convention. The default is a `NULL` object. See the description for the *SymmetricTensor* (page 333) object.
`quadrupolar`	*SymmetricTensor* (page 333)	An *optional* object describing the second-rank traceless electric quadrupole tensor. The default is a `NULL` object. See the description for the *SymmetricTensor* (page 333) object.

Table 2.3: The attributes of a Coupling object.

Attribute name	Type	Description
`site_index`	List of two integers	A *required* list with integers corresponding to the site index of the coupled sites, for example, [0, 1], [2, 1]. The order of the integers is irrelevant.
`isotropic_j`	ScalarQuantity	An *optional* physical quantity describing the isotropic *J*-coupling in Hz. The default value is `0 Hz`.
`j_symmetric`	*SymmetricTensor* (page 333)	An *optional* object describing the second-rank traceless symmetric *J*-coupling tensor following the Haeberlen convention. The default is a `NULL` object. See the description for the *SymmetricTensor* (page 333) object.
`dipolar`	*SymmetricTensor* (page 333)	An *optional* object describing the second-rank traceless dipolar tensor. The default is a `NULL` object. See the description for the *SymmetricTensor* (page 333) object.

Table 2.4: The attributes of a SymmetricTensor object.

Attribute name	Type	Description
`zeta` or `Cq` or `D`	ScalarQuantity	A *required* quantity. **Nuclear shielding:** The shielding anisotropy, `zeta`, calculated using the Haeberlen convention. The value is a physical quantity given in dimensionless frequency ratio, for example, `10 ppm` or `10 µHz/Hz`. **Electric quadrupole:** The quadrupole coupling constant, `Cq`. The value is a physical quantity given in units of frequency, for example, `3.1 MHz`. **J-coupling:** The J-coupling anisotropy, `zeta`, calculated using the Haeberlen convention. The value is a physical quantity given in frequency unit, for example, `10 Hz` or `0.3 kHz`. **Dipolar-coupling:** The dipolar-coupling constant, `D`. The value is a physical quantity given in frequency unit, for example, `1 kHz` or `9 kHz`.
`eta`	Float	A *required* asymmetry parameter calculated using the Haeberlen convention, for example, `0.75`. The parameter is set to zero for the dipolar tensor.
`alpha`	ScalarQuantity	An *optional* Euler angle, α. For example, `2.1 rad`. The default value is `0 rad`.
`beta`	ScalarQuantity	An *optional* Euler angle, β. For example, `90°`. The default value is `0 rad`.
`gamma`	ScalarQuantity	An *optional* Euler angle, γ. For example, `0.5 rad`. The default value is `0 rad`.

CHAPTER

THREE

THE BASICS

We have put together a set of guidelines for using the `mrsimulator` package. We encourage our users to follow these guidelines for consistency. In `mrsimulator`, the solid-state nuclear magnetic resonance (ssNMR) spectrum is calculated through an instance of the *Simulator* (page 289) class.

Import the *Simulator* (page 289) class using

```
>>> from mrsimulator import Simulator
```

and create an instance as follows,

```
>>> sim = Simulator()
```

Here, the variable `sim` is an instance of the *Simulator* (page 289) class. The two attributes of this class that you will frequently use are the ***spin_systems*** (page 289) and ***methods*** (page 289), whose values are a list of *SpinSystem* (page 297) and *Method* (page 310) objects, respectively. The default value of these attributes is an empty list.

```
>>> sim.spin_systems
[]
>>> sim.methods
[]
```

Before you can start simulating the NMR spectrum, you need to understand the role of the SpinSystem and Method objects. The following provides a brief description of the respective objects.

3.1 Setting up the SpinSystem objects

An NMR spin system is an isolated system of sites (spins) and couplings. You may construct a spin system with as many sites and couplings as necessary; for this example, we stick to a single-site spin system. Let's start by first building a site.

A site object is a collection of attributes that describe site-specific interactions. In NMR, these spin interactions are described by a second-rank tensor. Site-specific interactions include the interaction between the magnetic dipole moment of the nucleus and the surrounding magnetic field and the interaction between the electric quadrupole moment of the nucleus with the surrounding electric field gradient. The latter is zero for sites with the spin quantum number, $I = 1/2$.

Let's start with a spin-1/2 isotope, ^{29}Si, and create a site.

```
>>> the_site = {
...     "isotope": "29Si",
...     "isotropic_chemical_shift": "-101.1 ppm",
```

(continues on next page)

(continued from previous page)

```
...     "shielding_symmetric": {"zeta": "70.5 ppm", "eta": 0.5},
... }
```

In the above code, `the_site` is a simplified python dictionary representation of a *Site* (page 302) object. This site describes a ^{29}Si isotope with a -101.1 ppm isotropic chemical shift along with the symmetric part of the nuclear shielding anisotropy tensor, described here with the parameters *zeta* and *eta* using the Haeberlen convention.

That's it! Now that we have a site, we can create a single-site spin system following,

```
>>> the_spin_system = {
...     "name": "site A",
...     "description": "A test 29Si site",
...     "sites": [the_site],  # from the above code
...     "abundance": "80%",
... }
```

As mentioned before, a spin system is a collection of sites and couplings. In the above example, we have created a spin system with a single site and no coupling. Here, the attribute *sites* hold a list of sites. The attributes *name*, *description*, and *abundance* are optional.

Until now, we have only created a python dictionary representation of a spin system. To run the simulation, you need to create an instance of the *SpinSystem* (page 297) class. Import the SpinSystem class and use it's *parse_dict_with_units()* (page 301) method to parse the python dictionary and create an instance of the spin system class, as follows,

```
>>> from mrsimulator import SpinSystem
>>> system_object_1 = SpinSystem.parse_dict_with_units(the_spin_system)
```

Note: We provide the *parse_dict_with_units()* (page 301) method because it allows the user to create spin systems, where the attribute value is a physical quantity, represented as a string with a value and a unit. Physical quantities remove the ambiguity in the units, which is otherwise a source of general confusion within many scientific applications. With this said, parsing physical quantities can add significant overhead when used in an iterative algorithm, such as the least-squares minimization. In such cases, we recommend defining objects directly. See the *Uncoupled Spin System: Using objects* (page 29) for details.

We have successfully created a spin system object. To create more spin system objects, repeat the above set of instructions. In this example, we stick with a single spin system object. Once all spin system objects are ready, add these objects to the instance of the Simulator class, as follows

```
>>> sim.spin_systems += [system_object_1] # add all spin system objects.
```

3.2 Setting up the Method objects

A *Method* (page 310) object is a collection of attributes that describe an NMR method. In `mrsimulator`, all methods are described through five keywords -

Keywords	Description
channels	A list of isotope symbols over which the given method applies.
magnetic_flux_density	The macroscopic magnetic flux density of the applied external magnetic field.
rotor_angle	The angle between the sample rotation axis and the applied external magnetic field.
rotor_frequency	The sample rotation frequency.
spectral_dimensions	A list of spectral dimensions. The coordinates along each spectral dimension is described with the keywords, *count* (N), *spectral_width* (ν_{sw}), and *reference_offset* (ν_0). The coordinates are evaluated as, $$\left([0, 1, 2, ...N-1] - \frac{T}{2}\right)\frac{\nu_{\text{sw}}}{N} + \nu_0 \quad (3.1)$$ where $T = N$ when N is even else $T = N - 1$.

Let's start with the simplest method, the *BlochDecaySpectrum()* (page 320). The following is a python dictionary representation of the BlochDecaySpectrum method.

```
>>> method_dict = {
...     "channels": ["29Si"],
...     "magnetic_flux_density": "9.4 T",
...     "rotor_angle": "54.735 deg",
...     "rotor_frequency": "0 Hz",
...     "spectral_dimensions": [{
...         "count": 2048,
...         "spectral_width": "25 kHz",
...         "reference_offset": "-8 kHz",
...         "label": r"$^{29}$Si resonances",
...     }]
... }
```

Here, the key *channels* is a list of isotope symbols over which the method is applied. A Bloch Decay method only has a single channel. In this example, it is given a value of `29Si`, which implies that the simulated spectrum from this method will comprise frequency components arising from the ^{29}Si resonances. The keys *magnetic_flux_density*, *rotor_angle*, and *rotor_frequency* collectively describe the spin environment under which the resonance frequency is evaluated. The key *spectral_dimensions* is a list of spectral dimensions. A Bloch Decay method only has one spectral dimension. In this example, the spectral dimension defines a frequency dimension with 2048 points, spanning 25 kHz with a reference offset of -8 kHz.

Like before, you may parse the above `method_dict` using the `parse_dict_with_units()` function of the method. Import the BlochDecaySpectrum class and create an instance of the method, following,

```
>>> from mrsimulator.methods import BlochDecaySpectrum
>>> method_object = BlochDecaySpectrum.parse_dict_with_units(method_dict)
```

Here, `method_object` is an instance of the *Method* (page 310) class.

Likewise, you may create multiple method objects. In this example, we stick with a single method. Finally, add all the method objects, in this case, `method_object`, to the instance of the Simulator class, `sim`, as follows,

```
>>> sim.methods += [method_object] # add all methods.
```

3.3 Running simulation

To simulate the spectrum, run the simulator with the *run()* (page 294) method, as follows,

```
>>> sim.run()
```

Note: In `mrsimulator`, all resonant frequencies are calculated assuming the weakly-coupled (Zeeman) basis for the spin system.

The simulator object, `sim`, will process every method over all the spin systems and store the result in the *simulation* (page 311) attribute of the respective Method object. In this example, we have a single method. You may access the simulation data for this method as,

```
>>> data_0 = sim.methods[0].simulation
>>> # data_n = sim.method[n].simulation # when there are multiple methods.
```

Here, `data_0` is a CSDM object holding the simulation data from the method at index 0 of the *methods* (page 289) attribute from the `sim` object.

See also:

CSDM: The core scientific dataset model (CSDM)[1] is a lightweight and portable file format model for multi-dimensional scientific datasets and is supported by numerous NMR software—DMFIT, SIMPSON, jsNMR, and RMN. We also provide a python package csdmpy.

3.4 Visualizing the dataset

At this point, you may continue with additional post-simulation processing. We end this example with a plot of the data from the simulation. Figure 3.1 depicts the plot of the simulated spectrum.

For a quick plot of the csdm data, you may use the csdmpy library. The *csdmpy* package uses the matplotlib library to produce basic plots. You may optionally customize the plot using matplotlib methods.

```
>>> import matplotlib.pyplot as plt
>>> plt.figure(figsize=(6, 3.5)) # set the figure size
>>> ax = plt.subplot(projection='csdm')
>>> ax.plot(data_0, linewidth=1.5)
>>> ax.invert_xaxis() # reverse x-axis
>>> plt.tight_layout(pad=0.1)
>>> plt.show()
```

[1] Srivastava, D. J., Vosegaard, T., Massiot, D., Grandinetti, P. J. Core Scientific Dataset Model: A lightweight and portable model and file format for multi-dimensional scientific data. PLOS ONE, 2020, **15**, 1. DOI 10.1371/e0225953

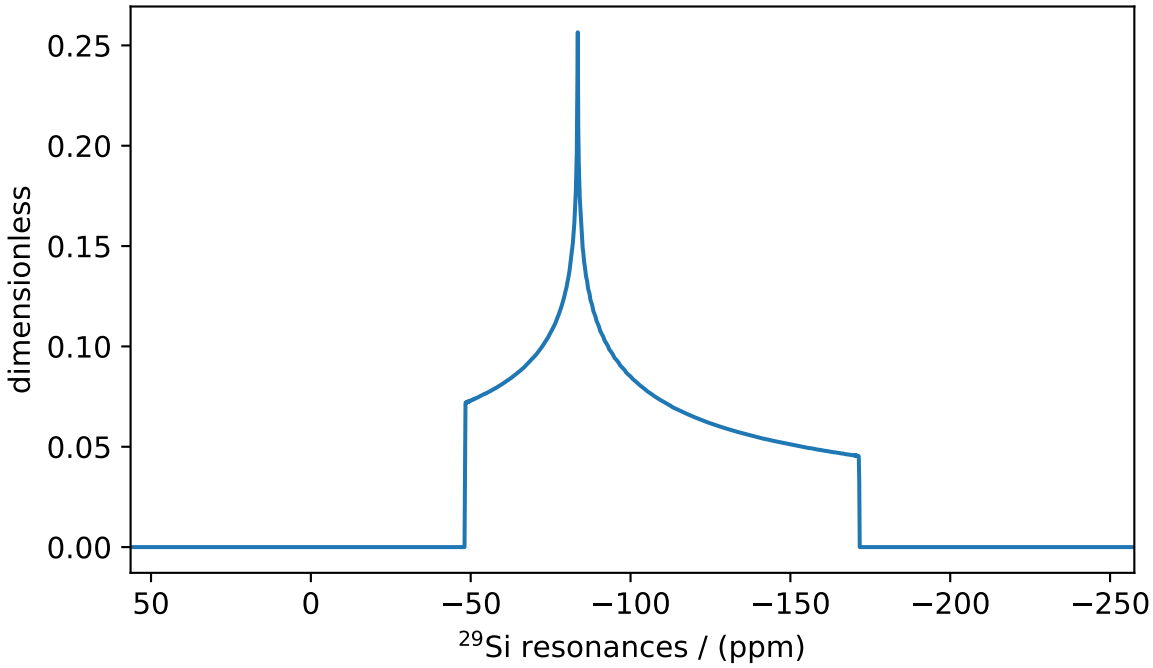


Figure 3.1: An example of solid-state static NMR spectrum simulation.

CHAPTER

FOUR

UNCOUPLED SPIN SYSTEM: USING OBJECTS

In the previous section on getting started, we show an examples where we parse the python dictionaries to create instances of the *SpinSystem* (page 297) and *Method* (page 310) objects. In this section, we'll illustrate how we can achieve the same result using the core `mrsimulator` objects.

Note: Unlike python dictionary objects from previous examples, when using `mrsimulator` objects, the attribute value is given as a number rather than a string with a number and a unit. We assume default units for the class attributes. To learn more about the default units, please refer to the documentation of the respective class. For the convenience of our users, we have added an attribute, `property_units`, to every class that holds the default unit of the respective class attributes.

Let's start by importing the classes.

```
>>> from mrsimulator import Simulator, SpinSystem, Site
>>> from mrsimulator.methods import BlochDecaySpectrum
```

The following code is used to produce the figures in this section.

```
>>> import matplotlib.pyplot as plt
>>> import matplotlib as mpl
>>> mpl.rcParams["figure.figsize"] = (6, 3.5)
>>> mpl.rcParams["font.size"] = 11
...
>>> # function to render figures.
>>> def plot(csdm_object):
...     # set matplotlib axes projection='csdm' to directly plot CSDM objects.
...     ax = plt.subplot(projection='csdm')
...     ax.plot(csdm_object, linewidth=1.5)
...     ax.invert_xaxis()
...     plt.tight_layout()
...     plt.show()
```

4.1 Site object

As the name suggests, a *Site* (page 302) object is used in creating sites. For example,

```
>>> C13A = Site(isotope='13C')
```

The above code creates a site with a ^{13}C isotope. Because, no further information is delivered to the site object, other attributes such as the isotropic chemical shift assume their default value.

```
>>> C13A.isotropic_chemical_shift # value is given in ppm
0.0
```

Here, the isotropic chemical shift is given in ppm. This information is also present in the `property_units` attribute of the instance. For example,

```
>>> C13A.property_units
{'isotropic_chemical_shift': 'ppm'}
```

Let's create a few more sites.

```
>>> C13B = Site(isotope='13C', isotropic_chemical_shift=-10)
>>> H1 = Site(isotope='1H', shielding_symmetric=dict(zeta=5.1, eta=0.1))
>>> O17 = Site(isotope='17O', isotropic_chemical_shift=41.7, quadrupolar=dict(Cq=5.15e6, eta=0.
↪21))
```

The site, `C13B`, is the second ^{13}C site with an isotropic chemical shift of -10 ppm.

In creating the site, `H1`, we use the dictionary object to describe a traceless symmetric second-rank irreducible nuclear shielding tensor, using the attributes *zeta* and *eta*, respectively. The parameter *zeta* and *eta* are defined as per the Haeberlen convention and describes the anisotropy and asymmetry parameter of the tensor, respectively. The default unit of the attributes from the *shielding_symmetric* is found with the `property_units` attribute, such as

```
>>> H1.shielding_symmetric.property_units
{'zeta': 'ppm', 'alpha': 'rad', 'beta': 'rad', 'gamma': 'rad'}
```

For site, `O17`, we once again make use of the dictionary object, only this time to describe a traceless symmetric second-rank irreducible electric quadrupole tensor, using the attributes *Cq* and *eta*, respectively. The parameter *Cq* is the quadrupole coupling constant, and *eta* is the asymmetry parameters of the quadrupole tensor, respectively. The default unit of these attributes is once again found with the `property_units` attribute,

```
>>> O17.quadrupolar.property_units
{'Cq': 'Hz', 'alpha': 'rad', 'beta': 'rad', 'gamma': 'rad'}
```

4.2 SpinSystem object

A SpinSystem object contains sites and couplings along with the abundance of the respective spin system. In this version, we focus on the spin systems with a single site, and therefore the couplings are irrelevant.

Let's use the sites we have already created to set up four spin systems.

```
>>> system_1 = SpinSystem(name='C13A', sites=[C13A], abundance=20)
>>> system_2 = SpinSystem(name='C13B', sites=[C13B], abundance=56)
```

(continues on next page)

(continued from previous page)

```
>>> system_3 = SpinSystem(name='H1', sites=[H1], abundance=100)
>>> system_4 = SpinSystem(name='O17', sites=[O17], abundance=1)
```

4.3 Method object

Likewise, we can create a *BlochDecaySpectrum* (page 320) object following,

```
>>> from mrsimulator.methods import BlochDecaySpectrum
>>> method_1 = BlochDecaySpectrum(
...     channels=["13C"],
...     spectral_dimensions = [dict(
...         count=2048,
...         spectral_width=25000, # in Hz.
...         label=r"$^{13}$C resonances",
...     )]
... )
```

The above method, `method_1`, is defined to record ^{13}C resonances over 25 kHz spectral width using 2048 points. The unspecified attributes, such as *rotor_frequency*, *rotor_angle*, *magnetic_flux_density*, assume their default value. The default units of these attributes is once again found with the `property_units` attribute,

```
>>> method_1.property_units
{'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}
```

4.4 Simulator object

The use of the simulator object is the same as described in the previous section.

```
>>> sim = Simulator()
>>> sim.spin_systems += [system_1, system_2, system_3, system_4] # add the spin systems
>>> sim.methods += [method_1] # add the method
```

4.5 Running simulation

Let's run the simulator and observe the spectrum.

```
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

Notice, we have four single-site spin systems within the `sim` object, two with ^{13}C sites, one with ^{1}H site, and one with an ^{17}O site, along with a BlochDecaySpectrum method which is tuned to record the resonances from the ^{13}C channel. When you run this simulation, only ^{13}C resonances are recorded, as seen from Figure 4.1, where just the two ^{13}C isotropic chemical shifts resonances are observed.

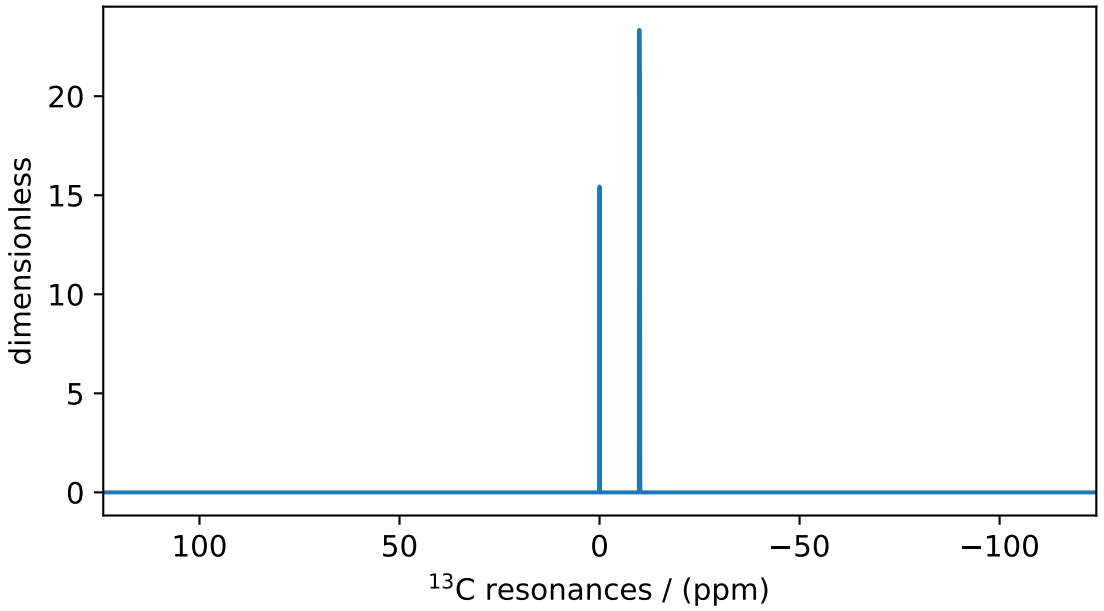


Figure 4.1: An example solid-state NMR simulation of ^{13}C isotropic spectrum.

4.5.1 Modifying the site attributes

Let's modify the `C13A` and `C13B` sites by adding the shielding tensors information.

```
>>> sim.spin_systems[0].sites[0].shielding_symmetric = dict(zeta=80, eta=0.5) # site C13A
>>> sim.spin_systems[1].sites[0].shielding_symmetric = dict(zeta=-100, eta=0.25) # site C13B
```

Running the simulation with the previously defined method will produce two overlapping CSA patterns, see Figure 4.2.

```
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

4.5.2 Modifying the rotor frequency of the method

Let's turn up the rotor frequency from 0 Hz (default) to 1 kHz. Note, that we do not add another method to the `sim` object, but update the existing method at index 0 with a new method. Figure 4.3 depicts the simulation from this method.

```
>>> # Update the method object at index 0.
>>> sim.methods[0] = BlochDecaySpectrum(
...     channels=["13C"],
...     rotor_frequency=1000, # in Hz.  <------------ updated entry
...     spectral_dimensions=[dict(
...         count=2048,
...         spectral_width=25000, # in Hz.
...         label=r"$^{13}$C resonances",
```

(continues on next page)

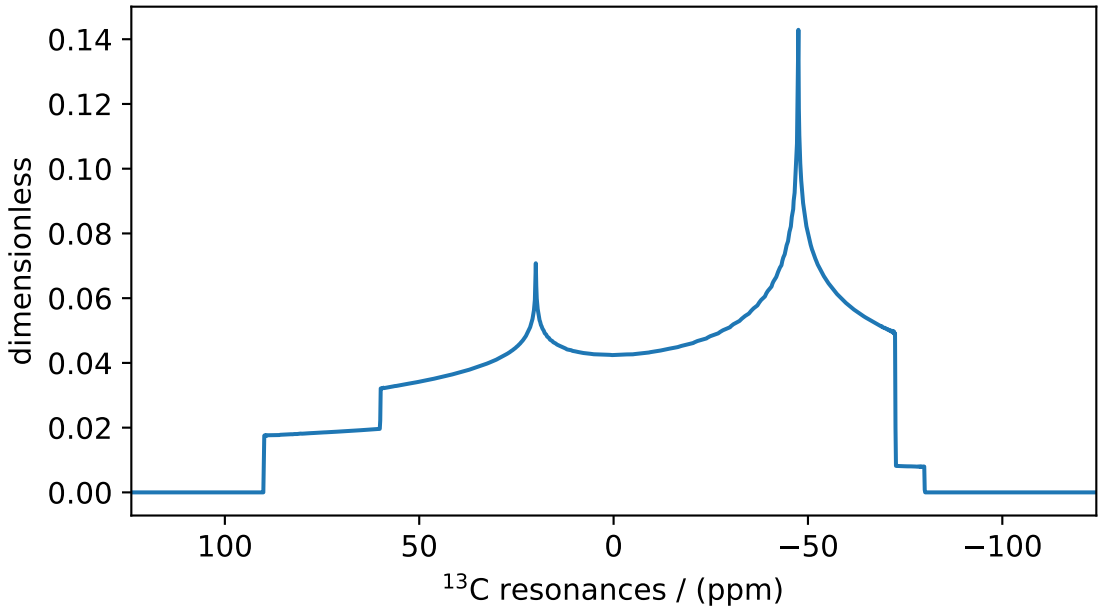


Figure 4.2: An example state-solid NMR simulation of static ^{13}C CSA spectrum.

(continued from previous page)

```
...     )]
... )
```

```
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

4.5.3 Modifying the rotor angle of the method

Let's also set the rotor angle from magic angle (default) to 90 degrees. Again, we update the method at index 0. Figure 4.4 depicts the simulation from this method.

```
>>> # Update the method object at index 0.
>>> sim.methods[0] = BlochDecaySpectrum(
...     channels=["13C"],
...     rotor_frequency=1000, # in Hz.
...     rotor_angle=90*3.1415926/180, # 90 degree in radians.  <------------ updated entry
...     spectral_dimensions=[dict(
...         count=2048,
...         spectral_width=25000, # in Hz.
...         label=r"$^{13}$C resonances",
...     )]
... )
```

```
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

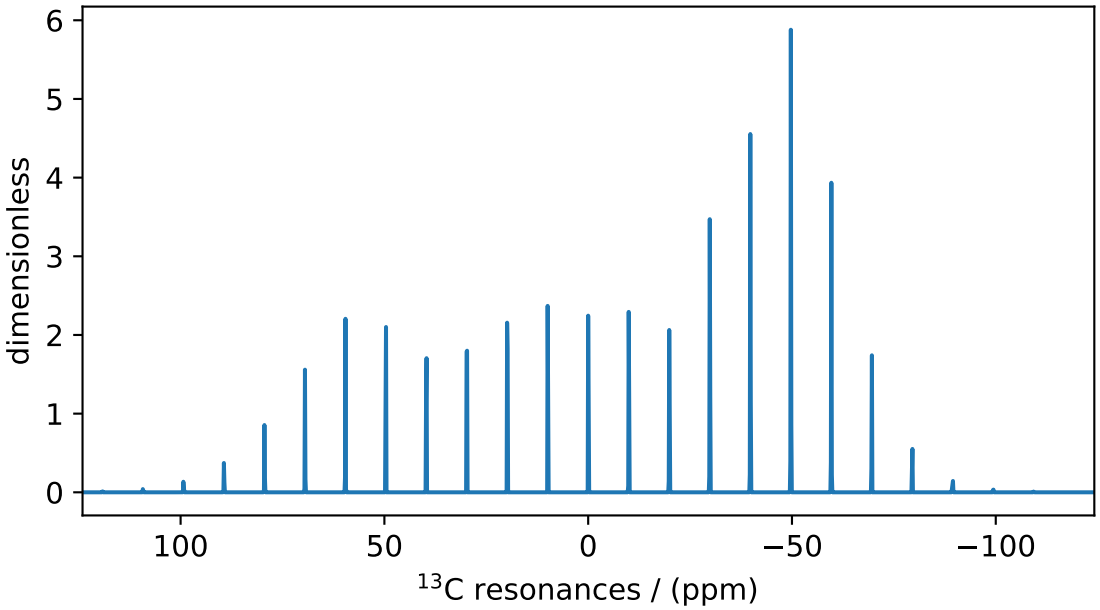


Figure 4.3: An example of the solid-state ^{13}C MAS sideband simulation.

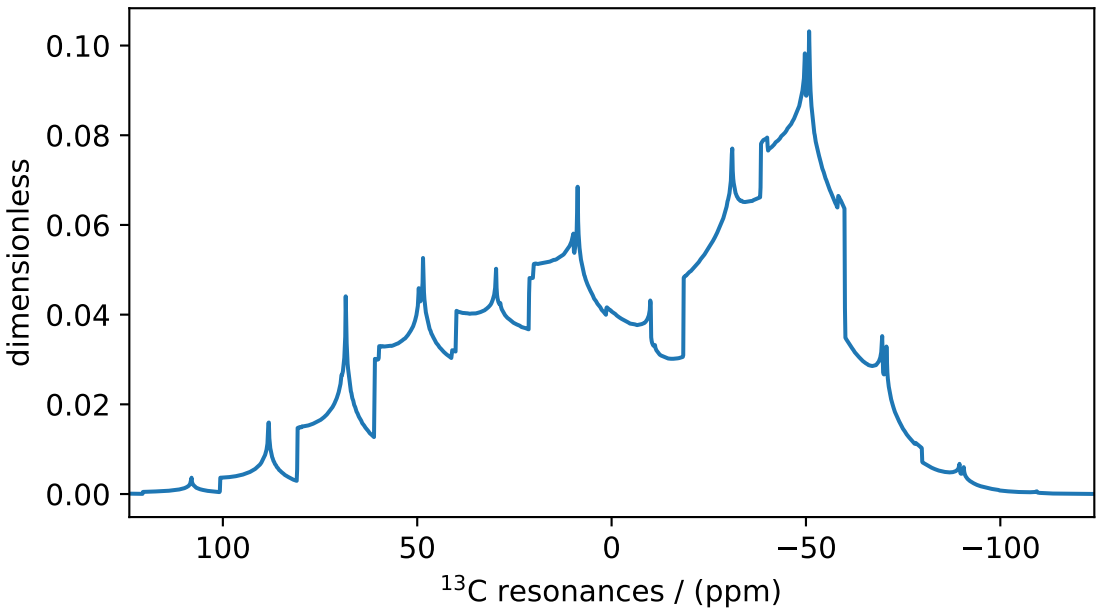


Figure 4.4: An example of the solid-state ^{13}C VAS sideband simulation.

4.5.4 Switching the detection channels of the method

To switch to another channels, update the value of the *channels* attribute of the method. Here, we update the method to *1H* channel.

```
>>> # Update the method object at index 0.
>>> sim.methods[0] = BlochDecaySpectrum(
...     channels=["1H"], # <------------ updated entry
...     rotor_frequency=1000, # in Hz.
...     rotor_angle=90*3.1415926/180, # 90 degree in radians.
...     spectral_dimensions=[dict(
...         count=2048,
...         spectral_width=25000, # in Hz.
...         label=r"$^1$H resonances",
...     )]
... )
```

```
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

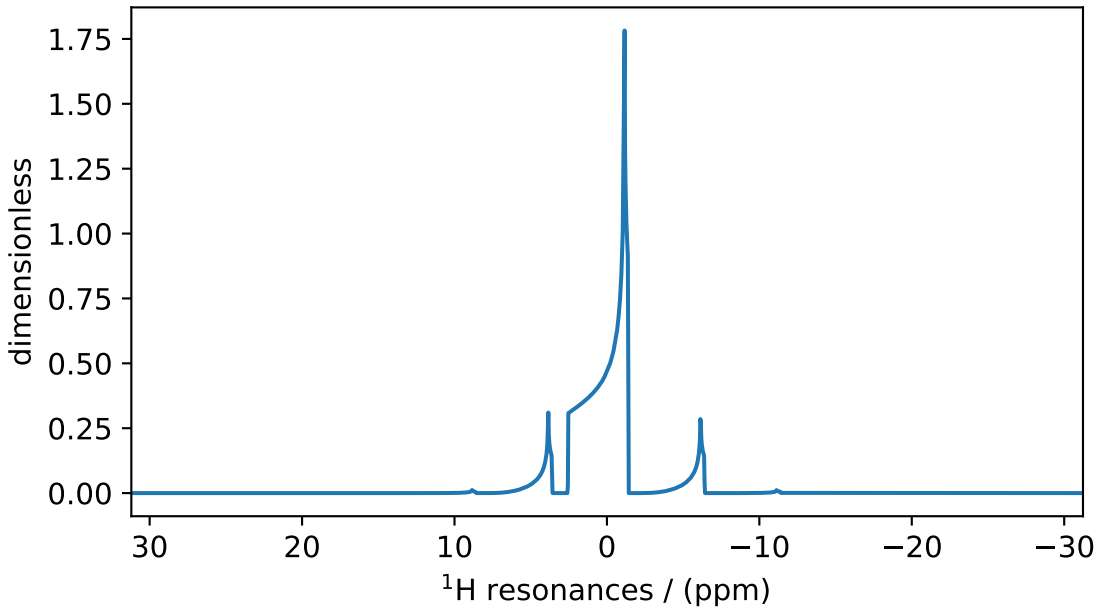


Figure 4.5: An example of solid-state ^{1}H VAS sideband simulation.

In Figure 4.5, we see a 90° spinning sideband ^{1}H-spectrum, whose frequency contributions arise from `system_3` because `system_3` is the only spin system with ^{1}H site.

Note, although you are free to assign any channel to the ***channels*** (page 310) attribute of the BlochDecaySpectrum method, only channels whose isotopes are also a member of the spin systems will produce a spectrum. For example, the following method

```
>>> # Update the method object at index 0.
>>> sim.methods[0] = BlochDecaySpectrum(
...     channels=["23Na"], # <------------ updated entry
...     rotor_frequency=1000, # in Hz.
...     rotor_angle=90*3.1415926/180, # 90 degree in radians.
...     spectral_dimensions=[dict(
...         count=2048,
...         spectral_width=25000, # in Hz.
...         label=r"$^{23}$Na resonances",
...     )]
... )
```

is defined to collect the resonances from ^{23}Na isotope. As you may have noticed, we do not have any ^{23}Na site in the spin systems. Simulating the spectrum from this method will result in a zero amplitude spectrum, see Figure 4.6.

```
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

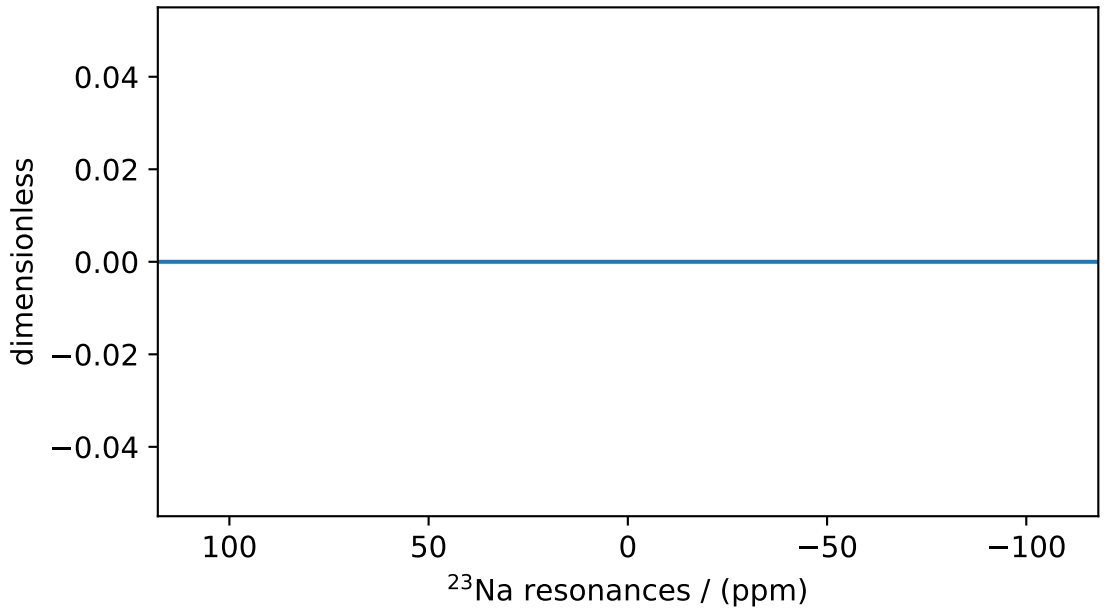


Figure 4.6: An example of a simulation where the isotope from the method's channel attribute does not exist within the spin systems.

Switching the channel to 17O

Likewise, update the value of the *channels* attribute to *17O*.

```
>>> sim.methods[0] = BlochDecaySpectrum(
...     channels=["17O"],
...     rotor_frequency= 15000, # in Hz.
...     rotor_angle = 0.9553166, # magic angle is rad.
...     spectral_dimensions=[dict(
...         count=2048,
...         spectral_width=25000, # in Hz.
...         label=r"$^{17}$O resonances",
...     )]
... )
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

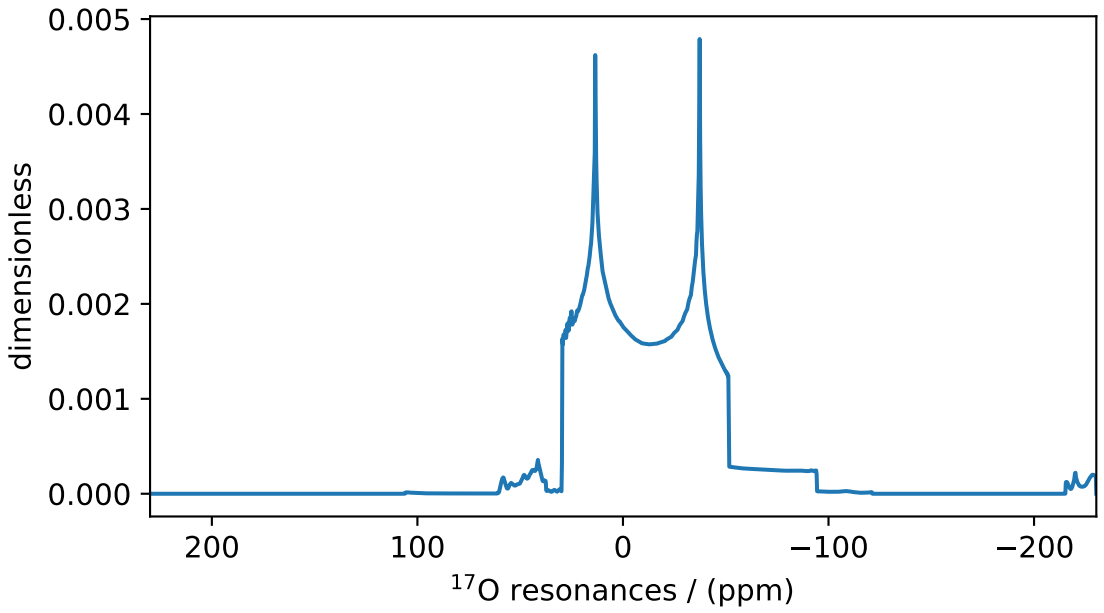


Figure 4.7: An example of the solid-state ^{17}O BlochDecaySpectrum simulation.

If you are familiar with the quadrupolar line-shapes, you may immediately associate the spectrum in Figure 4.7 to a second-order quadrupolar line-shape of the central transition. You may also notice some unexpected resonances around 50 ppm and -220 ppm. These unexpected resonances are the spinning sidebands of the satellite transitions. Note, the BlochDecaySpectrum method computes resonances from all transitions with $p = \Delta m = -1$.

Let's see what transition pathways are used in our simulation. Use the *get_transition_pathways()* (page 312) function of the Method instance to see the list of transition pathways, for example,

```
>>> from pprint import pprint
>>> pprint(sim.methods[0].get_transition_pathways(system_4)) # 17O
[|-2.5⟩⟨-1.5|, |-1.5⟩⟨-0.5|, |-0.5⟩⟨0.5|, |0.5⟩⟨1.5|, |1.5⟩⟨2.5|]
```

Notice, there are five transition pathways for the ^{17}O site, one associated with the central-transition, two with the inner-satellites, and two with the outer-satellites. For central transition selective simulation, use the *BlochDecayCTSpectrum* (page 321) method.

```
>>> from mrsimulator.methods import BlochDecayCTSpectrum
>>> sim.methods[0] = BlochDecayCTSpectrum(
...     channels=["17O"],
...     rotor_frequency= 15000, # in Hz.
...     rotor_angle = 0.9553166, # magic angle is rad.
...     spectral_dimensions=[dict(
...         count=2048,
...         spectral_width=25000, # in Hz.
...         label=r"$^{17}$O resonances",
...     )]
... )
>>> # the transition pathways
>>> print(sim.methods[0].get_transition_pathways(system_4)) # 17O
[|-0.5⟩⟨0.5|]
```

Now, you may simulate the central transition selective spectrum. Figure 4.8 depicts a central transition selective spectrum.

```
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

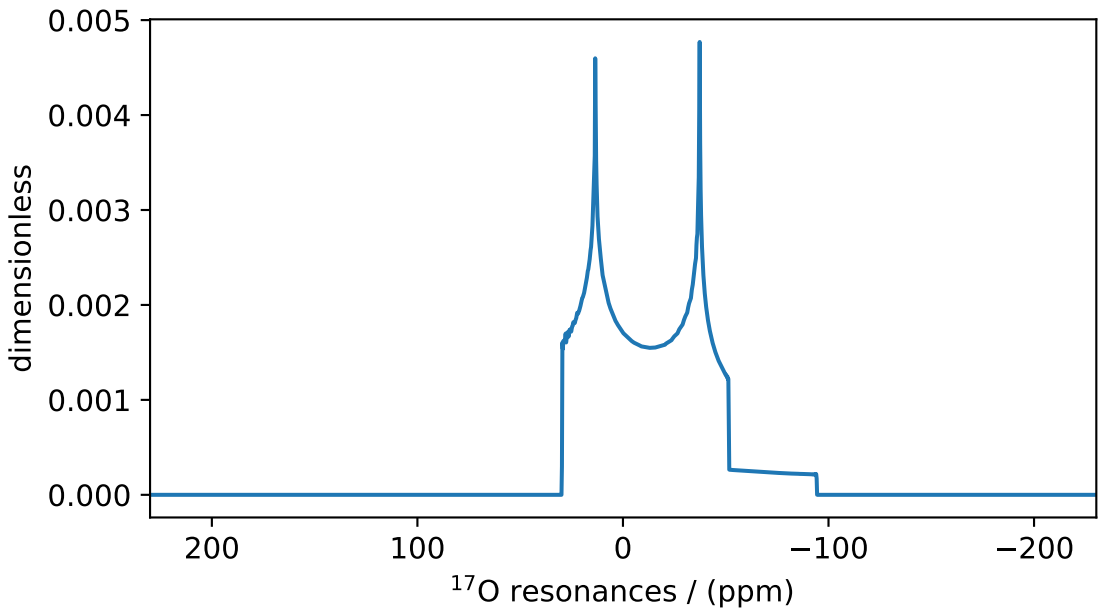


Figure 4.8: An example of the solid-state ^{17}O BlochDecayCTSpectrum simulation.

CHAPTER

FIVE

COUPLED SPIN SYSTEM: USING OBJECTS

In this example, we will simulate the ^{1}H NMR spectrum of ethanol using the core `mrsimulator` objects. Let's start by importing all the necessary packages.

```
>>> import matplotlib.pyplot as plt
>>> from mrsimulator import Simulator, SpinSystem, Site, Coupling
>>> from mrsimulator.methods import BlochDecaySpectrum
```

5.1 Setting up coupled SpinSystem objects

5.1.1 Sites

An NMR spin system is defined by an isolated set of sites (spins) and couplings. You can make a spin system as large and complex as needed, but for this example, we will build the most abundant isotopomer of the ethanol molecule (all carbons are ^{12}C, and the oxygen is ^{16}O). We start by defining the three distinct proton sites and then build a list to hold all the sites (site indices correspond to atoms as shown in the structure below).

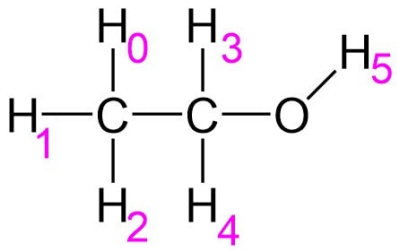


Figure 5.1: A representation of Ethanol molecule. The proton subscripts correspond to the site indexes used in the spin system.

```
>>> H_CH3 = Site(isotope='1H', isotropic_chemical_shift=1.226)  # methyl proton
>>> H_CH2 = Site(isotope='1H', isotropic_chemical_shift=2.61)  # methylene proton
>>> H_OH = Site(isotope='1H', isotropic_chemical_shift=3.687)  # hydroxyl proton
...
>>> etoh_sites = [H_CH3, H_CH3, H_CH3, H_CH2, H_CH2, H_OH]
```

5.1.2 Couplings

Now, we need to define the ${}^3J_{HH}$ couplings that cause the splittings in the spectrum of ethanol. In `mrsimulator`, all couplings are defined using the *Coupling* (page 306) class. Let's start by defining a coupling between a methyl and a methylene proton.

```
>>> HH_coupling_1 = Coupling(site_index=[0, 3], isotropic_j=7)
>>> HH_coupling_1.property_units
{'isotropic_j': 'Hz'}
```

The attribute *site_index* holds a pair of integers, where each integer is the index of the coupled site object. The attribute *isotropic_j* is the isotropic J-coupling between the coupled sites in units of *Hz*. Like every other object, the information on the default unit is held with the `property_units` attribute. In the above example, we define a coupling between site 0 (methyl) and site 3 (methylene). The indexes 0 and 3 are relative to the list of site objects in `etoh_sites`. The isotropic J-coupling is 7 Hz. Now, we define the rest of the methyl-methylene couplings and make a list to hold them all.

Note: Strong J couplings do not lead to splittings in an NMR spectrum, so they are not included in the list of couplings in this example. Additionally, note that `mrsimulator` library only supports weak couplings between sites.

```
>>> HH_coupling_2 = Coupling(site_index=[0, 4], isotropic_j=7)
>>> HH_coupling_3 = Coupling(site_index=[1, 3], isotropic_j=7)
>>> HH_coupling_4 = Coupling(site_index=[1, 4], isotropic_j=7)
>>> HH_coupling_5 = Coupling(site_index=[2, 3], isotropic_j=7)
>>> HH_coupling_6 = Coupling(site_index=[2, 4], isotropic_j=7)
>>>
>>> etoh_couplings = [
...     HH_coupling_1,
...     HH_coupling_2,
...     HH_coupling_3,
...     HH_coupling_4,
...     HH_coupling_5,
...     HH_coupling_6,
... ]
```

5.1.3 Spin system

Now, we add the sites and couplings to the spin system object.

```
>>> etoh = SpinSystem(sites=etoh_sites, couplings=etoh_couplings)
```

We have successfully built our ethanol spin system. If you need to create more spin systems, repeat these instructions, but for this example, we will stick with a single spin system.

5.2 Setting up the Method objects

Next, we create a method to simulate a simple 1D pulse-acquire ^1H spectrum.

```
>>> method_H = BlochDecaySpectrum(
...     channels=['1H'],
...     magnetic_flux_density=9.4,  # T
...     spectral_dimensions=[{
...         "count": 3000,
...         "spectral_width": 1.5e3,  # in Hz
...         "reference_offset": 940,  # in Hz
...         "label": "$^{1}$H frequency",
...     }],
... )
```

In the above code, *channels* is a list of isotope symbols that a method will use. The Bloch Decay method only uses one channel, and in this case we are simulating a ^{1}H spectrum. *magnetic_flux_density* describes the environment under which the resonance frequency is evaluated. *spectral_dimensions* contains a list of spectral dimensions (only one for the Bloch Decay method). In this case, we define a frequency dimension with 3000 points, spanning 1.5 kHz with a reference offset of 940 Hz.

You can create as many methods as you need, but in this case we will stick with the one method.

5.3 Running simulation

Next, we need to create an instance of the simulator object and then add our spin system and method to it. Then, we run the simulator with the *run()* (page 294) method.

```
>>> sim = Simulator()
>>> sim.spin_systems = [etoh]
>>> sim.methods = [method_H]
>>> sim.run()
```

The simulator object has now processed the method with our spin system and has stored the result in the simulation attribute of that method. Let's get the data from the method so we can plot it.

```
>>> H_data = sim.methods[0].simulation
```

5.4 Visualizing the dataset

Now that we have our data, let's plot the spectrum using matplotlib.

```
>>> plt.figure(figsize=(10, 4)) # set the figure size
>>> ax = plt.subplot(projection='csdm')
>>> ax.plot(H_data.real, color="black", linewidth=0.5)
>>> ax.set_xlim(4, 0.75)
>>> plt.tight_layout()
>>> plt.show()
```

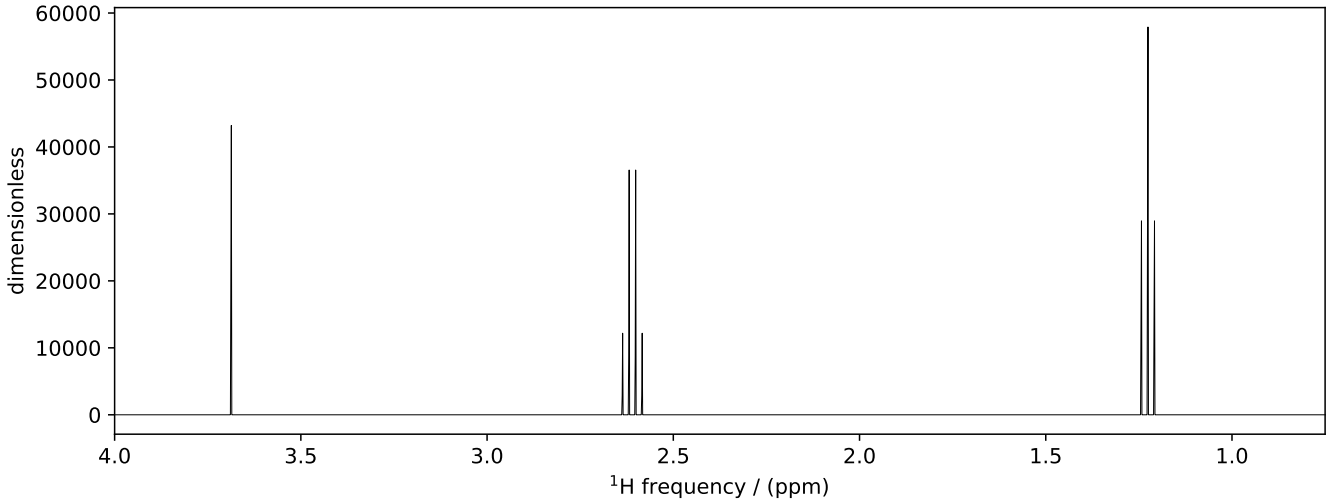


Figure 5.2: An example ^{1}H NMR spectrum simulation of Ethanol.

CHAPTER

SIX

CONFIGURING SIMULATOR OBJECT

The following code is used to produce the figures in this section.

```
>>> import matplotlib.pyplot as plt
>>> import matplotlib as mpl
>>> mpl.rcParams["figure.figsize"] = (6, 3.5)
>>> mpl.rcParams["font.size"] = 11
...
>>> # function to render figures.
>>> def plot(csdm_object):
...     # set matplotlib axes projection='csdm' to directly plot CSDM objects.
...     ax = plt.subplot(projection='csdm')
...     ax.plot(csdm_object, linewidth=1.5)
...     ax.invert_xaxis()
...     plt.tight_layout()
...     plt.show()
```

Up until now, we have been using the simulator object with the default setting. In `mrsimulator`, we choose the default settings such that it applies to a wide range of simulations including, static, magic angle spinning (MAS), and variable angle spinning (VAS) spectra. In certain situations, however, the default settings are not sufficient to accurately represent the spectrum. In such cases, the user can modify these settings as required. In the following section, we briefly describe the configuration settings.

The *Simulator* (page 289) class is configured using the *config* (page 290) attribute. The default value of the config attributes is as follows,

```
>>> from mrsimulator import Simulator, SpinSystem, Site
>>> from mrsimulator.methods import BlochDecaySpectrum
...
>>> sim = Simulator()
>>> sim.config
ConfigSimulator(number_of_sidebands=64, integration_volume='octant', integration_density=70,␣
↪decompose_spectrum='none')
```

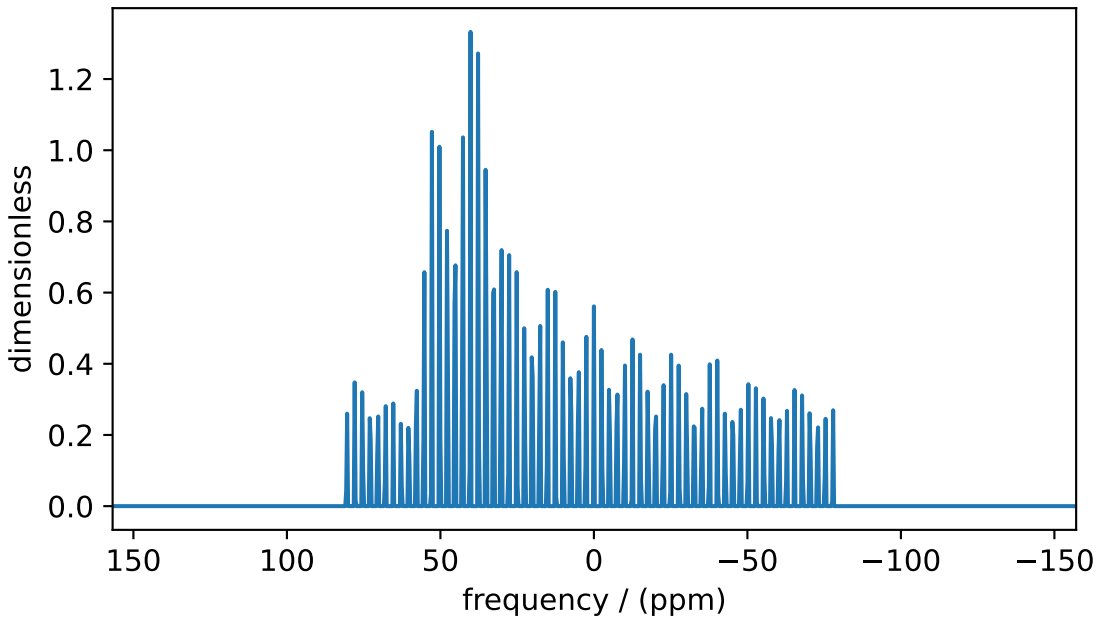
Here, the configurable attributes are `number_of_sidebands`, `integration_volume`, `integration_density`, and `decompose_spectrum`.

6.1 Number of sidebands

The value of this attribute is the number of sidebands requested in evaluating the spectrum. The default value is 64 and is sufficient for most cases, as seen from our previous examples. In certain circumstances, especially when the anisotropy is large or the rotor spin frequency is low, 64 sidebands might not be sufficient. In such cases, the user will need to increase the value of this attribute as required. Conversely, 64 sidebands might be redundant for other problems, in which case the user may want to reduce the value of this attribute. Note, reducing the number of sidebands will significantly improve computation performance, which might save computation time when used in iterative algorithms, such as least-squares minimization.

The following is an example of when the number of sidebands is insufficient.

```
>>> sim = Simulator()
...
>>> # create a site with a large anisotropy, 100 ppm.
>>> Si29site = Site(isotope='29Si', shielding_symmetric={'zeta': 100, 'eta': 0.2})
...
>>> # create a method. Set a low rotor frequency, 200 Hz.
>>> method = BlochDecaySpectrum(
...     channels=['29Si'],
...     rotor_frequency=200, # in Hz.
...     spectral_dimensions=[dict(count=1024, spectral_width=25000)]
... )
...
>>> sim.spin_systems += [SpinSystem(sites=[Si29site])]
>>> sim.methods += [method]
...
>>> # simulate and plot
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```



If you are familiar with the NMR spinning sideband patterns, you may notice that the sideband simulation spectrum

Figure 6.1: Inaccurate spinning sidebands simulation resulting from computing a relatively low number of sidebands.

in Figure 6.1 is inaccurate, as evident from the abrupt termination of the sideband amplitudes at the edges. As mentioned earlier, this inaccuracy arises from evaluating a small number of sidebands relative to the given anisotropy. Let's increase the number of sidebands to 90 and observe. Figure 6.2 depicts an accurate spinning sideband simulation.

```
>>> # set the number of sidebands to 90.
>>> sim.config.number_of_sidebands = 90
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

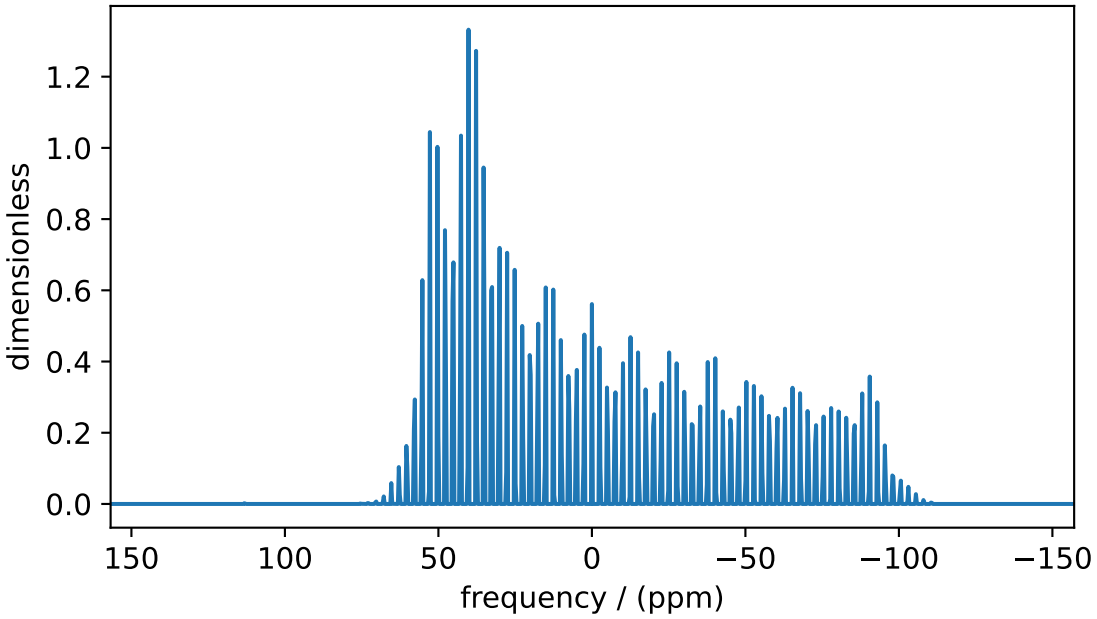


Figure 6.2: Accurate spinning sideband simulation when using a large number of sidebands.

6.2 Integration volume

The attribute *integration_volume* is an enumeration with two literals, `octant` and `hemisphere`. The integration volume refers to the volume of the sphere over which the NMR frequencies are integrated. The default value is *octant*, i.e., the spectrum comprises of integrated frequencies arising from the positive octant of the sphere. The `mrsimulator` package enables the user to exploit the orientational symmetry of the problem, and thus optimize the simulation by performing a partial integration —*octant* or *hemisphere*. To learn more about the orientational symmetries, please refer to Eden et. al.[1]

Consider the ^{29}Si site, `Si29site`, from the previous example. This site has a symmetric shielding tensor with *zeta* and *eta* as 100 ppm and 0.2, respectively. With only *zeta* and *eta*, we can exploit the symmetry of the problem, and evaluate the frequency integral over the octant, which is equivalent to the integration over the sphere. By adding the

[1] Edén, M. and Levitt, M. H. Computation of orientational averages in solid-state nmr by gaussian spherical quadrature. J. Mag. Res., **132**, *2*, 220–239, 1998. doi:10.1006/jmre.1998.1427.

Euler angles to this tensor, we break the symmetry, and the integration over the octant is no longer accurate. Consider the following examples.

```
>>> # add Euler angles to the shielding tensor.
>>> Si29site.shielding_symmetric.alpha = 1.563 # in rad
>>> Si29site.shielding_symmetric.beta = 1.2131 # in rad
>>> Si29site.shielding_symmetric.gamma = 2.132 # in rad
...
>>> # Let's observe the static spectrum which is more intuitive.
>>> sim.methods[0] = BlochDecaySpectrum(
...     channels=['29Si'],
...     rotor_frequency=0, # in Hz.
...     spectral_dimensions=[dict(count=1024, spectral_width=25000)]
... )
...
>>> # simulate and plot
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

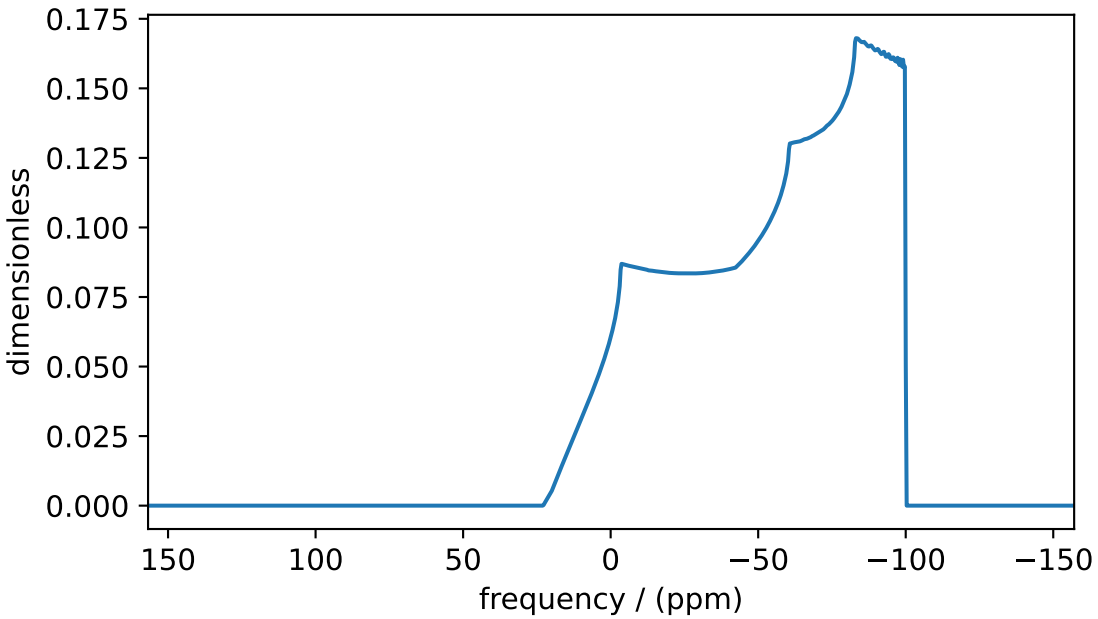


Figure 6.3: An example of an incomplete spectral averaging, where the simulation comprises of frequency contributions evaluated over the positive octant.

The spectrum in Figure 6.3 is incorrect. To fix this, set the integration volume to *hemisphere* and re-simulate. Figure 6.4 depicts the accurate simulation of the CSA tensor.

```
>>> # set integration volume to 'hemisphere'.
>>> sim.config.integration_volume = 'hemisphere'
...
>>> # simulate and plot
>>> sim.run()
```

(continues on next page)

(continued from previous page)

```
>>> plot(sim.methods[0].simulation)
```

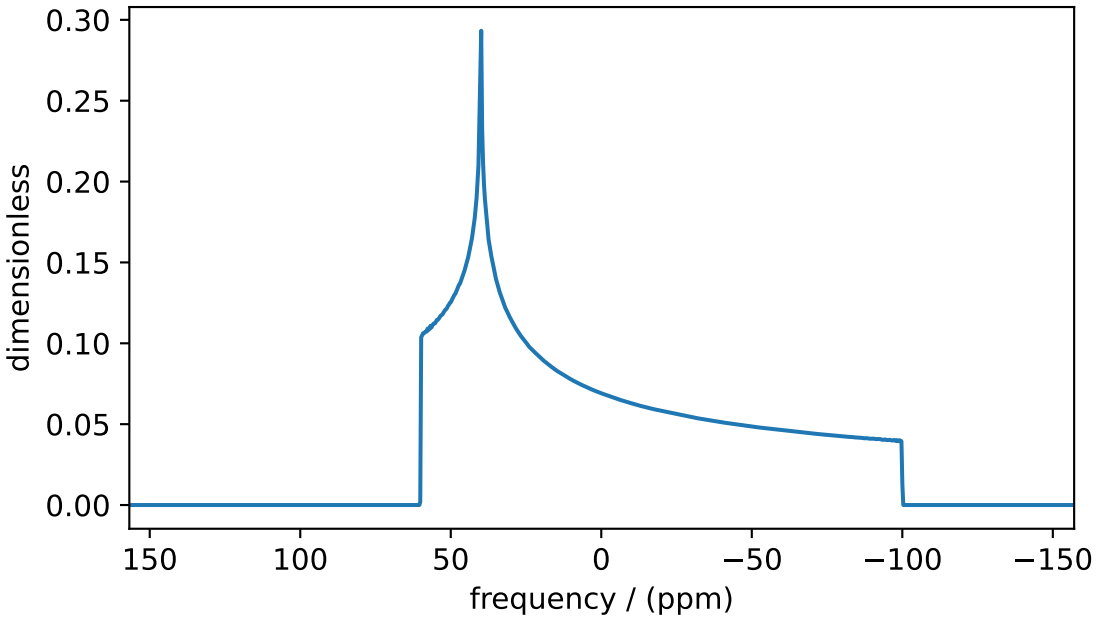


Figure 6.4: The spectrum resulting from the frequency contributions evaluated over the top hemisphere.

6.3 Integration density

Integration density controls the number of orientational points sampled over the given volume. The resulting spectrum is an integration of the NMR resonance frequency evaluated at these orientations. The total number of orientations, Θ_{count}, is given as

$$\Theta_{\text{count}} = M(n+1)(n+2)/2, \tag{6.1}$$

where M is the number of octants and n is value of this attribute. The number of octants is deciphered form the value of the *integration_ volume* attribute. The default value of this attribute, 70, produces 2556 orientations at which the NMR frequency contribution is evaluated. The user may increase or decrease the value of this attribute as required by the problem.

Consider the following example.

```
>>> sim = Simulator()
>>> sim.config.integration_density
70
>>> sim.config.get_orientations_count() # 1 * 71 * 72 / 2
2556
>>> sim.config.integration_density = 100
>>> sim.config.get_orientations_count() # 1 * 101 * 102 / 2
5151
```

6.4 Decompose spectrum

The attribute *decompose_spectrum* is an enumeration with two literals, `none`, and `spin_system`. The value of this attribute lets us know how the user intends the simulation to be stored.

6.4.1 none

If the value is `none` (default), the result of the simulation is a single spectrum where the frequency contributions from all the spin systems are co-added. Consider the following example.

```
>>> # Create two sites
>>> site_A = Site(isotope='1H', shielding_symmetric={'zeta': 5, 'eta': 0.1})
>>> site_B = Site(isotope='1H', shielding_symmetric={'zeta': -2, 'eta': 0.83})
...
>>> # Create two spin systems, each with single site.
>>> system_A = SpinSystem(sites=[site_A], name='System-A')
>>> system_B = SpinSystem(sites=[site_B], name='System-B')
...
>>> # Create a method object.
>>> method = BlochDecaySpectrum(
...     channels=['1H'],
...     spectral_dimensions=[dict(count=1024, spectral_width=10000)]
... )
...
>>> # Create simulator object.
>>> sim = Simulator()
>>> sim.spin_systems += [system_A,  system_B] # add the spin systems
>>> sim.methods += [method] # add the method
...
>>> # simulate and plot.
>>> sim.run()
>>> plot(sim.methods[0].simulation)
```

Figure 6.5 depicts the simulation of the spectrum from two spin systems where the contributions from individual spin systems are co-added.

6.4.2 spin_system

When the value of this attribute is `spin_system`, the resulting simulation is a series of spectra, each arising from a spin system. In this case, the number of spectra is the same as the number of spin system objects. Try setting the value of the decompose_spectrum attribute to *spin_system* and observe the simulation.

```
>>> # set decompose_spectrum to true.
>>> sim.config.decompose_spectrum = "spin_system"
...
>>> # simulate.
>>> sim.run()
...
>>> # plot of the two spectrum
>>> plot(sim.methods[0].simulation)
```

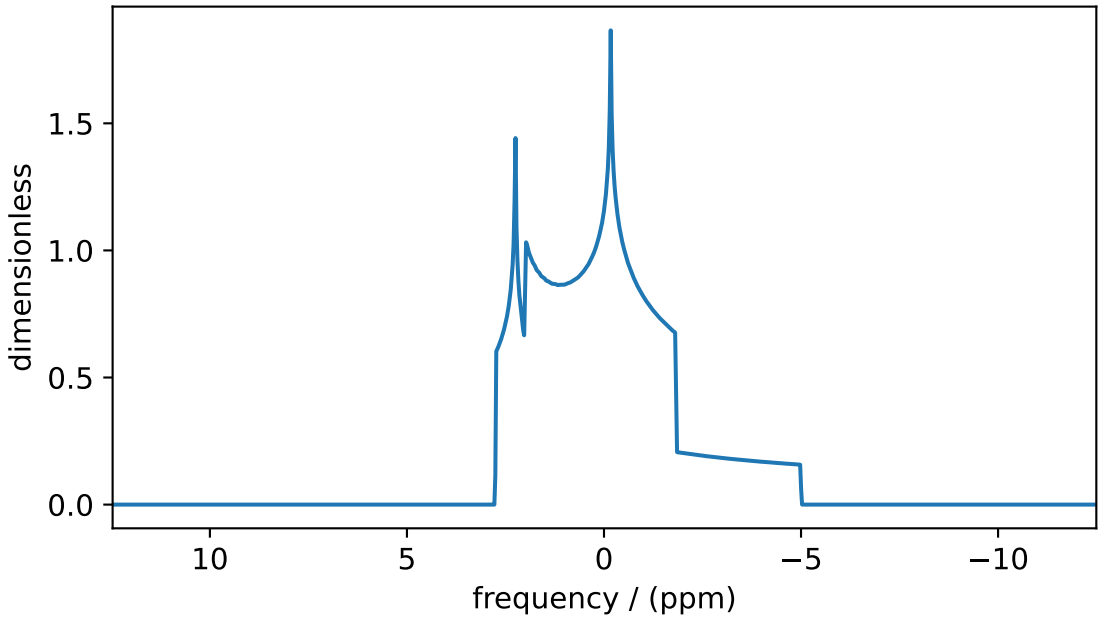


Figure 6.5: The spectrum is an integration of the spectra from individual spin systems when the value of *decompose_spectrum* is `none`.

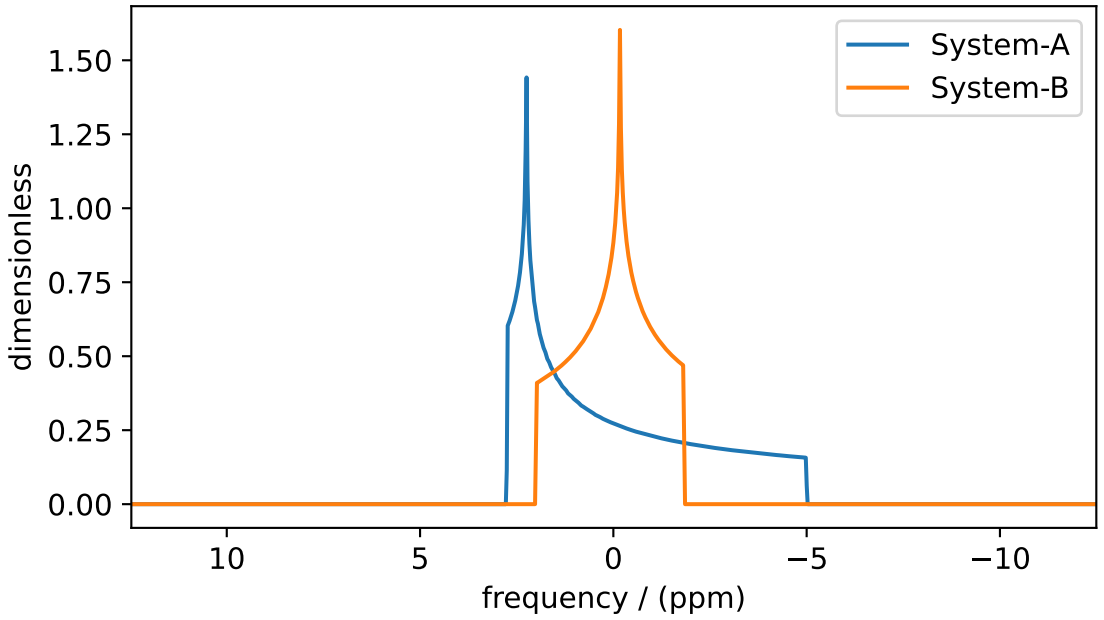


Figure 6.6: Spectrum from individual spin systems when the value of the *decompose_spectrum* config is `spin_system`.

CHAPTER

SEVEN

MRSIMULATOR I/O

7.1 Simulator object

Export simulator object to a JSON file

To serialize a *Simulator* (page 289) object to a JSON-compliant file, use the *save()* (page 295) method of the object.

```
>>> sim_coesite.save('sample.mrsim')
```

where `sim_coesite` is a *Simulator* (page 289) object. By default, the attribute values are serialized as physical quantities, represented as a string with a value and a unit.

Load simulator object from a JSON file

To load a JSON-compliant *Simulator* (page 289) serialized file, use the *load()* (page 295) method of the class. By default, the load method parses the file for units.

```
>>> from mrsimulator import Simulator
>>> sim_load = Simulator.load('sample.mrsim')
>>> sim_coesite == sim_load
True
```

7.2 Spin systems objects from Simulator class

Export spin systems to a JSON file

You may also serialize the spin system objects from the *Simulator* (page 289) object to a JSON-compliant file using the *export_spin_systems()* (page 294) method as

```
>>> sim_coesite.export_spin_systems('coesite_spin_systems.mrsys')
```

Import spin systems from a JSON file

Similarly, a list of spin systems can be directly imported from a JSON serialized file. To import the spin systems, use the *load_spin_systems()* (page 293) method of the *Simulator* (page 289) class as

```
>>> sim.load_spin_systems('coesite_spin_systems.mrsys')
```

Importing spin systems from URL

```
>>> from mrsimulator import Simulator
>>> sim = Simulator()
>>> filename = 'https://raw.githubusercontent.com/DeepanshS/mrsimulator-examples/master/spin_
↪systems.json'
>>> sim.load_spin_systems(filename)
>>> # The seven spin systems from the file are added to the sim object.
>>> len(sim.spin_systems)
7
```

7.3 Method objects from Simulator class

Export methods to a JSON file

Similarly, you may also serialize the method objects from the *Simulator* (page 289) object to a JSON-compliant file using *export_methods()* (page 294) as

```
>>> sim_coesite.export_methods('coesite_method.mrmtd')
```

Import methods from a JSON file

Likewise, to import methods, use *load_methods()* (page 294) as

```
>>> sim.load_methods('coesite_method.mrmtd')
```

7.4 Serialize simulation object from Method class as CSDM compliant file

Export simulation to a JSON file

You may serialize the simulation object from the method object to a CSDM compliant JSON file using the save function as follows,

```
>>> sim_coesite.method[0].simulation.save('coesite_simulation.csdf')
```

7.5 Serialize Simulator, SignalProcessor object to file

Export Simulator, SignalProcessor objects to a JSON file

You may serialize the Simulator, a list of SignalProcessor objects to a *.mrsim* file as follows. The order of SignalProcessor objects is the order of the methods in the Simulator object.

```
>>> from mrsimulator import save
>>> save('coesite.mrsim', sim_coesite, processors)
```

Load Simulator, SignalProcessor objects from a JSON file

```
>>> from mrsimulator import load
>>> sim_coesite, processors, _ = load('coesite.mrsim')
```

Part II

Signal Processing
(mrsimulator.SignalProcessor)

CHAPTER

EIGHT

SIGNAL PROCESSING

8.1 Introduction

After running a simulation, you may need to apply some post-simulation signal processing. For example, you may want to scale the intensities to match the experiment or convolve the spectrum with a Lorentzian, Gaussian, or sinc line-broadening functions. There are many signal-processing libraries, such as Numpy and Scipy, that you may use to accomplish this. Although, in NMR, certain operations like convolutions, Fourier transform, and apodizations are so regularly used that it soon becomes inconvenient to have to write your own set of code. For this reason, the `mrsimulator` package offers some frequently used NMR signal processing tools.

Note: The simulation object in *mrsimulator* is a CSDM object. A CSDM object is the python support for the core scientific dataset model (CSDM)[1], which is a new open-source universal file format for multi-dimensional datasets. Since CSDM objects hold a generic multi-dimensional scientific dataset, the following signal processing operation can be applied to any CSDM dataset, *i.e.*, NMR, EPR, FTIR, GC, etc.

In the following section, we demonstrate the use of the *SignalProcessor* (page 343) class in applying various operations to a generic CSDM object. But before we start explaining signal processing with CSDM objects, it seems necessary to first describe the construct of CSDM objects. Each CSDM object has two main attributes, *dimensions* and *dependent_variables*. The *dimensions* attribute holds a list of Dimension objects, which collectively form a multi-dimensional Cartesian coordinates grid system. A Dimension object can represent both physical and non-physical dimensions. The *dependent_variables* attribute holds the responses of the multi-dimensional grid points. You may have as many dependent variables as you like, as long as all dependent variables share the same coordinates grid, *i.e.*, dimensions.

8.2 SignalProcessor class

Signal processing is a series of operations that are applied to the dataset. In this workflow, the result from the previous operation becomes the input for the next operation.

In the `mrsimulator` library, all signal processing operations are accessed through the *signal_processing* module. Within the module is the *apodization* sub-module. An apodization is a point-wise multiplication operation of the input signal with the apodizing vector. See *Operations* (page 344) documentation for a complete list of operations.

Import the module as

```
>>> from mrsimulator import signal_processing as sp
```

[1] Srivastava, D. J., Vosegaard, T., Massiot, D., Grandinetti, P. J., Core Scientific Dataset Model: A lightweight and portable model and file format for multi-dimensional scientific data, PLOS ONE, **15**, 1-38, (2020). DOI:10.1371/journal.pone.0225953

8.3 Convolution

The convolution theorem states that under suitable conditions, the Fourier transform of a convolution of two signals is the pointwise product (apodization) of their Fourier transforms. In the following example, we employ this theorem to demonstrate how to apply a Gaussian convoluting to a dataset.

```
>>> processor = sp.SignalProcessor(
...     operations=[
...         sp.IFFT(), sp.apodization.Gaussian(FWHM='0.1 km'), sp.FFT()
...     ]
... )
```

Here, the *processor* is an instance of the *SignalProcessor* (page 343) class. The required attribute of this class, *operations*, is a list of operations. In the above example, we employ the convolution theorem by sandwiching the Gaussian apodization function between two Fourier transformations.

In this scheme, first, an inverse Fourier transform is applied to the datasets. On the resulting dataset, a Gaussian apodization, equivalent to a full width at half maximum of 0.1 km in the reciprocal dimension, is applied. The unit that you use for the FWHM attribute depends on the dimensionality of the dataset dimension. By choosing the unit as km, we imply that the corresponding dimension of the CSDM object undergoing the above series of operations has a dimensionality of length. Finally, a forward Fourier transform is applied to the apodized dataset.

Let's create a CSDM object and then apply the above signal processing operations.

```
>>> import csdmpy as cp
>>> import numpy as np
...
>>> # Creating a test CSDM object.
>>> test_data = np.zeros(500)
>>> test_data[250] = 1
>>> csdm_object = cp.CSDM(
...     dependent_variables=[cp.as_dependent_variable(test_data)],
...     dimensions=[cp.LinearDimension(count=500, increment='1 m')]
... )
```

Note: See csdmpy for a detailed description of generating CSDM objects. In `mrsimulator`, the simulation data is already stored as a CSDM object.

To apply the previously defined signal processing operations to the above CSDM object, use the *apply_operations()* (page 343) method of the `SignalProcessor` instance as follows,

```
>>> processed_data = processor.apply_operations(data=csdm_object)
```

The *data* is the required argument of the *apply_ operations* method, whose value is a CSDM object holding the dataset. The variable *processed_ data* holds the output, that is, the processed data as a CSDM object. The plot of the original and the processed data is shown below.

```
>>> import matplotlib.pyplot as plt
>>> _, ax = plt.subplots(1, 2, figsize=(8, 3), subplot_kw={"projection":"csdm"})
>>> ax[0].plot(csdm_object, color="black", linewidth=1)
>>> ax[0].set_title('Before')
>>> ax[1].plot(processed_data.real, color="black", linewidth=1)
>>> ax[1].set_title('After')
```

(continues on next page)

(continued from previous page)

```
>>> plt.tight_layout()
>>> plt.show()
```

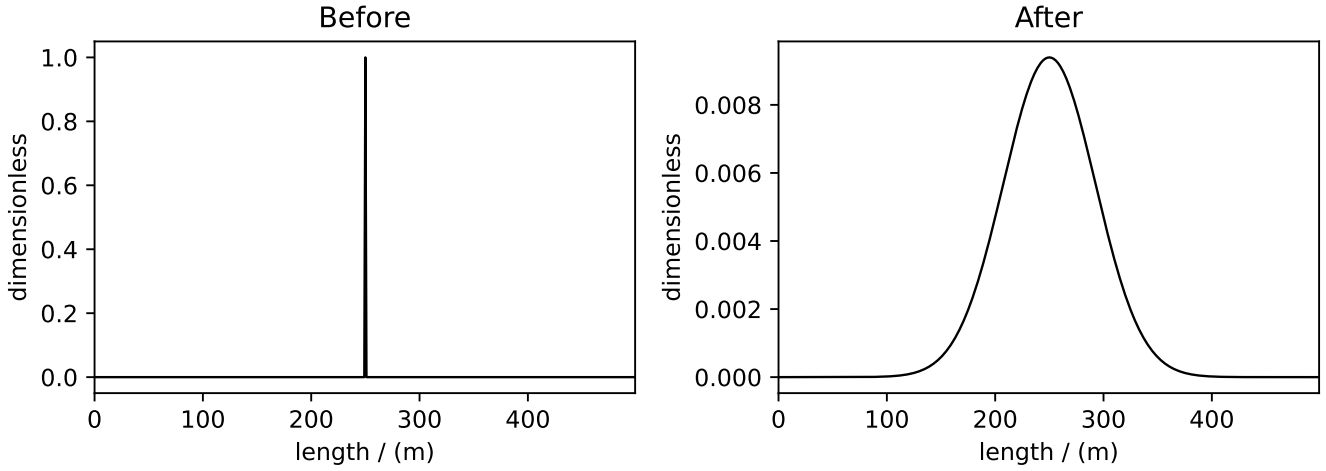


Figure 8.1: The figure depicts an application of Gaussian convolution on a CSDM object.

8.3.1 Multiple convolutions

As mentioned before, a CSDM object may hold multiple dependent variables. When using the list of the operations, you may selectively apply a given operation to a specific dependent-variable by specifying the index of the corresponding dependent-variable as an argument to the operation class. Consider the following list of operations.

```
>>> processor = sp.SignalProcessor(
...     operations=[
...         sp.IFFT(),
...         sp.apodization.Gaussian(FWHM='0.1 km', dv_index=0),
...         sp.apodization.Exponential(FWHM='50 m', dv_index=1),
...         sp.FFT(),
...     ]
... )
```

The above signal processing operations first applies an inverse Fourier transform, followed by a Gaussian apodization on the dependent variable at index 0, followed by an Exponential apodization on the dependent variable at index 1, and finally a forward Fourier transform. Note, the FFT and IFFT operations apply on all dependent-variables.

Let's add another dependent variable to the previously created CSDM object.

```
>>> # Add a dependent variable to the test CSDM object.
>>> test_data = np.zeros(500)
>>> test_data[150] = 1
>>> csdm_object.add_dependent_variable(cp.as_dependent_variable(test_data))
```

As before, apply the operations with the *apply_operations()* (page 343) method.

```
>>> processed_data = processor.apply_operations(data=csdm_object)
```

The plot of the dataset before and after signal processing is shown below.

```
>>> _, ax = plt.subplots(1, 2, figsize=(8, 3), subplot_kw={"projection":"csdm"})
>>> ax[0].plot(csdm_object, linewidth=1)
>>> ax[0].set_title('Before')
>>> ax[1].plot(processed_data.real, linewidth=1)
>>> ax[1].set_title('After')
>>> plt.tight_layout()
>>> plt.show()
```

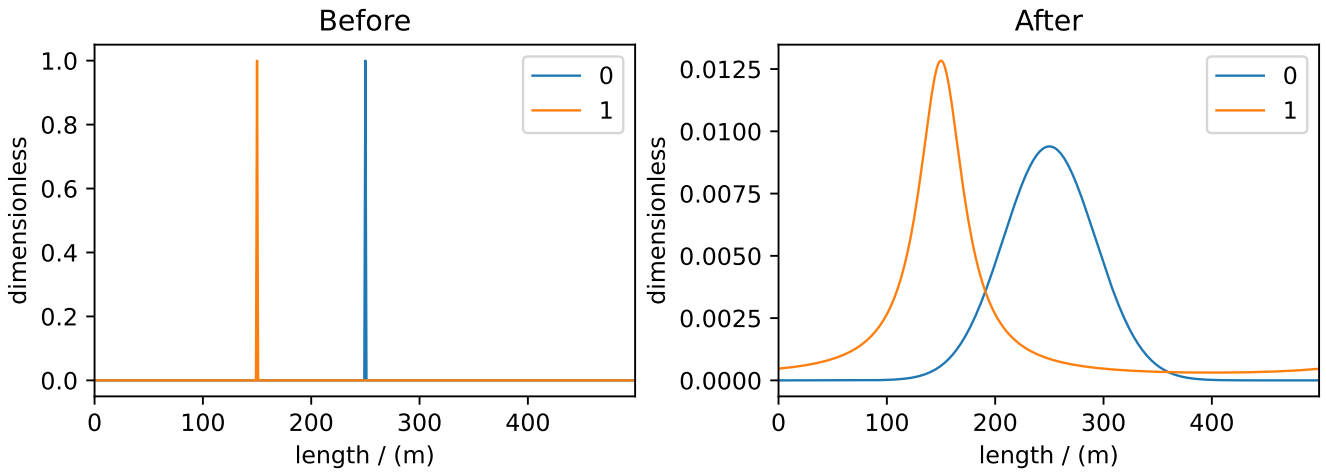


Figure 8.2: Gaussian and Lorentzian convolution applied to two different dependent variables of the CSDM object.

8.3.2 Convolution along multiple dimensions

In the case of multi-dimensional datasets, besides the dependent-variable index, you may also specify a dimension index along which a particular operation will apply. For example, consider the following 2D datasets as a CSDM object,

```
>>> # Create a two-dimensional CSDM object.
>>> test_data = np.zeros(600).reshape(30, 20)
>>> test_data[15, 10] = 1
>>> dv = cp.as_dependent_variable(test_data)
>>> dim1 = cp.LinearDimension(count=20, increment='0.1 ms', coordinates_offset='-1 ms', label='t1
↪')
>>> dim2 = cp.LinearDimension(count=30, increment='1 cm/s', label='s1')
>>> csdm_data = cp.CSDM(dependent_variables=[dv], dimensions=[dim1, dim2])
```

where `csdm_data` is a two-dimensional dataset. Now consider the following signal processing operations

```
>>> processor = sp.SignalProcessor(
...     operations=[
```

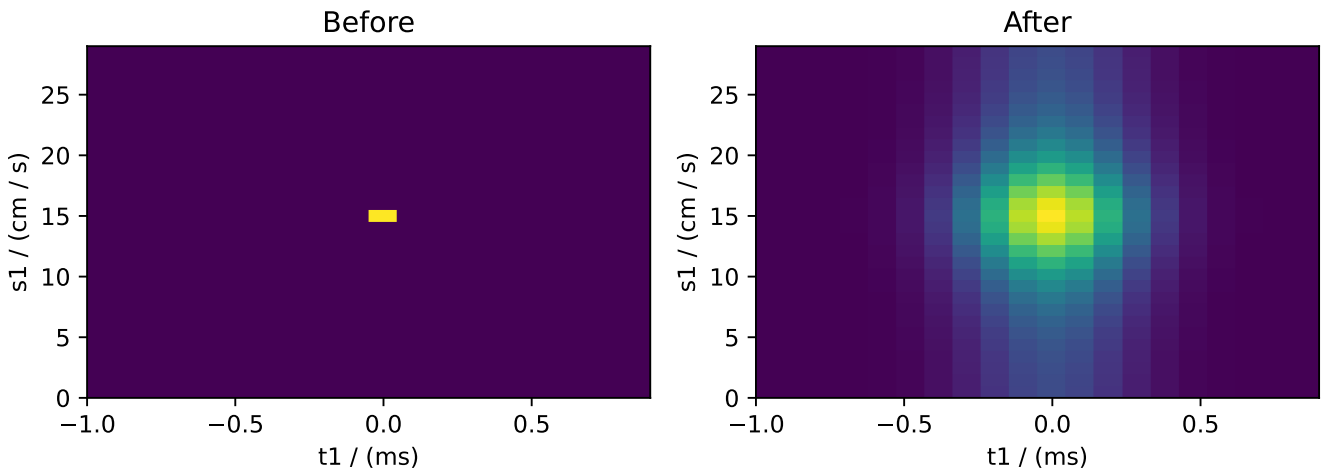
(continues on next page)

(continued from previous page)

```
...         sp.IFFT(dim_index=(0, 1)),
...         sp.apodization.Gaussian(FWHM='0.5 ms', dim_index=0),
...         sp.apodization.Exponential(FWHM='10 cm/s', dim_index=1),
...         sp.FFT(dim_index=(0, 1)),
...     ]
... )
>>> processed_data = processor.apply_operations(data=csdm_data)
```

The above set of operations first performs an inverse FFT on the dataset along the dimension index 0 and 1. The second and third operations apply a Gaussian and Lorentzian apodization along dimensions 0 and 1, respectively. The last operation is a forward Fourier transform. The before and after plots of the datasets are shown below.

```
>>> _, ax = plt.subplots(1, 2, figsize=(8, 3), subplot_kw={"projection":"csdm"})
>>> ax[0].imshow(csdm_data, aspect='auto')
>>> ax[0].set_title('Before')
>>> ax[1].imshow(processed_data.real, aspect='auto')
>>> ax[1].set_title('After')
>>> plt.tight_layout()
>>> plt.show()
```



8.4 Serializing the operations list

You may also serialize the operations list using the *json()* (page 343) method, as follows

```
>>> from pprint import pprint
>>> pprint(processor.json())
{'operations': [{'dim_index': [0, 1], 'function': 'IFFT'},
                {'FWHM': '0.5 ms',
                 'dim_index': 0,
                 'function': 'apodization',
                 'type': 'Gaussian'},
                {'FWHM': '10.0 cm / s',
                 'dim_index': 1,
```

(continues on next page)

(continued from previous page)

```
                'function': 'apodization',
                'type': 'Exponential'},
               {'dim_index': [0, 1], 'function': 'FFT'}]}
```

See also:

Simulation Examples (page 75) for application of signal processing on NMR simulations.

Part III

Models

CHAPTER

NINE

CZJZEK DISTRIBUTION

The Czjzek distribution models random variations of a second-rank traceless symmetric tensors about zero, i.e., a tensor with zeta of zero. See *Czjzek distribution* (page 283) and references within for a brief description of the model.

9.1 Czjzek distribution of symmetric shielding tensors

To generate a Czjzek distribution, use the *CzjzekDistribution* (page 347) class as follows.

```
>>> from mrsimulator.models import CzjzekDistribution
>>> cz_model = CzjzekDistribution(sigma=0.8)
```

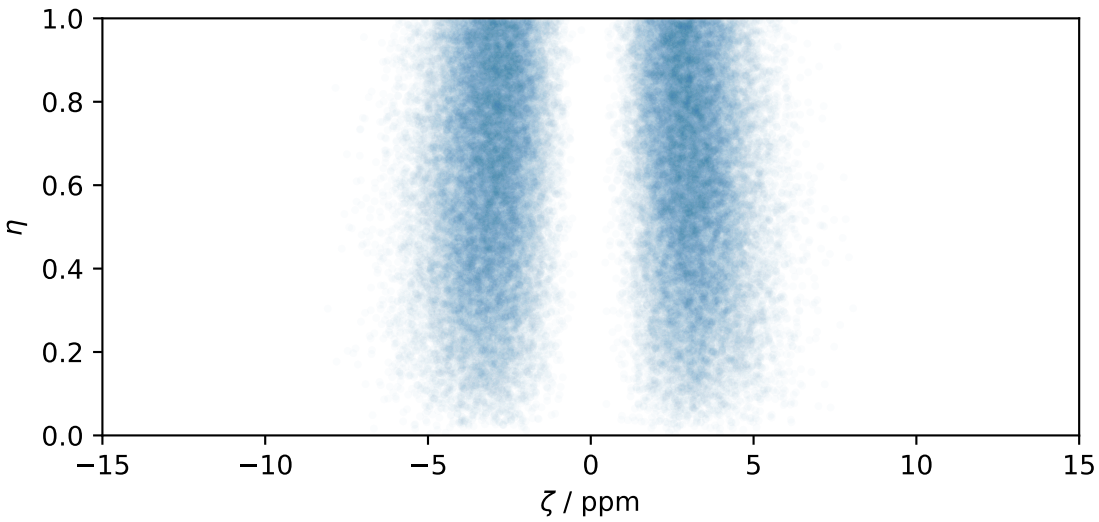
The *CzjzekDistribution* class accepts a single argument, *sigma*, which is the standard deviation of the second-rank traceless symmetric tensor parameters. In the above example, we create `cz_model` as an instance of the CzjzekDistribution class with $\sigma = 0.8$.

Note, *cz_model* is only a class instance of the Czjzek distribution. You can either draw random points from this distribution or generate a probability distribution function. Let's first draw points from this distribution, using the *rvs()* (page 347) method of the instance.

```
>>> zeta_dist, eta_dist = cz_model.rvs(size=50000)
```

In the above example, we draw *size=50000* random points of the distribution. The output `zeta_dist` and `eta_dist` hold the tensor parameter coordinates of the points, defined in the Haeberlen convention. The scatter plot of these coordinates is shown below.

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(zeta_dist, eta_dist, s=4, alpha=0.02)
>>> plt.xlabel('$\zeta$ / ppm')
>>> plt.ylabel('$\eta$')
>>> plt.xlim(-15, 15)
>>> plt.ylim(0, 1)
>>> plt.tight_layout()
>>> plt.show()
```



9.2 Czjzek distribution of symmetric quadrupolar tensors

The Czjzek distribution of symmetric quadrupolar tensors follows a similar setup as the Czjzek distribution of symmetric shielding tensors, except we assign the outputs to Cq and η_q. In the following example, we generate the probability distribution function using the *pdf()* (page 348) method.

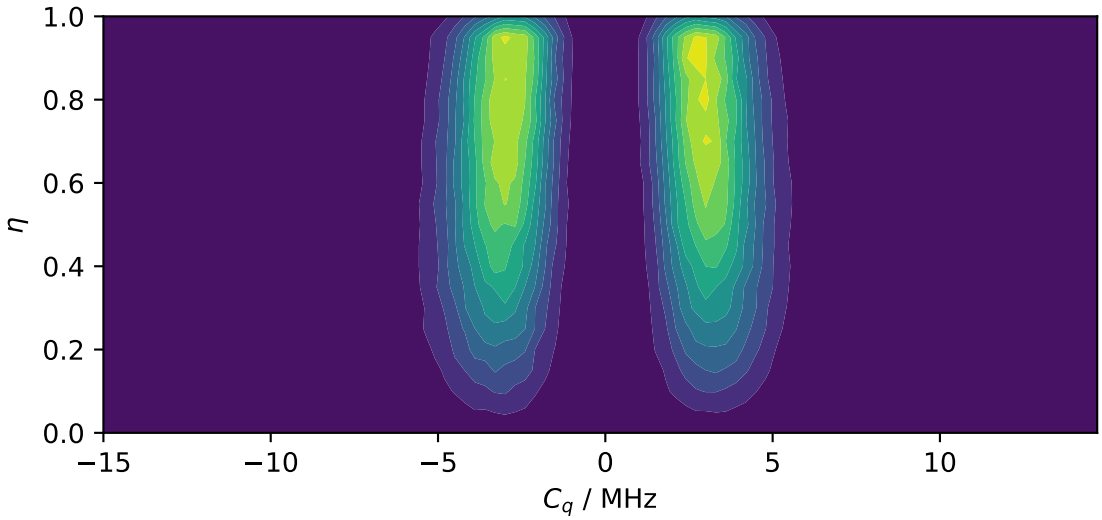
```
>>> import numpy as np
>>> Cq_range = np.arange(100)*0.3 - 15 # pre-defined Cq range in MHz.
>>> eta_range = np.arange(21)/20  # pre-defined eta range.
>>> Cq, eta, amp = cz_model.pdf(pos=[Cq_range, eta_range])
```

To generate a probability distribution, we need to define a grid system over which the distribution probabilities will be evaluated. We do so by defining the range of coordinates along the two dimensions. In the above example, `Cq_range` and `eta_range` are the range of Cq and η_q coordinates, which is then given as the argument to the *pdf()* (page 348) method. The output `Cq`, `eta`, and `amp` hold the two coordinates and amplitude, respectively.

The plot of the Czjzek probability distribution is shown below.

```
>>> import matplotlib.pyplot as plt
>>> plt.contourf(Cq, eta, amp, levels=10)
>>> plt.xlabel('$C_q$ / MHz')
>>> plt.ylabel('$\eta$')
>>> plt.tight_layout()
>>> plt.show()
```

Note: The `pdf` method of the instance generates the probability distribution function by first drawing random points from the distribution and then binning it onto a pre-defined grid.



9.3 Mini-gallery using the Czjzek distributions

- *Czjzek distribution (Shielding and Quadrupolar)* (page 113)
- *Czjzek distribution, ^{27}Al (I=5/2) 3QMAS* (page 160)

CHAPTER

TEN

EXTENDED CZJZEK DISTRIBUTION

The Extended Czjzek distribution models random variations of a second-rank traceless symmetric tensors about a non-zero tensor. See *Extended Czjzek distribution* (page 284) and references within for a brief description of the model.

10.1 Extended Czjzek distribution of symmetric shielding tensors

To generate an extended Czjzek distribution, use the *ExtCzjzekDistribution* (page 348) class as follows.

```
>>> from mrsimulator.models import ExtCzjzekDistribution
>>> shielding_tensor = {'zeta': 80, 'eta': 0.4}
>>> shielding_model = ExtCzjzekDistribution(shielding_tensor, eps=0.1)
```

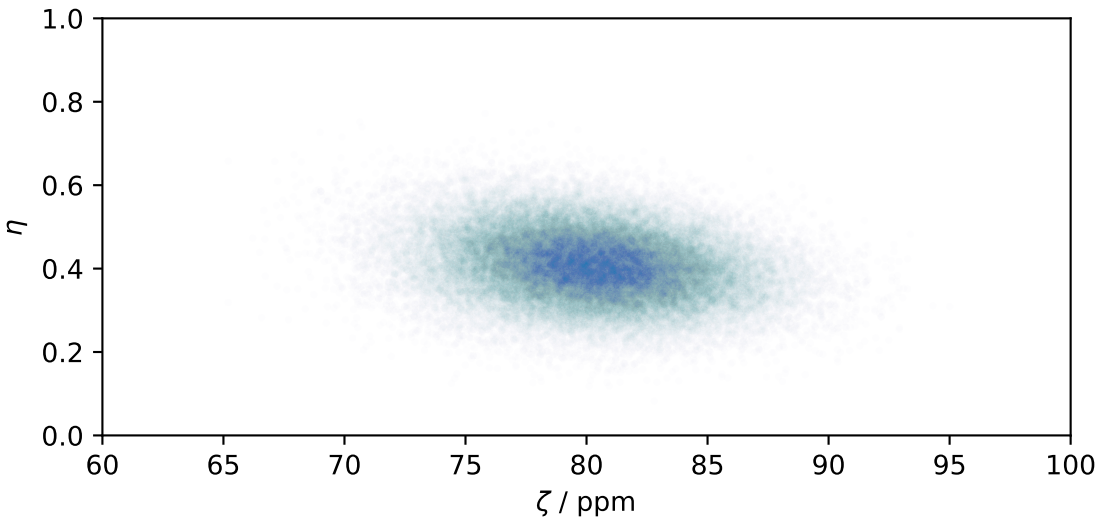
The *ExtCzjzekDistribution* class accepts two arguments. The first argument is the dominant tensor about which the perturbation applies, and the second parameter, *eps*, is the perturbation fraction. The minimum value of the *eps* parameter is 0, which means the distribution is a delta function at the dominant tensor parameters. As the value of *eps* increases, the distribution gets broader. At *eps* values greater than 1, the extended Czjzek distribution approaches a Czjzek distribution. In the above example, we create an extended Czjzek distribution about a second-rank traceless symmetric shielding tensor described by anisotropy of 80 ppm and an asymmetry parameter of 0.4. The perturbation fraction is 0.1.

As before, you may either draw random samples from this distribution or generate a probability distribution function. Let's first draw points from this distribution, using the *rvs()* (page 349) method of the instance.

```
>>> zeta_dist, eta_dist = shielding_model.rvs(size=50000)
```

In the above example, we draw *size=50000* random points of the distribution. The output `zeta_dist` and `eta_dist` hold the tensor parameter coordinates of the points, defined in the Haeberlen convention. The scatter plot of these coordinates is shown below.

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(zeta_dist, eta_dist, s=4, alpha=0.01)
>>> plt.xlabel('$\zeta$ / ppm')
>>> plt.ylabel('$\eta$')
>>> plt.xlim(60, 100)
>>> plt.ylim(0, 1)
>>> plt.tight_layout()
>>> plt.show()
```



10.2 Extended Czjzek distribution of symmetric quadrupolar tensors

The extended Czjzek distribution of symmetric quadrupolar tensors follows a similar setup as the extended Czjzek distribution of symmetric shielding tensors, shown above. In the following example, we generate the probability distribution function using the *pdf()* (page 349) method.

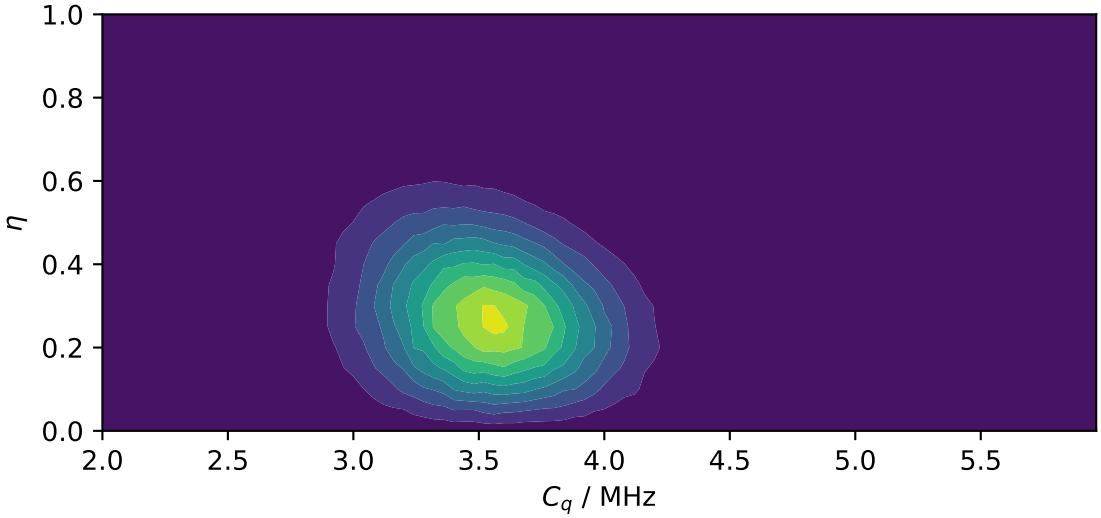
```
>>> import numpy as np
>>> Cq_range = np.arange(100)*0.04 + 2 # pre-defined Cq range in MHz.
>>> eta_range = np.arange(21)/20  # pre-defined eta range.
...
>>> quad_tensor = {'Cq': 3.5, 'eta': 0.23} # Cq assumed in MHz
>>> model_quad = ExtCzjzekDistribution(quad_tensor, eps=0.2)
>>> Cq, eta, amp = model_quad.pdf(pos=[Cq_range, eta_range])
```

As with the case with Czjzek distribution, to generate a probability distribution of the extended Czjzek distribution, we need to define a grid system over which the distribution probabilities will be evaluated. We do so by defining the range of coordinates along the two dimensions. In the above example, `Cq_range` and `eta_range` are the range of Cq and η_q coordinates, which is then given as the argument to the *pdf()* (page 349) method. The output `Cq`, `eta`, and `amp` hold the two coordinates and amplitude, respectively.

The plot of the extended Czjzek probability distribution is shown below.

```
>>> import matplotlib.pyplot as plt
>>> plt.contourf(Cq, eta, amp, levels=10)
>>> plt.xlabel('$C_q$ / MHz')
>>> plt.ylabel('$\eta$')
>>> plt.tight_layout()
>>> plt.show()
```

Note: The `pdf` method of the instance generates the probability distribution function by first drawing random points from the distribution and then binning it onto a pre-defined grid.



10.3 Mini-gallery using the extended Czjzek distributions

- *Extended Czjzek distribution (Shielding and Quadrupolar)* (page 117)
- *Simulating site disorder (crystalline)* (page 156)

Part IV

Examples and Benchmarks

CHAPTER

ELEVEN

SIMULATION EXAMPLES

In this section, we use the `mrsimulator` tools to create spin systems and simulate spectrum with practical/experimental applications. The examples illustrate

- building spin systems (uncoupled and weakly-coupled),
- building NMR methods,
- simulating spectrum, and
- processing spectrum (e.g. adding line-broadening).

For mrsimulator applications related to least-squares fitting, see the *Fitting Examples (Least Squares)* (page 165).

11.1 1D NMR simulation (small molecules/crystalline solids)

The following examples are the NMR spectrum simulation of small molecules and crystalline solids for the following methods:

- Bloch decay method (*BlochDecaySpectrum* (page 320)),
- Central transition selective Bloch decay method (*BlochDecayCTSpectrum* (page 321)).
- Generic one-dimensional method (*Method1D* (page 319)).

11.1.1 Wollastonite, ^{29}Si (I=1/2)

^{29}Si (I=1/2) spinning sideband simulation.

Wollastonite is a high-temperature calcium-silicate, β-$Ca_3Si_3O_9$, with three distinct ^{29}Si sites. The ^{29}Si tensor parameters were obtained from Hansen *et al.*[1]

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
```

Step 1: Create the sites.

```
S29_1 = Site(
    isotope="29Si",
    isotropic_chemical_shift=-89.0,  # in ppm
    shielding_symmetric={"zeta": 59.8, "eta": 0.62},  # zeta in ppm
```

(continues on next page)

[1] Hansen, M. R., Jakobsen, H. J., Skibsted, J., ^{29}Si Chemical Shift Anisotropies in Calcium Silicates from High-Field ^{29}Si MAS NMR Spectroscopy, Inorg. Chem. 2003, **42**, 7, 2368-2377. DOI: 10.1021/ic020647f

(continued from previous page)

```
)
S29_2 = Site(
    isotope="29Si",
    isotropic_chemical_shift=-89.5,  # in ppm
    shielding_symmetric={"zeta": 52.1, "eta": 0.68},  # zeta in ppm
)
S29_3 = Site(
    isotope="29Si",
    isotropic_chemical_shift=-87.8,  # in ppm
    shielding_symmetric={"zeta": 69.4, "eta": 0.60},  # zeta in ppm
)

sites = [S29_1, S29_2, S29_3]  # all sites
```

Step 2: Create the spin systems from these sites. Again, we create three single-site spin systems for better performance.

```
spin_systems = [SpinSystem(sites=[s]) for s in sites]
```

Step 3: Create a Bloch decay spectrum method.

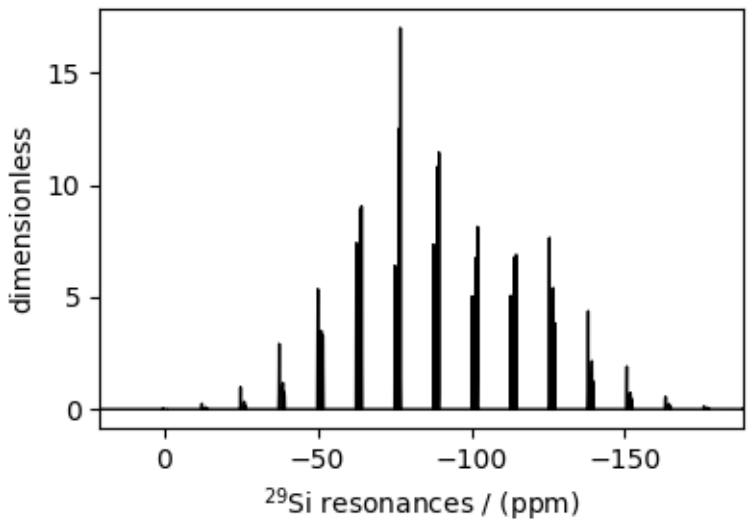
```
method = BlochDecaySpectrum(
    channels=["29Si"],
    magnetic_flux_density=14.1,  # in T
    rotor_frequency=1500,  # in Hz
    spectral_dimensions=[
        {
            "count": 2048,
            "spectral_width": 25000,  # in Hz
            "reference_offset": -10000,  # in Hz
            "label": r"$^{29}$Si resonances",
        }
    ],
)
```

Step 4: Create the Simulator object and add the method and spin system objects.

```
sim = Simulator()
sim.spin_systems += spin_systems  # add the spin systems
sim.methods += [method]  # add the method
```

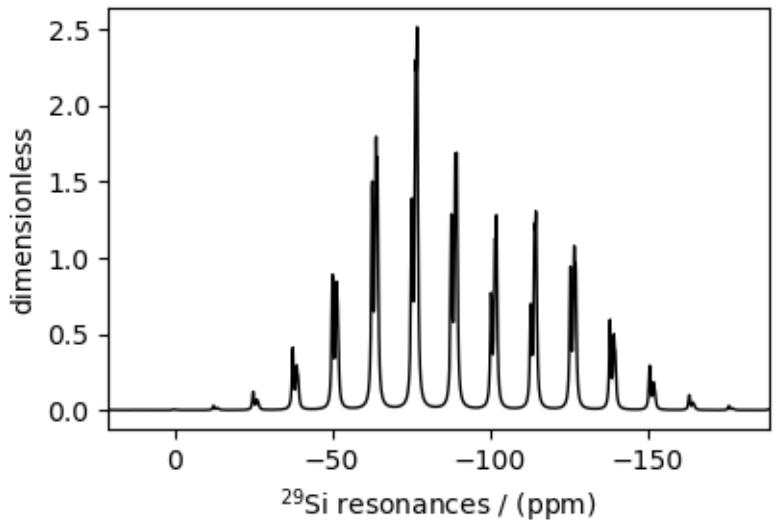
Step 5: Simulate the spectrum.

```
sim.run()

# The plot of the simulation before signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Step 6: Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[sp.IFFT(), sp.apodization.Exponential(FWHM="70 Hz"), sp.FFT()]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation)

# The plot of the simulation after signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(processed_data.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 1.379 seconds)

11.1.2 Potassium Sulfate, ^{33}S (I=3/2)

^{33}S (I=3/2) quadrupolar spectrum simulation.

The following example is the ^{33}S NMR spectrum simulation of potassium sulfate (K_2SO_4). The quadrupole tensor parameters for ^{33}S is obtained from Moudrakovski *et al.*[1]

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecayCTSpectrum
from mrsimulator import signal_processing as sp
```

Step 1: Create the spin system

```
site = Site(
    name="33S",
    isotope="33S",
    isotropic_chemical_shift=335.7,  # in ppm
    quadrupolar={"Cq": 0.959e6, "eta": 0.42},  # Cq is in Hz
)
spin_system = SpinSystem(sites=[site])
```

Step 2: Create a central transition selective Bloch decay spectrum method.

```
method = BlochDecayCTSpectrum(
    channels=["33S"],
    magnetic_flux_density=21.14,  # in T
    rotor_frequency=14000,  # in Hz
    spectral_dimensions=[
        {
            "count": 2048,
            "spectral_width": 5000,  # in Hz
            "reference_offset": 22500,  # in Hz
            "label": r"$^{33}$S resonances",
        }
    ],
)
```

Step 3: Create the Simulator object and add method and spin system objects.

```
sim = Simulator()
sim.spin_systems += [spin_system]  # add the spin system
sim.methods += [method]  # add the method
```

Step 4: Simulate the spectrum.

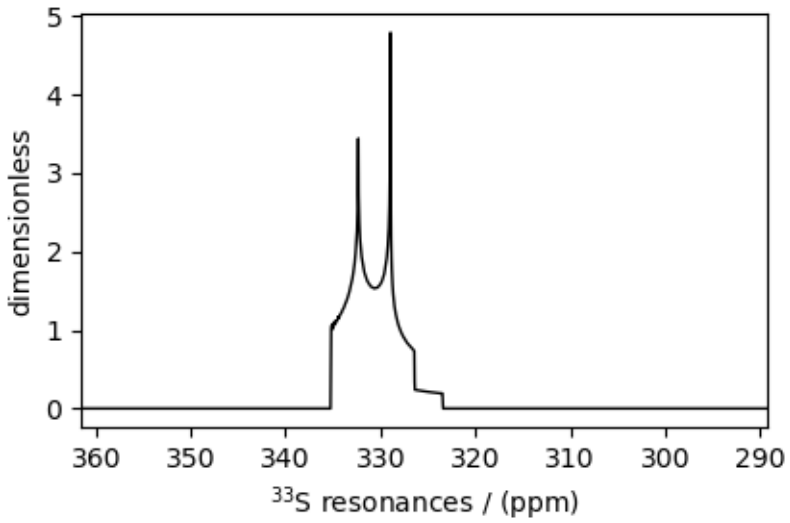
```
sim.run()

# The plot of the simulation before signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
```

(continues on next page)

[1] Moudrakovski, I., Lang, S., Patchkovskii, S., and Ripmeester, J. High field ^{33}S solid state NMR and first-principles calculations in potassium sulfates. J. Phys. Chem. A, 2010, **114**, *1*, 309–316. DOI: 10.1021/jp908206c

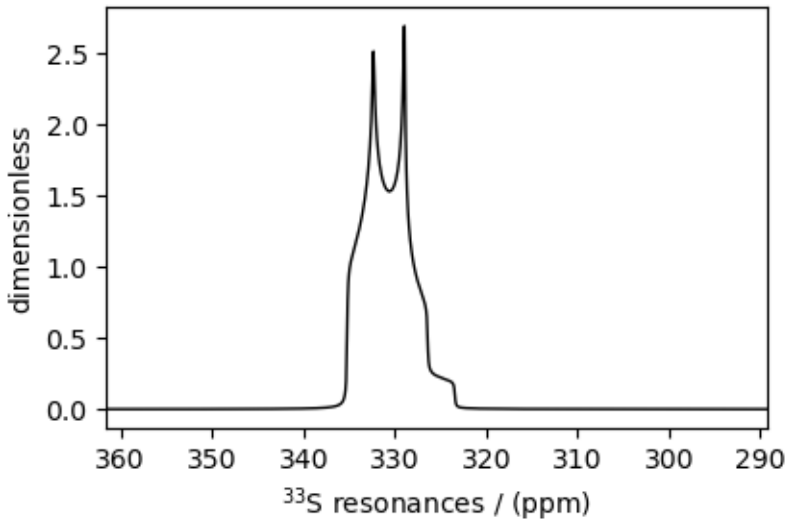
(continued from previous page)

```
ax.plot(sim.methods[0].simulation.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Step 5: Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[sp.IFFT(), sp.apodization.Exponential(FWHM="10 Hz"), sp.FFT()]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation)

# The plot of the simulation after signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(processed_data.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.340 seconds)

11.1.3 Coesite, ^{17}O (I=5/2)

^{17}O (I=5/2) quadrupolar spectrum simulation.

Coesite is a high-pressure (2-3 GPa) and high-temperature (700°C) polymorph of silicon dioxide SiO_2. Coesite has five crystallographic ^{17}O sites. In the following, we use the ^{17}O EFG tensor information from Grandinetti *et al.*[1]

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecayCTSpectrum
from mrsimulator import signal_processing as sp
```

Step 1: Create the sites.

```
# default unit of isotropic_chemical_shift is ppm and Cq is Hz.
O17_1 = Site(
    isotope="17O", isotropic_chemical_shift=29, quadrupolar={"Cq": 6.05e6, "eta": 0.000}
)
O17_2 = Site(
    isotope="17O", isotropic_chemical_shift=41, quadrupolar={"Cq": 5.43e6, "eta": 0.166}
)
O17_3 = Site(
    isotope="17O", isotropic_chemical_shift=57, quadrupolar={"Cq": 5.45e6, "eta": 0.168}
)
O17_4 = Site(
    isotope="17O", isotropic_chemical_shift=53, quadrupolar={"Cq": 5.52e6, "eta": 0.169}
)
O17_5 = Site(
    isotope="17O", isotropic_chemical_shift=58, quadrupolar={"Cq": 5.16e6, "eta": 0.292}
```

(continues on next page)

[1] Grandinetti, P. J., Baltisberger, J. H., Farnan, I., Stebbins, J. F., Werner, U. and Pines, A. Solid-State ^{17}O Magic-Angle and Dynamic-Angle Spinning NMR Study of the SiO_2 Polymorph Coesite, J. Phys. Chem. 1995, **99**, *32*, 12341-12348. DOI: 10.1021/j100032a045

(continued from previous page)

```
)

# all five sites.
sites = [O17_1, O17_2, O17_3, O17_4, O17_5]
```

Step 2: Create the spin systems from these sites. For optimum performance, we create five single-site spin systems instead of a single five-site spin system. The abundance of each spin system is taken from above reference.

```
abundance = [0.83, 1.05, 2.16, 2.05, 1.90]
spin_systems = [SpinSystem(sites=[s], abundance=a) for s, a in zip(sites, abundance)]
```

Step 3: Create a central transition selective Bloch decay spectrum method.

```
method = BlochDecayCTSpectrum(
    channels=["17O"],
    rotor_frequency=14000,  # in Hz
    spectral_dimensions=[
        {
            "count": 2048,
            "spectral_width": 50000,  # in Hz
            "label": r"$^{17}$O resonances",
        }
    ],
)
```

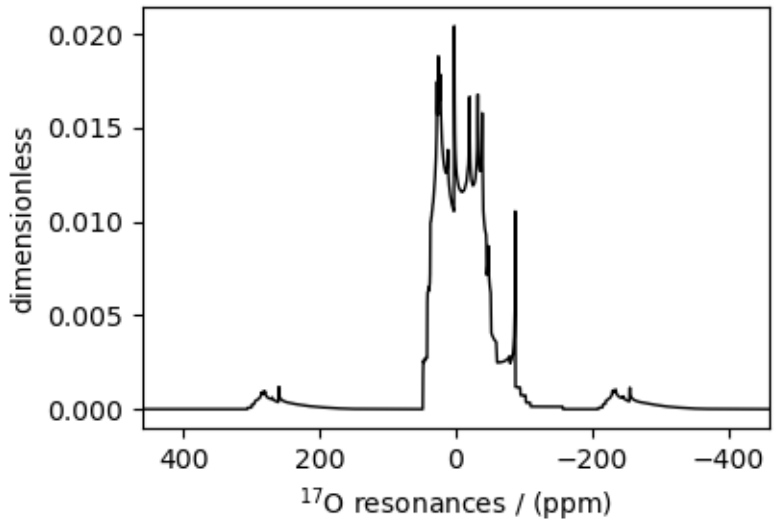
The above method is set up to record the ^{17}O resonances at the magic angle, spinning at 14 kHz and 9.4 T (default, if the value is not provided) external magnetic flux density. The resonances are recorded over 50 kHz spectral width using 2048 points.

Step 4: Create the Simulator object and add the method and spin system objects.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [method]  # add the method
```

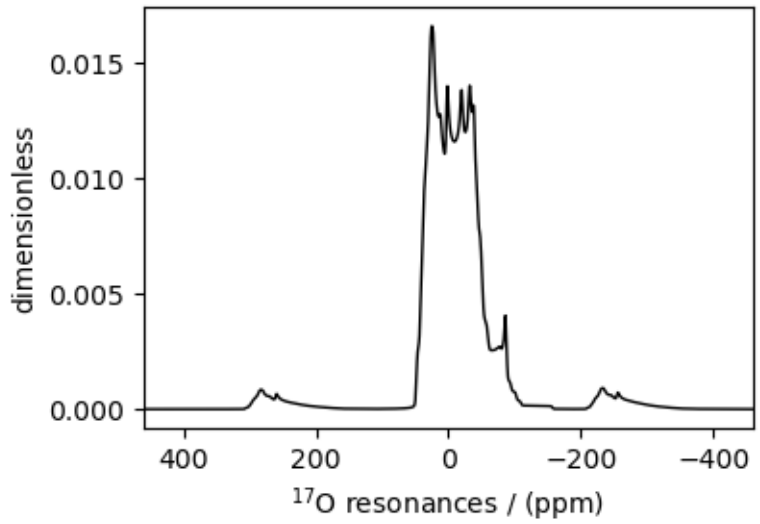
Step 5: Simulate the spectrum.

```
sim.run()

# The plot of the simulation before signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Step 6: Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="30 Hz"),
        sp.apodization.Gaussian(FWHM="145 Hz"),
        sp.FFT(),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation)

# The plot of the simulation after signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(processed_data.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.355 seconds)

11.1.4 Non-coincidental Quad and CSA, ^{17}O (I=5/2)

^{17}O (I=5/2) quadrupolar static spectrum simulation.

The following example illustrates the simulation of NMR spectra arising from non-coincidental quadrupolar and shielding tensors. The tensor parameter values for the simulation are obtained from Yamada *et al.*[1], for the ^{17}O site in benzanilide.

Warning: The Euler angles representation using by Yamada *et al* is different from the representation used in mrsimulator. The resulting simulation might not resemble the published spectrum.

```
import numpy as np
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecayCTSpectrum
```

Step 1: Create the spin system.

```
site = Site(
    isotope="17O",
    isotropic_chemical_shift=320,  # in ppm
    shielding_symmetric={"zeta": 376.667, "eta": 0.345},
    quadrupolar={
        "Cq": 8.97e6,  # in Hz
        "eta": 0.15,
        "alpha": 5 * np.pi / 180,
        "beta": np.pi / 2,
```

(continues on next page)

[1] Yamada, K., Dong, S., Wu, G., Solid-State 17O NMR Investigation of the Carbonyl Oxygen Electric-Field-Gradient Tensor and Chemical Shielding Tensor in Amides, J. Am. Chem. Soc. 2000, **122**, 11602-11609. DOI: 10.1021/ja0008315

(continued from previous page)

```
        "gamma": 70 * np.pi / 180,
    },
)
spin_system = SpinSystem(sites=[site])
```

Step 2: Create a central transition selective Bloch decay spectrum method.

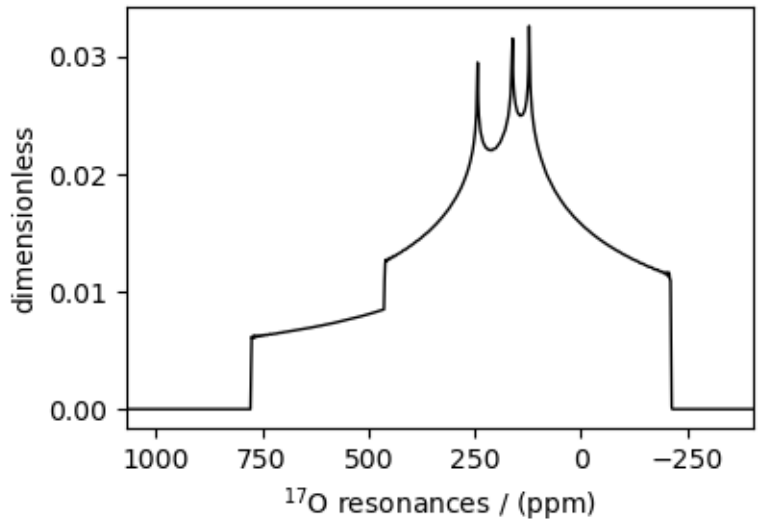
```
method = BlochDecayCTSpectrum(
    channels=["17O"],
    magnetic_flux_density=11.74,  # in T
    rotor_frequency=0,  # in Hz
    spectral_dimensions=[
        {
            "count": 1024,
            "spectral_width": 1e5,  # in Hz
            "reference_offset": 22500,  # in Hz
            "label": r"$^{17}$O resonances",
        }
    ],
)
```

Step 3: Create the Simulator object and add method and spin system objects.

```
sim = Simulator()
sim.spin_systems = [spin_system]  # add the spin system
sim.methods = [method]  # add the method

# Since the spin system have non-zero Euler angles, set the integration_volume to
# hemisphere.
sim.config.integration_volume = "hemisphere"
```

Step 4: Simulate the spectrum.

```
sim.run()

# The plot of the simulation before signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.169 seconds)

11.1.5 Arbitrary spin transition (single-quantum)

^{27}Al (I=5/2) quadrupolar spectrum simulation.

The mrsimulator built-in one-dimensional methods, BlochDecaySpectrum and BlochDecayCTSpectrum, are designed to simulate spectrum from all single quantum transitions or central transition selective transition, respectively. In this example, we show how you can simulate any arbitrary transition using the generic Method1D method.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import Method1D
```

Create a single-site arbitrary spin system.

```
site = Site(
    name="27Al",
    isotope="27Al",
    isotropic_chemical_shift=35.7,  # in ppm
    quadrupolar={"Cq": 5.959e6, "eta": 0.32},  # Cq is in Hz
)
spin_system = SpinSystem(sites=[site])
```

Selecting spin transitions for simulation

The arguments of the Method1D object are the same as the arguments of the BlochDecaySpectrum method; however, unlike a BlochDecaySpectrum method, the *SpectralDimension* (page 313) object in Method1D contains additional argument—*events*.

The *Event* (page 315) object is a collection of attributes, which are local to the event. It is here where we define a *transition_ query* to select one or more transitions for simulating the spectrum. The two attributes of the *transition_ query* are P and D, which are given as,

$$\begin{aligned} P &= m_f - m_i \\ D &= m_f^2 - m_i^2, \end{aligned} \tag{11.1}$$

where m_f and m_i are the spin quantum numbers for the final and initial energy states. Based on the query, the method selects all transitions from the spin system that satisfy the query selection criterion. For example, to simulate a spectrum for the satellite transition, $|-1/2\rangle \rightarrow |-3/2\rangle$, set the value of

$$\begin{aligned} P &= \left(-\frac{3}{2}\right) - \left(-\frac{1}{2}\right) = -1 \\ D &= \frac{9}{4} - \frac{1}{4} = 2. \end{aligned} \tag{11.2}$$

For illustrative purposes, let's look at the infinite speed spectrum from this satellite transition.

```
method = Method1D(
    channels=["27Al"],
    magnetic_flux_density=21.14,  # in T
    rotor_frequency=1e9,  # in Hz
    spectral_dimensions=[
        {
            "count": 1024,
            "spectral_width": 1e4,  # in Hz
            "reference_offset": 1e4,  # in Hz
            "events": [
                {"transition_query": {"P": [-1], "D": [2]}}  # <-- select transitions
            ],
        }
    ],
)
```

Create the Simulator object and add the method and the spin system object.

```
sim = Simulator()
sim.spin_systems += [spin_system]  # add the spin system
sim.methods += [method]  # add the method
```

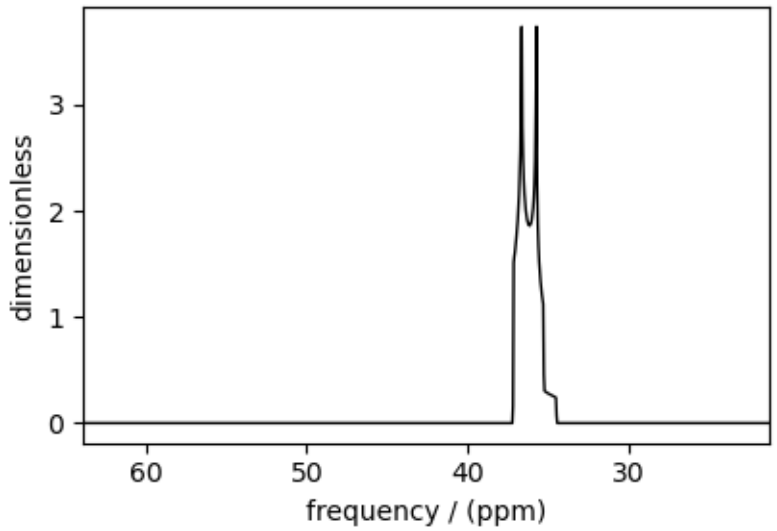
Simulate the spectrum.

```
sim.run()

# The plot of the simulation before signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation.real, color="black", linewidth=1)
ax.invert_xaxis()
```

(continues on next page)

(continued from previous page)

```
plt.tight_layout()
plt.show()
```



Selecting both inner and outer-satellite transitions

You may use the same transition query selection criterion to select multiple transitions. Consider the following transitions with respective P and D values.

- $|-1/2\rangle \rightarrow |-3/2\rangle$ $(P = -1, D = 2)$
- $|-3/2\rangle \rightarrow |-5/2\rangle$ $(P = -1, D = 4)$

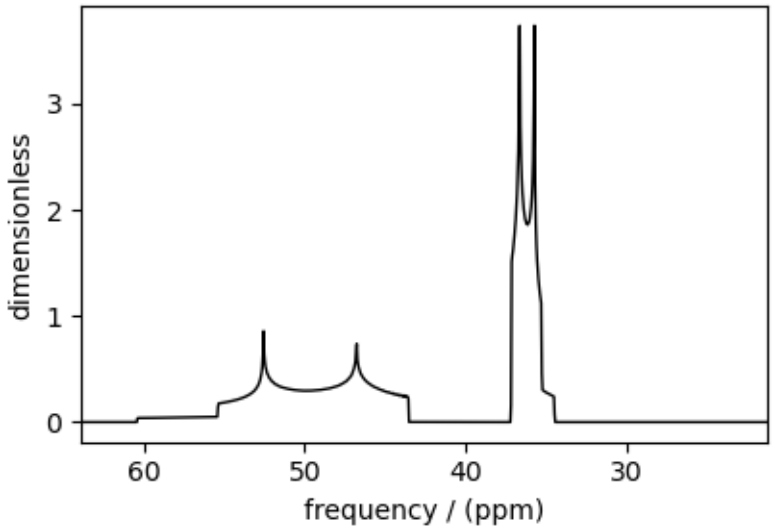
```
method2 = Method1D(
    channels=["27Al"],
    magnetic_flux_density=21.14,  # in T
    rotor_frequency=1e9,  # in Hz
    spectral_dimensions=[
        {
            "count": 1024,
            "spectral_width": 1e4,  # in Hz
            "reference_offset": 1e4,  # in Hz
            "events": [
                {"transition_query": {"P": [-1], "D": [2, 4]}}  # <-- select transitions
            ],
        }
    ],
)
```

Update the method object in the Simulator object.

```
sim.methods[0] = method2  # add the method
```

Simulate the spectrum.

```
sim.run()

# The plot of the simulation before signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.292 seconds)

11.1.6 Arbitrary spin transition (multi-quantum)

^{33}S (I=5/2) quadrupolar spectrum simulation.

Simulate a triple quantum spectrum.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import Method1D
```

Create a single-site arbitrary spin system.

```
site = Site(
    name="27Al",
    isotope="27Al",
    isotropic_chemical_shift=35.7,  # in ppm
    quadrupolar={"Cq": 2.959e6, "eta": 0.98},  # Cq is in Hz
)
spin_system = SpinSystem(sites=[site])
```

Selecting the triple-quantum transition

For spin-site spin-5/2 spin system, there are three triple-quantum transition

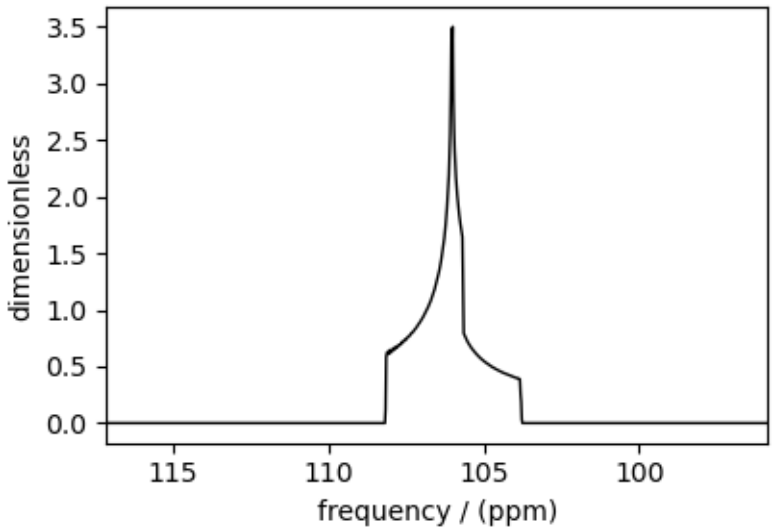
- $|1/2\rangle \rightarrow |-5/2\rangle$ $(P = -3, D = 6)$
- $|3/2\rangle \rightarrow |-3/2\rangle$ $(P = -3, D = 0)$
- $|5/2\rangle \rightarrow |-1/2\rangle$ $(P = -3, D = -6)$

To select one or more triple-quantum transitions, assign the respective value of P and D to the *transition_ query*.

```
method = Method1D(
    channels=["27Al"],
    magnetic_flux_density=21.14,  # in T
    rotor_frequency=1e9,  # in Hz
    spectral_dimensions=[
        {
            "count": 1024,
            "spectral_width": 5e3,  # in Hz
            "reference_offset": 2.5e4,  # in Hz
            "events": [
                {  # symmetric triple quantum transitions
                    "transition_query": {"P": [-3], "D": [0]}
                }
            ],
        }
    ],
)
```

Create the Simulator object and add the method and the spin system object.

```
sim = Simulator()
sim.spin_systems += [spin_system]  # add the spin system
sim.methods += [method]  # add the method
sim.run()

# The plot of the simulation before signal processing.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.155 seconds)

11.1.7 Coupled spin-1/2 (Static dipolar spectrum)

^{13}C-^{1}H static dipolar coupling simulation.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
```

Spin Systems

Create a 13C-1H coupled spin system.

```
spin_system = SpinSystem(
    sites=[
        {"isotope": "13C", "isotropic_chemical_shift": 0.0},
        {"isotope": "1H", "isotropic_chemical_shift": 0.0},
    ],
    couplings=[{"site_index": [0, 1], "dipolar": {"D": -2e4}}],
)
```

Methods

Create a BlochDecaySpectrum method.

```
method = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=9.4,  # in T
    spectral_dimensions=[{"count": 2048, "spectral_width": 8.0e4}],
)
```

Simulator

Create the Simulator object and add the method and the spin system object.

```
sim = Simulator()
sim.spin_systems += [spin_system]  # add the spin system.
sim.methods += [method]  # add the method.
sim.run()
```

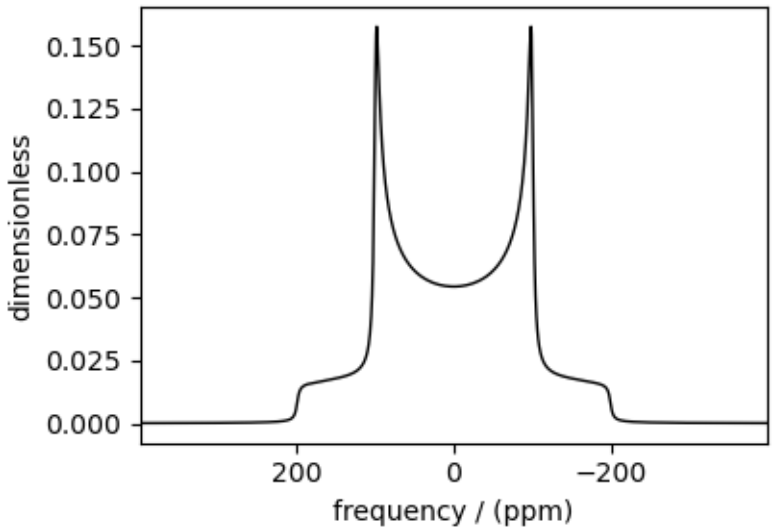
Post-Simulation Processing

Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="500 Hz"),
        sp.FFT(),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation)
```

Plot

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(processed_data.real, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.139 seconds)

11.1.8 Coupled spins 5/2-9/2 (Quad + J-coupling)

^{27}Al-^{93}Nb spin system spectrum.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem
from mrsimulator.methods import BlochDecayCTSpectrum
from mrsimulator import signal_processing as sp
```

Spin System

Create a 27Al-93Nb coupled spin system.

```
spin_system = SpinSystem(
    sites=[
        {
            "isotope": "27Al",
            "isotropic_chemical_shift": 0.0,  # in ppm
            "quadrupolar": {"Cq": 5.0e6, "eta": 0.0},  # Cq is in Hz
        },
        {
            "isotope": "93Nb",
            "isotropic_chemical_shift": 0.0,  # in ppm
        },
    ],
    couplings=[{"site_index": [0, 1], "isotropic_j": 200.0}],  # j-coupling in Hz
)
```

Method

Create a central transition selective Bloch decay spectrum method.

```
method = BlochDecayCTSpectrum(
    channels=["27Al"],
    magnetic_flux_density=9.4,  # in T
    rotor_frequency=5e3,  # in Hz
    spectral_dimensions=[
        {
            "count": 2048,
            "spectral_width": 4.0e4,  # in Hz
            "reference_offset": -2e3,  # in Hz
        }
    ],
)
```

Simulator

Create the Simulator object and add the method and the spin system object.

```
sim = Simulator()
sim.spin_systems += [spin_system]  # add the spin system
sim.methods += [method]  # add the method
sim.run()
```

Post-Simulation Processing

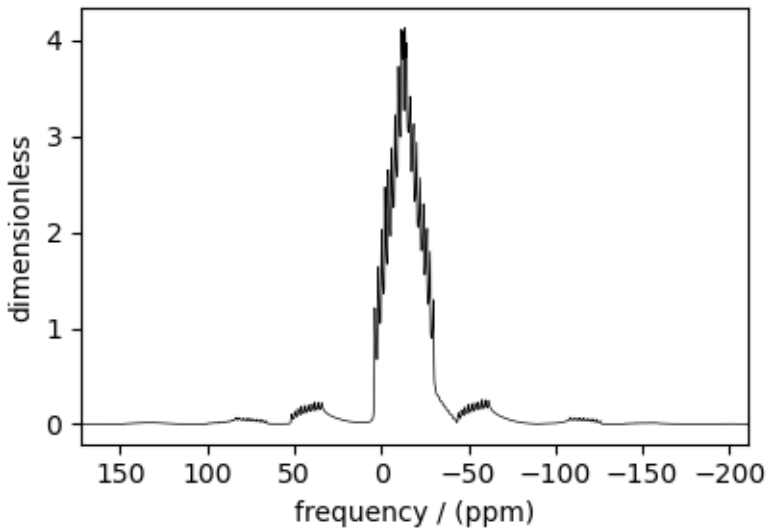
Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="30 Hz"),
        sp.FFT(),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation)
```

Plot

The plot of the simulation before signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(processed_data.real, color="black", linewidth=0.5)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.237 seconds)

11.1.9 Coupled spin-1/2 (CSA + heteronuclear dipolar + J-couplings)

^{13}C-^{1}H sideband simulation

The following simulation is an example by Edén[1] from *Computer Simulations in Solid-State NMR.III.Powder Averaging*. The simulation consists of sideband spectra from a 13C-1H coupled spin system computed at various spinning frequencies with different relative tensor orientations between the nuclear shielding and dipolar interaction tensors.

[1] Edén, M. Computer Simulations in Solid-State NMR. III. Powder Averaging, Concepts in Magnetic Resonance Part A, Vol. 18A(1) 24–55 (2003). DOI: doi.org/10.1002/cmr.a.10065

```
import numpy as np
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
```

Spin Systems

Here, we create three 13C-1H spin systems with different relative orientations between the shielding and dipolar tensors. The Euler angle orientations $\alpha = \gamma = 0$ and β values are listed below.

```
beta_orientation = [np.pi / 6, 5 * np.pi / 18, np.pi / 2]

# The `variable` spin_systems is a list of three coupled 13C-1H spin systems with
# different relative shielding and dipolar tensor orientation.
spin_systems = [
    SpinSystem(
        sites=[
            {
                "isotope": "13C",
                "isotropic_chemical_shift": 0.0,  # in ppm
                "shielding_symmetric": {
                    "zeta": 18.87562,  # in ppm
                    "eta": 0.4,
                    "beta": beta,
                },
            },
            {
                "isotope": "1H",
                "isotropic_chemical_shift": 0.0,  # in ppm
            },
        ],
        couplings=[
            {"site_index": [0, 1], "isotropic_j": 200.0, "dipolar": {"D": -2.1e4}}
        ],
    )
    for beta in beta_orientation
]
```

Methods

Next, we create methods to simulate the sideband manifolds for the above spin systems at four spinning rates: 3 kHz, 5 kHz, 8 kHz, 12 kHz.

```
spin_rates = [3e3, 5e3, 8e3, 12e3]  # in Hz

# The variable `methods` is a list of four BlochDecaySpectrum methods.
methods = [
    BlochDecaySpectrum(
        channels=["13C"],
        magnetic_flux_density=9.4,  # in T
        rotor_frequency=vr,  # in Hz
        spectral_dimensions=[{"count": 2048, "spectral_width": 8.0e4}],
```

(continues on next page)

(continued from previous page)

```
    )
    for vr in spin_rates
]
```

Simulator

Create the Simulator object and add the method and the spin system objects.

```
sim = Simulator()
sim.spin_systems += spin_systems  # add the three spin systems
sim.methods += methods  # add the four methods
sim.config.integration_volume = "hemisphere"  # set averaging to hemisphere
# decompose spectrum to individual spin systems.
sim.config.decompose_spectrum = "spin_system"
```

The run command will simulate twelve spectra corresponding to the three spin systems evaluated at four different methods (spinning speeds).

```
sim.run()
```

Post-Simulation Processing

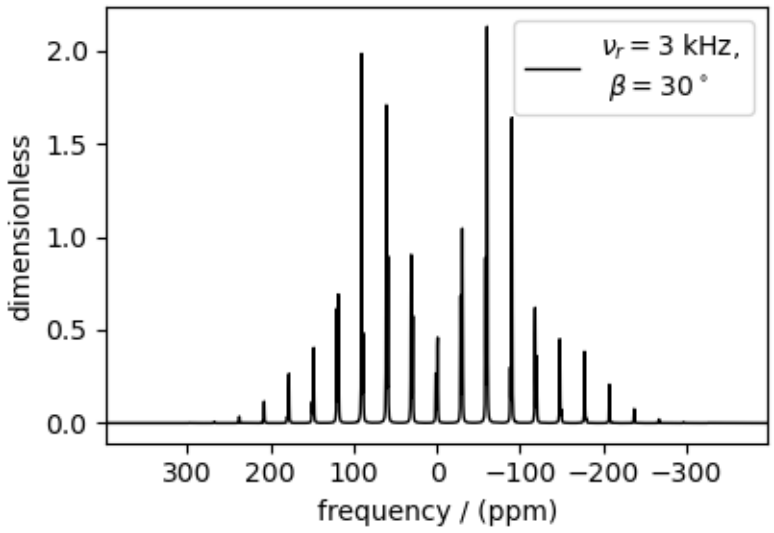
Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="50 Hz"),
        sp.FFT(),
    ]
)
# apply the same post-simulation processing to all the twelve simulations.
processed_data = [
    processor.apply_operations(data=method.simulation) for method in sim.methods
]
```

Plot

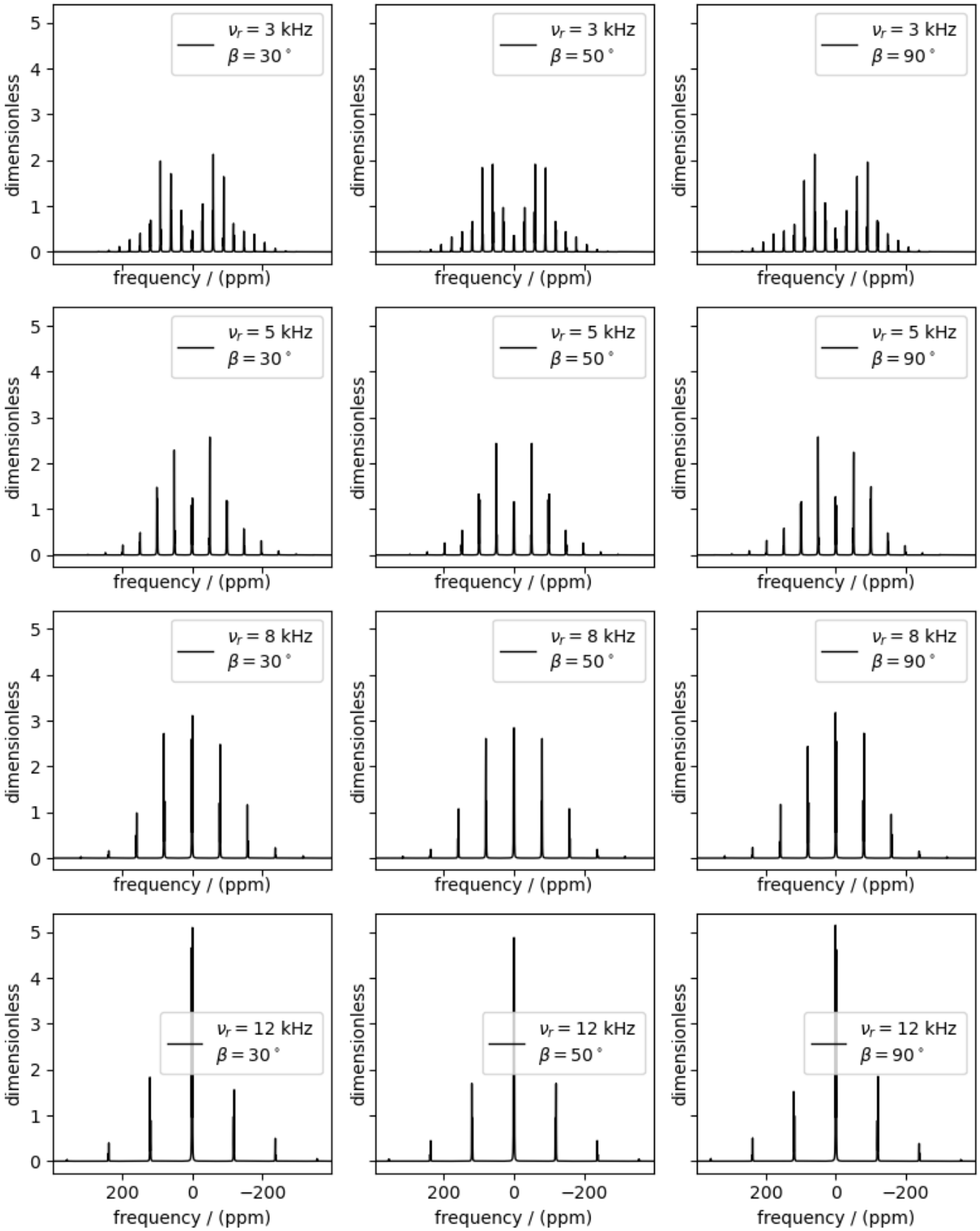
Let's first plot a single simulation, the one corresponding to a relative orientation of $\beta = 30°$ between the shielding and dipolar tensors and a spinning speed of 3 kHz.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(
    processed_data[0].split()[0].real,
    color="black",
    linewidth=1,
    label="$\\nu_r=$3 kHz, \n $\\beta=30^\\circ$",
)
ax.legend()
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



The following is a grid plot showing all twelve simulations. For reference, see Figure 11 from[?].

```
fig, ax = plt.subplots(
    nrows=4,
    ncols=3,
    subplot_kw={"projection": "csdm"},
    sharex=True,
    sharey=True,
    figsize=(8, 10.0),
)
for i, datum in enumerate(processed_data):
    datum_spin_sys = datum.split()  # get simulation from the three spin systems.
    for j, item in enumerate(datum_spin_sys):
        ax[i, j].plot(
            item.real,
            color="black",
            linewidth=1,
            label=(
                f"$\\nu_r={spin_rates[i]/1e3: .0f}$ kHz \n"
                f"$\\beta={beta_orientation[j]/np.pi*180: .0f}^\\circ$"
            ),
        )
        ax[i, j].invert_xaxis()
        ax[i, j].legend()
plt.tight_layout()
plt.show()
```


$\nu_r = 3$ kHz
$\beta = 30^\circ$
$\nu_r = 3$ kHz
$\beta = 50^\circ$
$\nu_r = 3$ kHz
$\beta = 90^\circ$
$\nu_r = 5$ kHz
$\beta = 30^\circ$
$\nu_r = 5$ kHz
$\beta = 50^\circ$
$\nu_r = 5$ kHz
$\beta = 90^\circ$
$\nu_r = 8$ kHz
$\beta = 30^\circ$
$\nu_r = 8$ kHz
$\beta = 50^\circ$
$\nu_r = 8$ kHz
$\beta = 90^\circ$
$\nu_r = 12$ kHz
$\beta = 30^\circ$
$\nu_r = 12$ kHz
$\beta = 50^\circ$
$\nu_r = 12$ kHz
$\beta = 90^\circ$
dimensionless
frequency / (ppm)
200
0
−200

Total running time of the script: (0 minutes 1.837 seconds)

11.1.10 Ethanol Revisited (^{1}H and ^{13}C NMR)

Simulating ^{1}H and ^{13}C isotopomers.

An astute observer may have noticed that the ^{1}H ethanol spectrum from *Coupled Spin System: Using objects* (page 39) was missing the characteristic ^{13}C satellite peaks. In this example, we will add these to the ^{1}H spectrum and plot the ^{13}C spectrum while we're at it!

As before, we start by importing the necessary packages.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site, Coupling
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
```

Spin Systems

The satellite peaks come from couplings between ^{1}H and ^{13}C in low-abundance isotopomers. First, let's define all the possible ^{1}H and ^{13}C sites.

```
H_CH3 = Site(isotope="1H", isotropic_chemical_shift=1.226)
H_CH2 = Site(isotope="1H", isotropic_chemical_shift=2.61)
H_OH = Site(isotope="1H", isotropic_chemical_shift=3.687)

C_CH3 = Site(isotope="13C", isotropic_chemical_shift=18)
C_CH2 = Site(isotope="13C", isotropic_chemical_shift=58)
```

Isotopomer 1

Now, let's define the couplings and build the spin system for the most abundant isotopomer (pictured below).

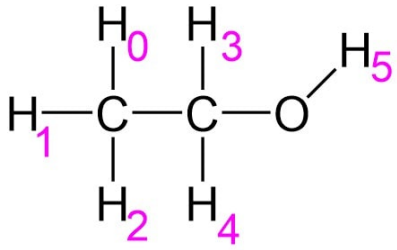


Figure 11.1: An isotopomer of ethanol containing all ^{1}H and all ^{12}C isotopes.

```
iso1_sites = [H_CH3, H_CH3, H_CH3, H_CH2, H_CH2, H_OH]

HH_coupling_1 = Coupling(site_index=[0, 3], isotropic_j=7)
HH_coupling_2 = Coupling(site_index=[0, 4], isotropic_j=7)
HH_coupling_3 = Coupling(site_index=[1, 3], isotropic_j=7)
HH_coupling_4 = Coupling(site_index=[1, 4], isotropic_j=7)
HH_coupling_5 = Coupling(site_index=[2, 3], isotropic_j=7)
```

(continues on next page)

(continued from previous page)

```
HH_coupling_6 = Coupling(site_index=[2, 4], isotropic_j=7)

iso1_couplings = [
    HH_coupling_1,
    HH_coupling_2,
    HH_coupling_3,
    HH_coupling_4,
    HH_coupling_5,
    HH_coupling_6,
]

isotopomer1 = SpinSystem(sites=iso1_sites, couplings=iso1_couplings, abundance=97.812)
```

Note: The abundance values were calculated with an assumption that only ^{1}H and ^{16}O are present. The abundance of ^{12}C is 98.9%, and the abundance of ^{13}C is 1.1%. So, the probability of the most abundant isotopomer is $0.989 \times 0.989 = 0.97812$

Isotopomer 2

Now, we build the sites, couplings ($^{1}J_{\mathrm{CH}}$ and $^{3}J_{\mathrm{HH}}$), and spin system for the isotopomer with the methyl carbon replaced with a ^{13}C (pictured below, ^{13}C marked in blue)

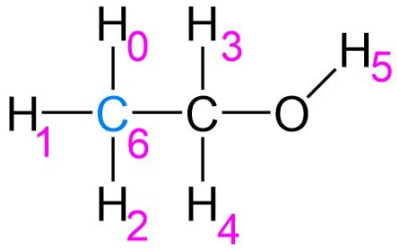


Figure 11.2: Second isotopomer of ethanol containing all ^{1}H, ^{13}C methyl, and ^{12}C methylene isotopes.

```
iso2_sites = [H_CH3, H_CH3, H_CH3, H_CH2, H_CH2, H_OH, C_CH3]

CH3_coupling_1 = Coupling(site_index=[0, 6], isotropic_j=125)
CH3_coupling_2 = Coupling(site_index=[1, 6], isotropic_j=125)
CH3_coupling_3 = Coupling(site_index=[2, 6], isotropic_j=125)

iso2_couplings = iso1_couplings + [CH3_coupling_1, CH3_coupling_2, CH3_coupling_3]

isotopomer2 = SpinSystem(sites=iso2_sites, couplings=iso2_couplings, abundance=1.088)
```

Isotopomer 3

Lastly, we build the sites, couplings, and spin system for the other isotopomer with the methylene carbon replaced with ^{13}C (pictured below, ^{13}C marked in blue)

```
iso3_sites = [H_CH3, H_CH3, H_CH3, H_CH2, H_CH2, H_OH, C_CH2]

CH2_coupling_1 = Coupling(site_index=[3, 6], isotropic_j=141)
```

(continues on next page)

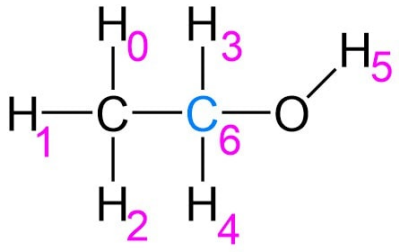


Figure 11.3: Third isotopomer of ethanol containing all ^{1}H, ^{12}C methyl, and ^{13}C methylene isotopes.

(continued from previous page)

```
CH2_coupling_2 = Coupling(site_index=[4, 6], isotropic_j=141)

iso3_couplings = iso1_couplings + [CH2_coupling_1, CH2_coupling_2]

isotopomer3 = SpinSystem(sites=iso3_sites, couplings=iso3_couplings, abundance=1.088)
```

Methods

Now, we define simple 1 pulse-acquire methods for both ^{1}H and ^{13}C.

```
method_H = BlochDecaySpectrum(
    channels=["1H"],
    magnetic_flux_density=9.4,  # T
    spectral_dimensions=[
        {
            "count": 16000,
            "spectral_width": 1.5e3,
            "reference_offset": 950,
            "label": "$^{1}$H frequency",
        }
    ],
)

method_C = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=9.4,  # T
    spectral_dimensions=[
        {
            "count": 32000,
            "spectral_width": 8e3,
            "reference_offset": 4e3,
            "label": "$^{13}$C frequency",
        }
    ],
)
```

Simulation

Now, we create an instance of the simulator object, add our three spin systems, add our two methods, and run the simulation.

```
spin_systems = [isotopomer1, isotopomer2, isotopomer3]
methods = [method_H, method_C]
sim = Simulator(spin_systems=spin_systems, methods=methods)
sim.run()
```

Let's set up our post-simulation processing.

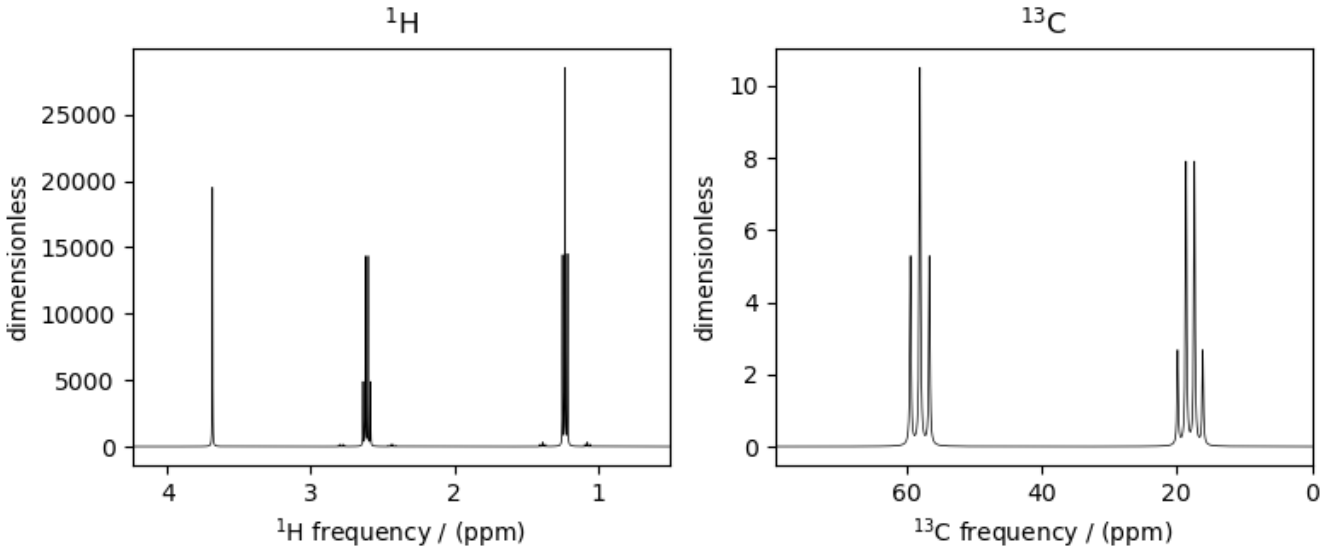
```
processor_1H = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="1 Hz"),
        sp.FFT(),
    ]
)

processor_13C = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="20 Hz"),
        sp.FFT(),
    ]
)
```

Now, let's get our two datasets out of the simulation object and apply the post-processing.

```
H_data = sim.methods[0].simulation
C_data = sim.methods[1].simulation

processed_H_data = processor_1H.apply_operations(data=H_data)
processed_C_data = processor_13C.apply_operations(data=C_data)
```

Lastly, we plot the two spectra.

```
fig, ax = plt.subplots(
    nrows=1, ncols=2, subplot_kw={"projection": "csdm"}, figsize=[8, 3.5]
)

ax[0].plot(processed_H_data.real, color="black", linewidth=0.5)
ax[0].invert_xaxis()
ax[0].set_title("$^1$H")

ax[1].plot(processed_C_data.real, color="black", linewidth=0.5)
ax[1].invert_xaxis()
ax[1].set_title("$^{13}$C")

plt.tight_layout()
plt.show()
```



Now, we see the ^{13}C satellites on either side of the peaks near 1.2 ppm and 2.6 ppm in the ^{1}H spectrum.

Total running time of the script: (0 minutes 0.464 seconds)

11.2 1D NMR simulation (macromolecules/amorphous solids)

The following examples are the NMR spectrum simulation of macromolecules and amorphous materials for the following methods:

- Bloch decay method (*BlochDecaySpectrum* (page 320)),
- Central transition selective Bloch decay method (*BlochDecayCTSpectrum* (page 321)).

For NMR simulation of amorphous solids, we also show examples of simulating spectrum using user-defined model or using commonly accepted models such as Czjzek or extended Czjzek distribution.

11.2.1 Protein GB1, ^{13}C and ^{15}N (I=1/2)

^{13}C/^{15}N (I=1/2) spinning sideband simulation.

The following is the spinning sideband simulation of a macromolecule, protein GB1. The ^{13}C and ^{15}N CSA tensor parameters were obtained from Hung *et al.*[1], which consists of 42 ^{13}Cα, 44 ^{13}CO, and 44 ^{15}NH tensors. In the following example, instead of creating 130 spin systems, we download the spin systems from a remote file and load it directly to the Simulator object.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
```

Create the Simulator object and load the spin systems from an external file.

[1] Hung I., Ge Y., Liu X., Liu M., Li C., Gan Z., Measuring ^{13}C/^{15}N chemical shift anisotropy in [^{13}C, ^{15}N] uniformly enriched proteins using CSA amplification, Solid State Nuclear Magnetic Resonance. 2015, **72**, 96-103. DOI: 10.1016/j.ssnmr.2015.09.002

```
sim = Simulator()

file_ = "https://sandbox.zenodo.org/record/687656/files/protein_GB1_15N_13CA_13CO.mrsys"
sim.load_spin_systems(file_)  # load the spin systems.
print(f"number of spin systems = {len(sim.spin_systems)}")
```

Out:

```
number of spin systems = 130
```

```
all_sites = sim.sites().to_pd()
all_sites.head()
```

Create a ^{13}C Bloch decay spectrum method.

```
method_13C = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=11.74,  # in T
    rotor_frequency=3000,  # in Hz
    spectral_dimensions=[
        {
            "count": 8192,
            "spectral_width": 5e4,  # in Hz
            "reference_offset": 2e4,  # in Hz
            "label": r"$^{13}$C resonances",
        }
    ],
)
```

Since the spin systems contain both ^{13}C and ^{15}N sites, let's also create a ^{15}N Bloch decay spectrum method.

```
method_15N = BlochDecaySpectrum(
    channels=["15N"],
    magnetic_flux_density=11.74,  # in T
    rotor_frequency=3000,  # in Hz
    spectral_dimensions=[
        {
            "count": 8192,
            "spectral_width": 4e4,  # in Hz
            "reference_offset": 7e3,  # in Hz
            "label": r"$^{15}$N resonances",
        }
    ],
)
```

Add the methods to the Simulator object and run the simulation

```
# Add the methods.
sim.methods = [method_13C, method_15N]

# Run the simulation.
sim.run()
```

(continues on next page)

(continued from previous page)

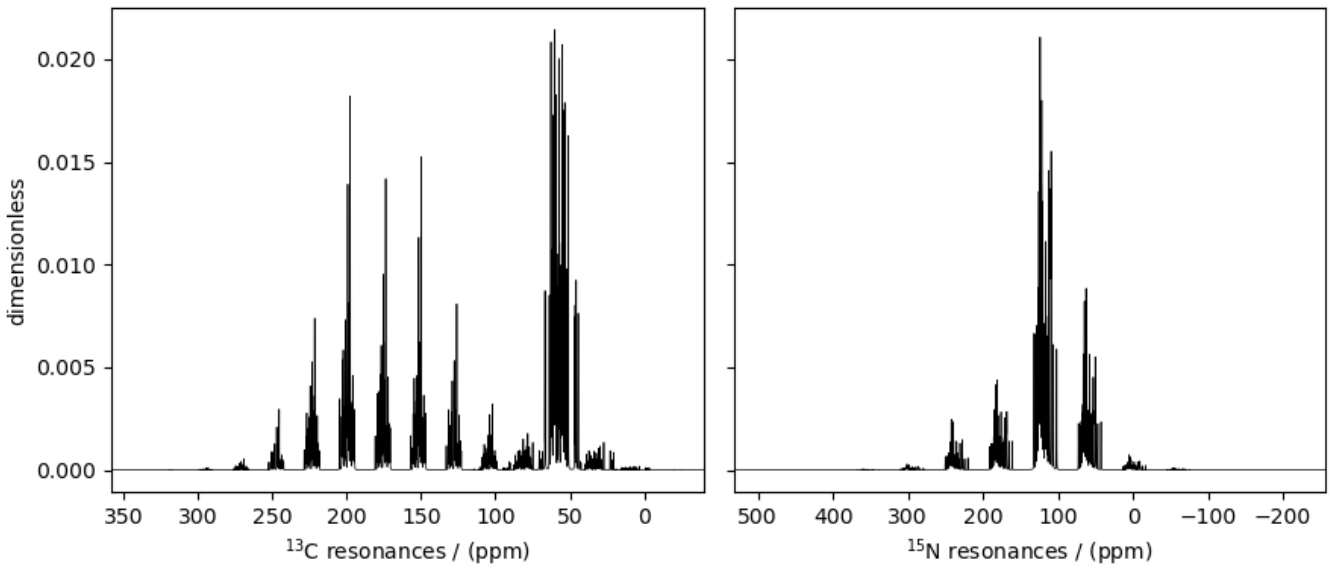
```
# Get the simulation data from the respective methods.
data_13C = sim.methods[0].simulation  # method at index 0 is 13C Bloch decay method.
data_15N = sim.methods[1].simulation  # method at index 1 is 15N Bloch decay method.
```

Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[sp.IFFT(), sp.apodization.Exponential(FWHM="10 Hz"), sp.FFT()]
)
# apply post-simulation processing to data_13C
processed_data_13C = processor.apply_operations(data=data_13C).real

# apply post-simulation processing to data_15N
processed_data_15N = processor.apply_operations(data=data_15N).real
```

The plot of the simulation after signal processing.

```
fig, ax = plt.subplots(
    1, 2, subplot_kw={"projection": "csdm"}, sharey=True, figsize=(9, 4)
)

ax[0].plot(processed_data_13C, color="black", linewidth=0.5)
ax[0].invert_xaxis()

ax[1].plot(processed_data_15N, color="black", linewidth=0.5)
ax[1].set_ylabel(None)
ax[1].invert_xaxis()

plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 2.584 seconds)

11.2.2 Amorphous material, ^{29}Si (I=1/2)

^{29}Si (I=1/2) simulation of amorphous material.

One of the advantages of the `mrsimulator` package is that it is a fast NMR spectrum simulation library. We can exploit this feature to simulate bulk spectra and eventually model amorphous materials. In this section, we illustrate how the `mrsimulator` library may be used in simulating the NMR spectrum of amorphous materials.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

from mrsimulator import Simulator
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator.utils.collection import single_site_system_generator
```

Generating tensor parameter distribution

We model the amorphous material by assuming a distribution of interaction tensors. For example, a tri-variate normal distribution of the shielding tensor parameters, *i.e.*, the isotropic chemical shift, the anisotropy parameter, ζ, and the asymmetry parameter, η. In the following, we use pure NumPy and SciPy methods to generate the three-dimensional distribution, as follows,

```
mean = [-100, 50, 0.15]  # given as [isotropic chemical shift in ppm, zeta in ppm, eta].
covariance = [[3.25, 0, 0], [0, 26.2, 0], [0, 0, 0.002]]  # same order as the mean.

# range of coordinates along the three dimensions
iso_range = np.arange(100) * 0.3055 - 115  # in ppm
zeta_range = np.arange(30) * 2.5 + 10  # in ppm
eta_range = np.arange(21) / 20

# The coordinates grid
iso, zeta, eta = np.meshgrid(iso_range, zeta_range, eta_range, indexing="ij")
pos = np.asarray([iso, zeta, eta]).T

# Three-dimensional probability distribution function.
pdf = multivariate_normal(mean=mean, cov=covariance).pdf(pos).T
```

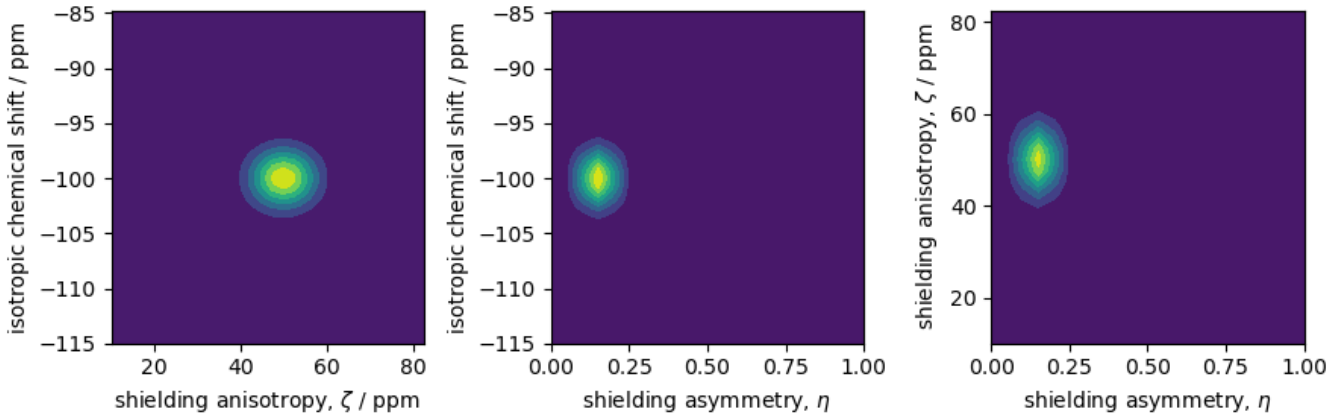
Here, `iso`, `zeta`, and `eta` are the isotropic chemical shift, nuclear shielding anisotropy, and nuclear shielding asymmetry coordinates of the 3D-grid system over which the multivariate normal probability distribution is evaluated. The mean of the distribution is given by the variable `mean` and holds a value of -100 ppm, 50 ppm, and 0.15 for the isotropic chemical shift, nuclear shielding anisotropy, and nuclear shielding asymmetry parameter, respectively. Similarly, the variable `covariance` holds the covariance matrix of the multivariate normal distribution. The two-dimensional projections from this three-dimensional distribution are shown below.

```
_, ax = plt.subplots(1, 3, figsize=(9, 3))

# isotropic shift v.s. shielding anisotropy
ax[0].contourf(zeta_range, iso_range, pdf.sum(axis=2))
ax[0].set_xlabel(r"shielding anisotropy, $\zeta$ / ppm")
ax[0].set_ylabel("isotropic chemical shift / ppm")

# isotropic shift v.s. shielding asymmetry
```

(continues on next page)

(continued from previous page)

```
ax[1].contourf(eta_range, iso_range, pdf.sum(axis=1))
ax[1].set_xlabel(r"shielding asymmetry, $\eta$")
ax[1].set_ylabel("isotropic chemical shift / ppm")

# shielding anisotropy v.s. shielding asymmetry
ax[2].contourf(eta_range, zeta_range, pdf.sum(axis=0))
ax[2].set_xlabel(r"shielding asymmetry, $\eta$")
ax[2].set_ylabel(r"shielding anisotropy, $\zeta$ / ppm")

plt.tight_layout()
plt.show()
```



Create the Simulator object

Spin system:

Let's create the sites and single-site spin system objects from these parameters. Use the *single_site_system_generator()* (page 339) utility function to generate single-site spin systems. # Here, `iso`, `zeta`, and `eta` are the array of tensor parameter coordinates, and `pdf` is the array of the corresponding amplitudes.

```
spin_systems = single_site_system_generator(
    isotopes="29Si",
    isotropic_chemical_shifts=iso,
    shielding_symmetric={"zeta": zeta, "eta": eta},
    abundance=pdf,
)
```

Method:

Let's also create a Bloch decay spectrum method.

```
method = BlochDecaySpectrum(
    channels=["29Si"],
    spectral_dimensions=[
        {"spectral_width": 25000, "reference_offset": -7000}  # values in Hz
    ],
)
```

The above method simulates a static ^{29}Si spectrum at 9.4 T field (default value).

Simulator:

Now that we have the spin systems and the method, create the simulator object and add the respective objects.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods += [method]  # add the method
```

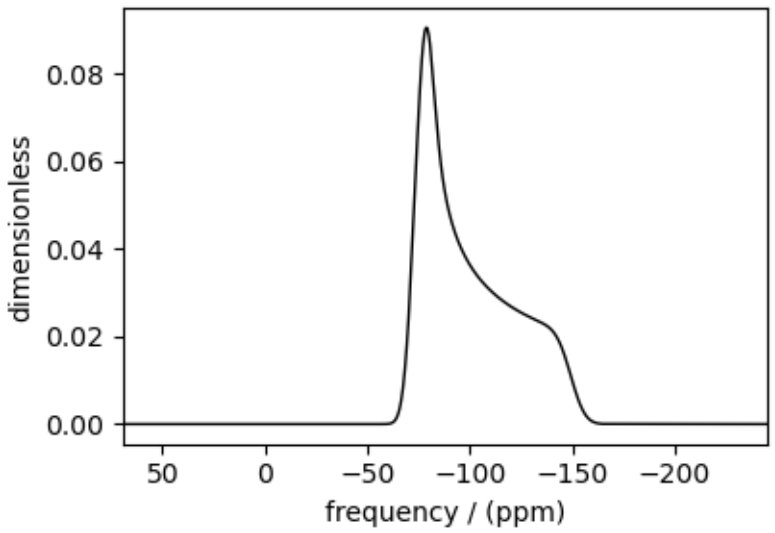
Static spectrum

Observe the static ^{29}Si NMR spectrum simulation.

```
sim.run()
```

The plot of the simulation.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Note: The broad spectrum seen in the above figure is a result of spectral averaging of spectra arising from a distribution of shielding tensors. There is no line-broadening filter applied to the spectrum.

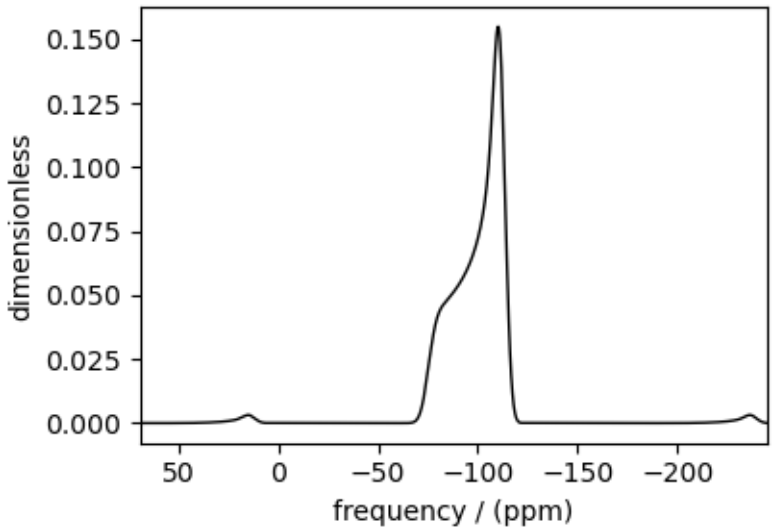
Spinning sideband simulation at 90°

Here is an example of a sideband simulation, spinning at a 90-degree angle.

```
sim.methods[0] = BlochDecaySpectrum(
    channels=["29Si"],
    rotor_frequency=5000,  # in Hz
    rotor_angle=1.57079,  # in rads, equivalent to 90 deg.
    spectral_dimensions=[
        {"spectral_width": 25000, "reference_offset": -7000}  # values in Hz
    ],
)
sim.config.number_of_sidebands = 8  # eight sidebands are sufficient for this example
sim.run()
```

The plot of the simulation.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Spinning sideband simulation at the magic angle

Here is another example of a sideband simulation at the magic angle.

```
sim.methods[0] = BlochDecaySpectrum(
    channels=["29Si"],
    rotor_frequency=1000,  # in Hz
    rotor_angle=54.735 * np.pi / 180.0,  # in rads
    spectral_dimensions=[
        {"spectral_width": 25000, "reference_offset": -7000}  # values in Hz
```

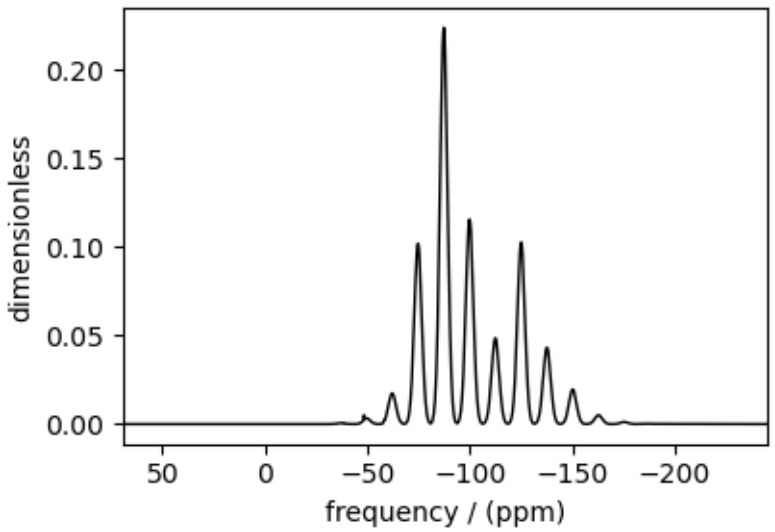
(continues on next page)

(continued from previous page)

```
    ],
)
sim.config.number_of_sidebands = 16  # sixteen sidebands are sufficient for this example
sim.run()
```

The plot of the simulation.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 11.267 seconds)

11.2.3 Amorphous material, ^{27}Al (I=5/2)

^{27}Al (I=5/2) simulation of amorphous material.

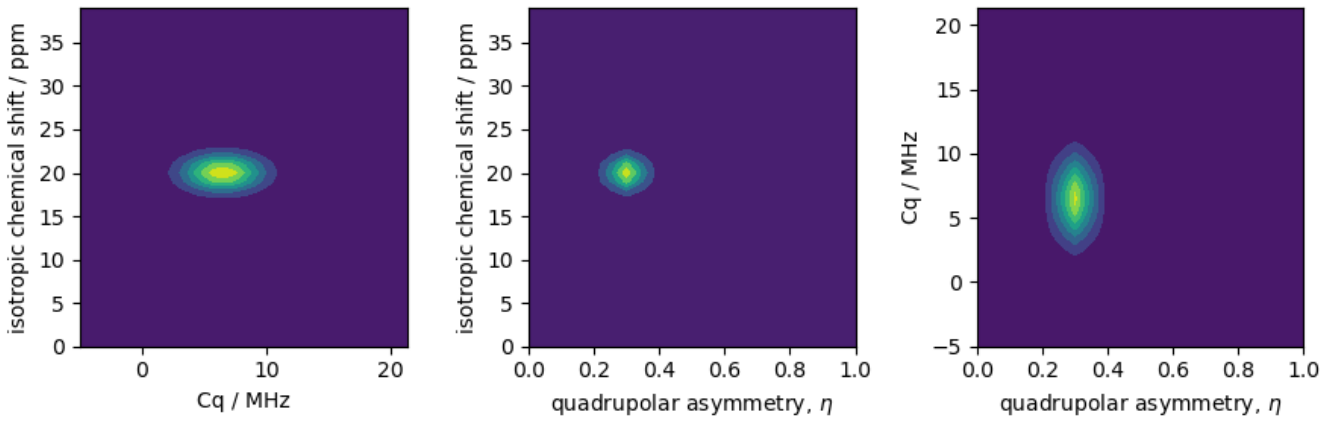
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

from mrsimulator import Simulator
from mrsimulator.methods import BlochDecayCTSpectrum
from mrsimulator.utils.collection import single_site_system_generator
```

In this section, we illustrate the simulation of a quadrupolar spectrum arising from a distribution of the electric field gradient (EFG) tensors from an amorphous material. We proceed by assuming a multi-variate normal distribution, as follows,

```
mean = [20, 6.5, 0.3]  # given as [isotropic chemical shift in ppm, Cq in MHz, eta].
covariance = [[1.98, 0, 0], [0, 4.9, 0], [0, 0, 0.0016]]  # same order as the mean.

# range of coordinates along the three dimensions
iso_range = np.arange(40)  # in ppm
Cq_range = np.arange(80) / 3 - 5  # in MHz
eta_range = np.arange(21) / 20

# The coordinates grid
iso, Cq, eta = np.meshgrid(iso_range, Cq_range, eta_range, indexing="ij")
pos = np.asarray([iso, Cq, eta]).T

# Three-dimensional probability distribution function.
pdf = multivariate_normal(mean=mean, cov=covariance).pdf(pos).T
```

Here, `iso`, `Cq`, and `eta` are the isotropic chemical shift, the quadrupolar coupling constant, and quadrupolar asymmetry coordinates of the 3D-grid system over which the multivariate normal probability distribution is evaluated. The mean of the distribution is given by the variable `mean` and holds a value of 20 ppm, 6.5 MHz, and 0.3 for the isotropic chemical shift, the quadrupolar coupling constant, and quadrupolar asymmetry parameter, respectively. Similarly, the variable `covariance` holds the covariance matrix of the multivariate normal distribution. The two-dimensional projections from this three-dimensional distribution are shown below.

```
_, ax = plt.subplots(1, 3, figsize=(9, 3))

# isotropic shift v.s. quadrupolar coupling constant
ax[0].contourf(Cq_range, iso_range, pdf.sum(axis=2))
ax[0].set_xlabel("Cq / MHz")
ax[0].set_ylabel("isotropic chemical shift / ppm")

# isotropic shift v.s. quadrupolar asymmetry
ax[1].contourf(eta_range, iso_range, pdf.sum(axis=1))
ax[1].set_xlabel(r"quadrupolar asymmetry, $\eta$")
ax[1].set_ylabel("isotropic chemical shift / ppm")

# quadrupolar coupling constant v.s. quadrupolar asymmetry
ax[2].contourf(eta_range, Cq_range, pdf.sum(axis=0))
ax[2].set_xlabel(r"quadrupolar asymmetry, $\eta$")
ax[2].set_ylabel("Cq / MHz")

plt.tight_layout()
plt.show()
```



Let's create the site and spin system objects from these parameters. Note, we create single-site spin systems for optimum performance. Use the *single_site_system_generator()* (page 339) utility function to generate single-site spin systems.

```
spin_systems = single_site_system_generator(
    isotopes="27Al",
    isotropic_chemical_shifts=iso,
    quadrupolar={"Cq": Cq * 1e6, "eta": eta},  # Cq in Hz
    abundance=pdf,
)
```

Static spectrum

Observe the static ^{27}Al NMR spectrum simulation. First, create a central transition selective Bloch decay spectrum method.

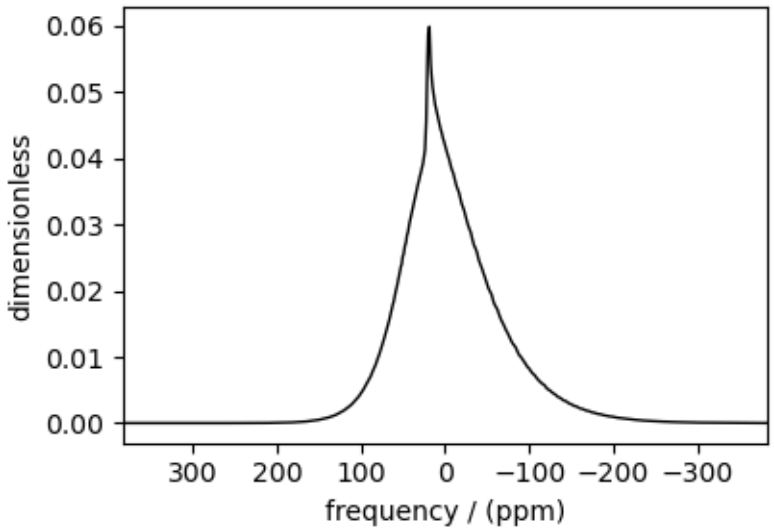
```
static_method = BlochDecayCTSpectrum(
    channels=["27Al"], spectral_dimensions=[{"spectral_width": 80000}]
)
```

Create the simulator object and add the spin systems and method.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [static_method]  # add the method
sim.run()
```

The plot of the corresponding spectrum.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Spinning sideband simulation at the magic angle

Simulation of the same spin systems at the magic angle and spinning at 25 kHz.

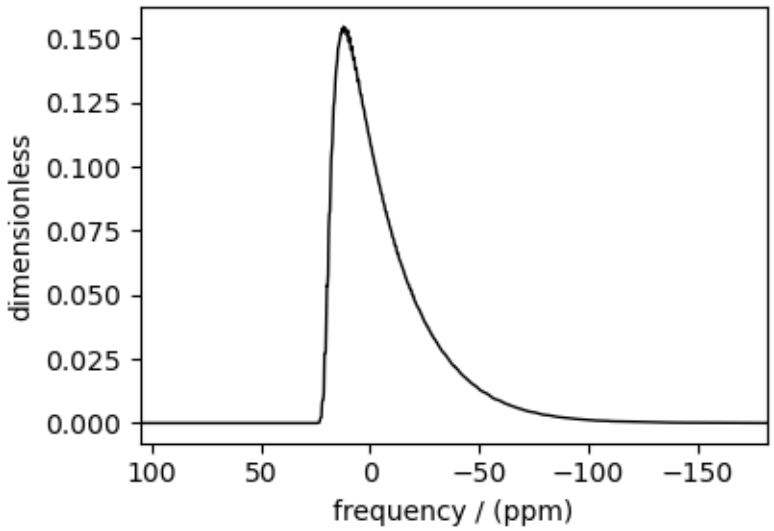
```
MAS_method = BlochDecayCTSpectrum(
    channels=["27Al"],
    rotor_frequency=25000,  # in Hz
    rotor_angle=54.735 * np.pi / 180.0,  # in rads
    spectral_dimensions=[
        {"spectral_width": 30000, "reference_offset": -4000}  # values in Hz
    ],
)
sim.methods[0] = MAS_method
```

Configure the sim object to calculate up to 4 sidebands, and run the simulation.

```
sim.config.number_of_sidebands = 4
sim.run()
```

and the corresponding plot.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 3.818 seconds)

11.2.4 Czjzek distribution (Shielding and Quadrupolar)

In this example, we illustrate the simulation of spectrum originating from a Czjzek distribution of traceless symmetric tensors. We show two cases, the Czjzek distribution of the shielding and quadrupolar tensor parameters, respectively.

Import the required modules.

```
import numpy as np
import matplotlib.pyplot as plt

from mrsimulator import Simulator
from mrsimulator.methods import BlochDecaySpectrum, BlochDecayCTSpectrum
from mrsimulator.models import CzjzekDistribution
from mrsimulator.utils.collection import single_site_system_generator
```

Symmetric shielding tensor

Create the Czjzek distribution

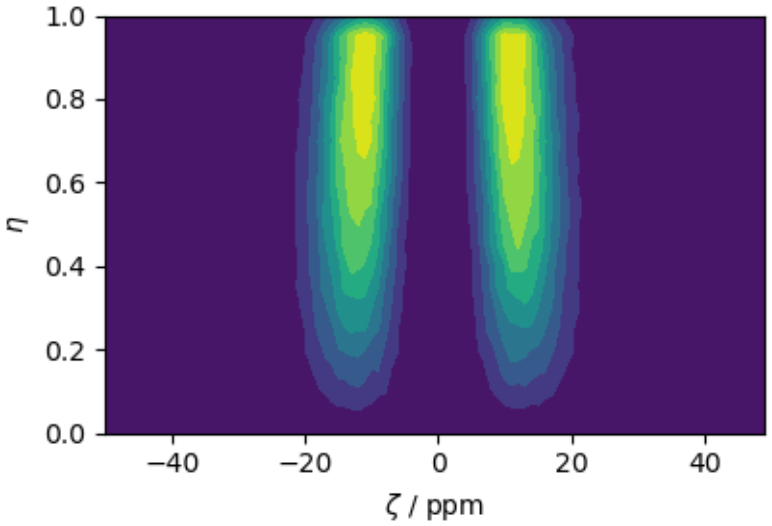
First, create a distribution of the zeta and eta parameters of the shielding tensors using the *Czjzek distribution* (page 65) model as follows.

```
# The range of zeta and eta coordinates over which the distribution is sampled.
z_range = np.arange(100) - 50  # in ppm
e_range = np.arange(21) / 20
z_dist, e_dist, amp = CzjzekDistribution(sigma=3.1415).pdf(pos=[z_range, e_range])
```

Here `z_range` and `e_range` are the coordinates along the ζ and η dimensions that form a two-dimensional ζ-η grid. The argument *sigma* of the CzjzekDistribution class is the standard deviation of the second-rank tensor parameters used in generating the distribution, and *pos* hold the one-dimensional arrays of ζ and η coordinates, respectively.

The following is the contour plot of the Czjzek distribution.

```
plt.figure(figsize=(4.25, 3.0))
plt.contourf(z_dist, e_dist, amp, levels=10)
plt.xlabel(r"$\zeta$ / ppm")
plt.ylabel(r"$\eta$")
plt.tight_layout()
plt.show()
```



Simulate the spectrum

To quickly generate single-site spin systems from the above ζ and η parameters, use the *single_site_system_generator()* (page 339) utility function.

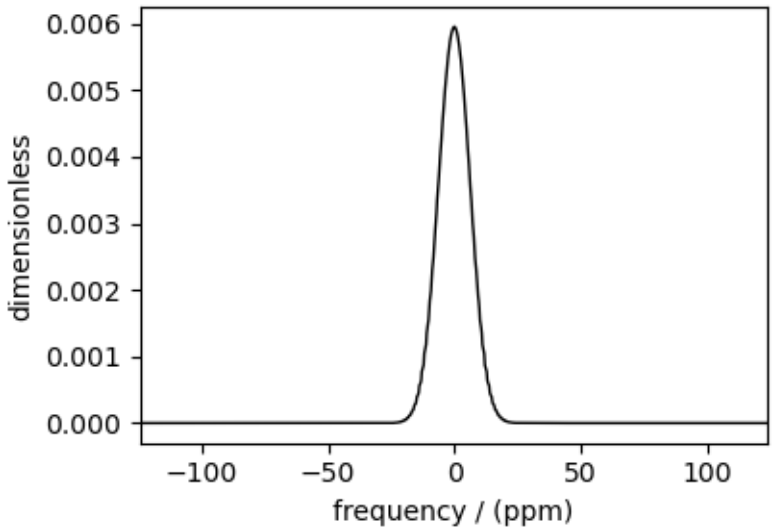
```
systems = single_site_system_generator(
    isotopes="13C", shielding_symmetric={"zeta": z_dist, "eta": e_dist}, abundance=amp
)
```

Here, the variable `systems` hold an array of single-site spin systems. Next, create a simulator object and add the above system and a method.

```
sim = Simulator()
sim.spin_systems = systems  # add the systems
sim.methods = [BlochDecaySpectrum(channels=["13C"])]  # add the method
sim.run()
```

The following is the static spectrum arising from a Czjzek distribution of the second-rank traceless shielding tensors.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
plt.tight_layout()
plt.show()
```

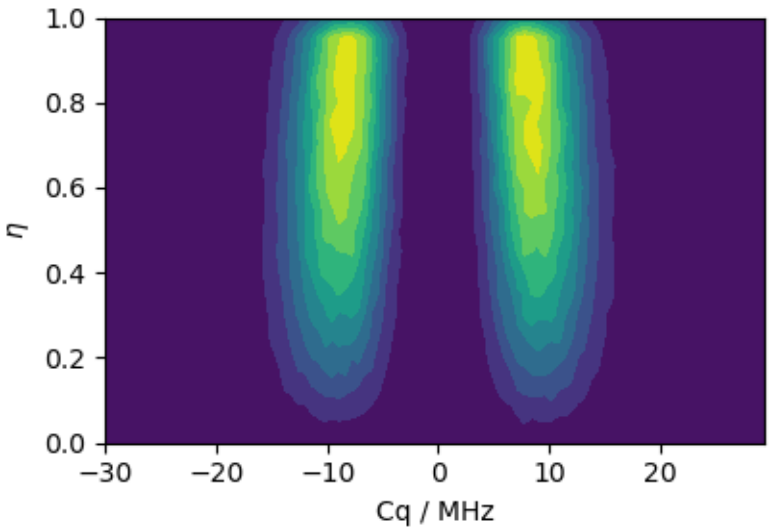


Quadrupolar tensor

Create the Czjzek distribution

Similarly, you may also create a Czjzek distribution of the electric field gradient (EFG) tensor parameters.

```
# The range of Cq and eta coordinates over which the distribution is sampled.
cq_range = np.arange(100) * 0.6 - 30  # in MHz
e_range = np.arange(21) / 20
cq_dist, e_dist, amp = CzjzekDistribution(sigma=2.3).pdf(pos=[cq_range, e_range])

# The following is the contour plot of the Czjzek distribution.
plt.figure(figsize=(4.25, 3.0))
plt.contourf(cq_dist, e_dist, amp, levels=10)
plt.xlabel(r"Cq / MHz")
plt.ylabel(r"$\eta$")
plt.tight_layout()
plt.show()
```



Simulate the spectrum

Create the spin systems.

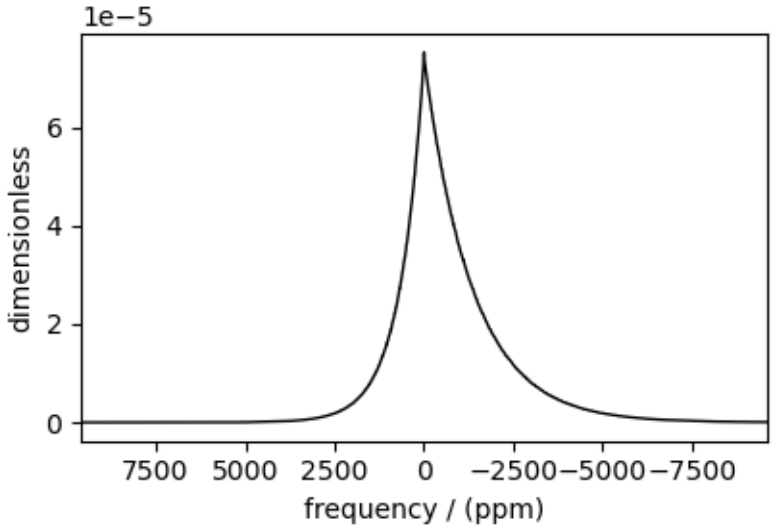
```
systems = single_site_system_generator(
    isotopes="71Ga", quadrupolar={"Cq": cq_dist * 1e6, "eta": e_dist}, abundance=amp
)
```

Create a simulator object and add the above system.

```
sim = Simulator()
sim.spin_systems = systems  # add the systems
sim.methods = [
    BlochDecayCTSpectrum(
        channels=["71Ga"],
        magnetic_flux_density=4.8,  # in T
        spectral_dimensions=[{"count": 2048, "spectral_width": 1.2e6}],
    )
]  # add the method
sim.run()
```

The following is the static spectrum arising from a Czjzek distribution of the second-rank traceless EFG tensors.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 3.173 seconds)

11.2.5 Extended Czjzek distribution (Shielding and Quadrupolar)

In this example, we illustrate the simulation of spectrum originating from an extended Czjzek distribution of traceless symmetric tensors. We show two cases, an extended Czjzek distribution of the shielding and quadrupolar tensor parameters, respectively.

Import the required modules.

```
import numpy as np
import matplotlib.pyplot as plt

from mrsimulator import Simulator
from mrsimulator.methods import BlochDecaySpectrum, BlochDecayCTSpectrum
from mrsimulator.models import ExtCzjzekDistribution
from mrsimulator.utils.collection import single_site_system_generator
```

Symmetric shielding tensor

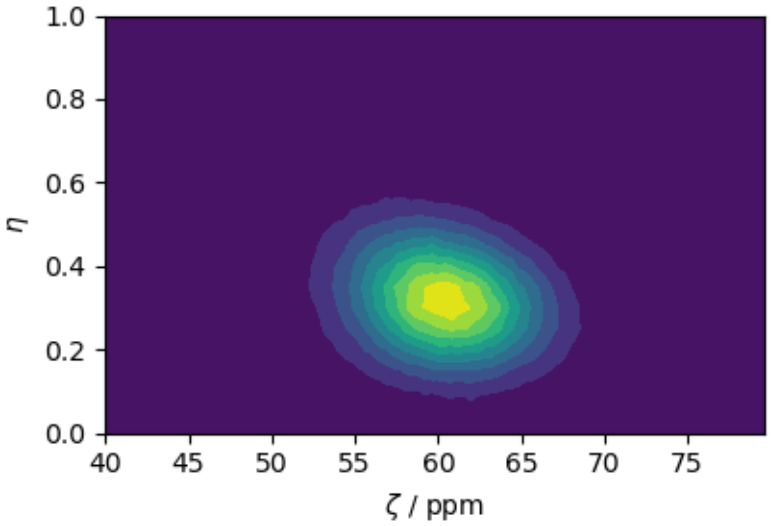
Create the extended Czjzek distribution

First, create a distribution of the zeta and eta parameters of the shielding tensors using the *Extended Czjzek distribution* (page 69) model as follows,

```
# The range of zeta and eta coordinates over which the distribution is sampled.
z_lim = np.arange(100) * 0.4 + 40  # in ppm
e_lim = np.arange(21) / 20

dominant = {"zeta": 60, "eta": 0.3}
z_dist, e_dist, amp = ExtCzjzekDistribution(dominant, eps=0.14).pdf(pos=[z_lim, e_lim])
```

The following is the plot of the extended Czjzek distribution.

```
plt.figure(figsize=(4.25, 3.0))
plt.contourf(z_dist, e_dist, amp, levels=10)
plt.xlabel(r"$\zeta$ / ppm")
plt.ylabel(r"$\eta$")
plt.tight_layout()
plt.show()
```



Simulate the spectrum

Create the spin systems from the above ζ and η parameters.

```
systems = single_site_system_generator(
    isotopes="13C", shielding_symmetric={"zeta": z_dist, "eta": e_dist}, abundance=amp
)
print(len(systems))
```

Out:

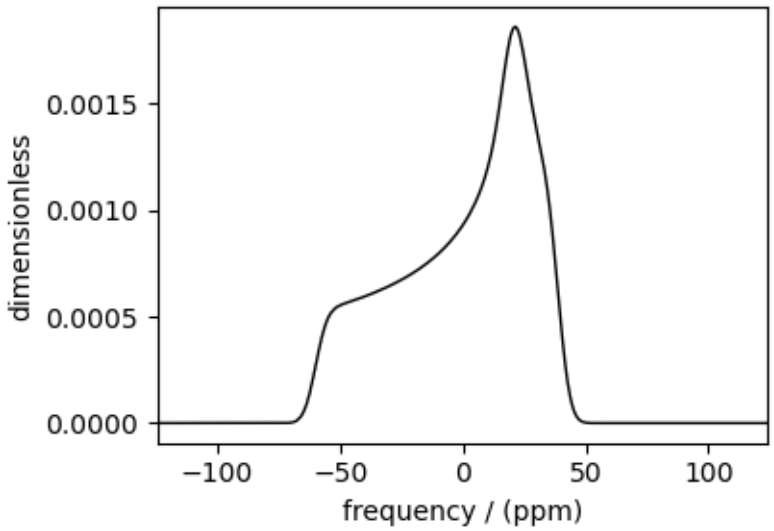
```
838
```

Create a simulator object and add the above system.

```
sim = Simulator()
sim.spin_systems = systems  # add the systems
sim.methods = [BlochDecaySpectrum(channels=["13C"])]  # add the method
sim.run()
```

The following is the static spectrum arising from a Czjzek distribution of the second-rank traceless shielding tensors.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
plt.tight_layout()
plt.show()
```



Quadrupolar tensor

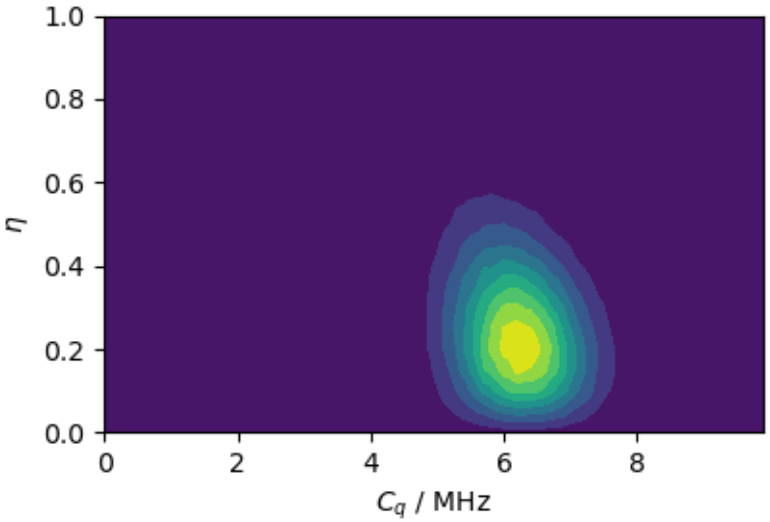
Create the extended Czjzek distribution

Similarly, you may also create an extended Czjzek distribution of the electric field gradient (EFG) tensor parameters.

```
# The range of Cq and eta coordinates over which the distribution is sampled.
cq_lim = np.arange(100) * 0.1  # assumed in MHz
e_lim = np.arange(21) / 20

dominant = {"Cq": 6.1, "eta": 0.1}
cq_dist, e_dist, amp = ExtCzjzekDistribution(dominant, eps=0.25).pdf(
    pos=[cq_lim, e_lim]
)
```

The following is the plot of the extended Czjzek distribution.

```
plt.figure(figsize=(4.25, 3.0))
plt.contourf(cq_dist, e_dist, amp, levels=10)
plt.xlabel(r"$C_q$ / MHz")
plt.ylabel(r"$\eta$")
plt.tight_layout()
plt.show()
```



Simulate the spectrum

Static spectrum Create the spin systems.

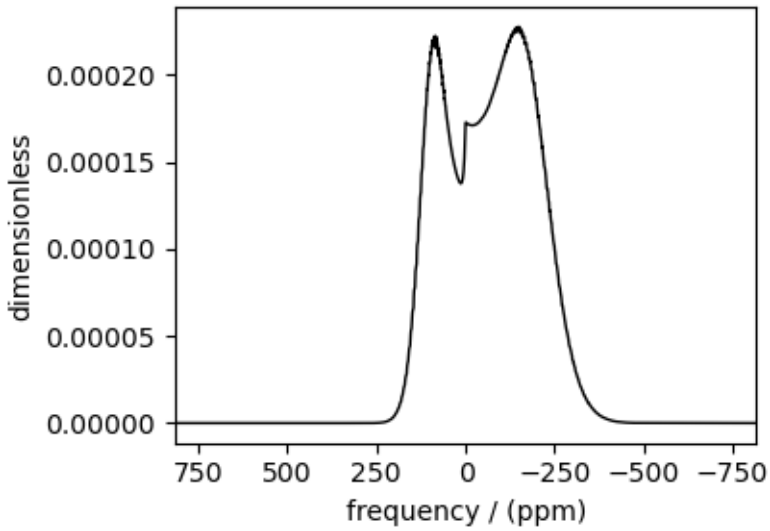
```
systems = single_site_system_generator(
    isotopes="71Ga", quadrupolar={"Cq": cq_dist * 1e6, "eta": e_dist}, abundance=amp
)
```

Create a simulator object and add the above system.

```
sim = Simulator()
sim.spin_systems = systems  # add the systems
sim.methods = [
    BlochDecayCTSpectrum(
        channels=["71Ga"],
        magnetic_flux_density=9.4,  # in T
        spectral_dimensions=[{"count": 2048, "spectral_width": 2e5}],
    )
]  # add the method
sim.run()
```

The following is a static spectrum arising from an extended Czjzek distribution of the second-rank traceless EFG tensors.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```

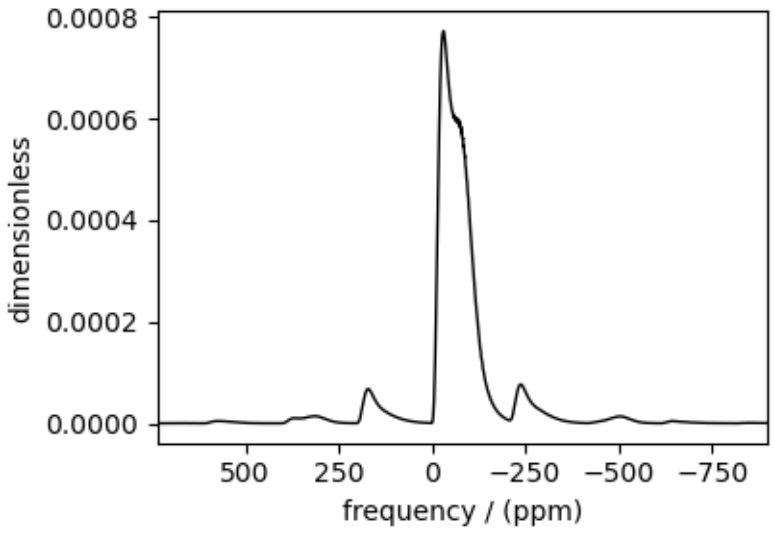


MAS spectrum

```
sim.methods = [
    BlochDecayCTSpectrum(
        channels=["71Ga"],
        magnetic_flux_density=9.4,  # in T
        rotor_frequency=25000,  # in Hz
        spectral_dimensions=[
            {"count": 2048, "spectral_width": 2e5, "reference_offset": -1e4}
        ],
    )
]  # add the method
sim.config.number_of_sidebands = 16
sim.run()
```

The following is the MAS spectrum arising from an extended Czjzek distribution of the second-rank traceless EFG tensors.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(sim.methods[0].simulation, color="black", linewidth=1)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 5.620 seconds)

11.3 2D NMR simulation (Crystalline solids)

The following examples are the NMR spectrum simulation for crystalline solids. The examples include the illustrations for the following methods:

- Triple-quantum variable-angle spinning (i.e., 3Q-MAS) using the specialized *ThreeQ_VAS()* (page 325) method.
- Satellite-transition variable-angle spinning (i.e., ST-MAS) using the specialized *ST1_VAS()* (page 329) method.
- Switched Angle Spinning (SAS) using the generic *Method2D()* (page 322) method.
- MAS-detected Dynamic Angle Spinning (DAS) using the generic *Method2D()* (page 322) method.
- Correlation of Anisotropies Separated Through Echo Refocusing (COASTER) using the generic *Method2D()* (page 322) method.
- Phase Adjusted Spinning Sidebands (PASS and QPASS) and Magic-Angle Turning (MAT and QMAT) using the specialized *SSB2D()* (page 332) method.

11.3.1 $RbNO_3$, ^{87}Rb (I=3/2) 3QMAS

^{87}Rb (I=3/2) triple-quantum magic-angle spinning (3Q-MAS) simulation.

The following is an example of the 3QMAS simulation of $RbNO_3$, which has three distinct ^{87}Rb sites. The ^{87}Rb tensor parameters were obtained from Massiot *et al.*[1]. In this simulation, a Gaussian broadening is applied to the spectrum as a post-simulation step.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import ThreeQ_VAS
from mrsimulator import signal_processing as sp
```

Generate the site and spin system objects.

[1] Massiot, D., Touzoa, B., Trumeaua, D., Coutures, J.P., Virlet, J., Florian, P., Grandinetti, P.J. Two-dimensional magic-angle spinning isotropic reconstruction sequences for quadrupolar nuclei, ssnmr, (1996), **6**, *1*, 73-83. DOI: 10.1016/0926-2040(95)01210-9

```
Rb87_1 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-27.4,  # in ppm
    quadrupolar={"Cq": 1.68e6, "eta": 0.2},  # Cq is in Hz
)
Rb87_2 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-28.5,  # in ppm
    quadrupolar={"Cq": 1.94e6, "eta": 1.0},  # Cq is in Hz
)
Rb87_3 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-31.3,  # in ppm
    quadrupolar={"Cq": 1.72e6, "eta": 0.5},  # Cq is in Hz
)

sites = [Rb87_1, Rb87_2, Rb87_3]  # all sites
spin_systems = [SpinSystem(sites=[s]) for s in sites]
```

Select a Triple Quantum variable-angle spinning method. You may optionally provide a *rotor_ angle* to the method. The default *rotor_ angle* is the magic-angle.

```
method = ThreeQ_VAS(
    channels=["87Rb"],
    magnetic_flux_density=9.4,  # in T
    spectral_dimensions=[
        {
            "count": 128,
            "spectral_width": 7e3,  # in Hz
            "reference_offset": -7e3,  # in Hz
            "label": "Isotropic dimension",
        },
        {
            "count": 256,
            "spectral_width": 1e4,  # in Hz
            "reference_offset": -4e3,  # in Hz
            "label": "MAS dimension",
        },
    ],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [method]  # add the method.
sim.run()
```

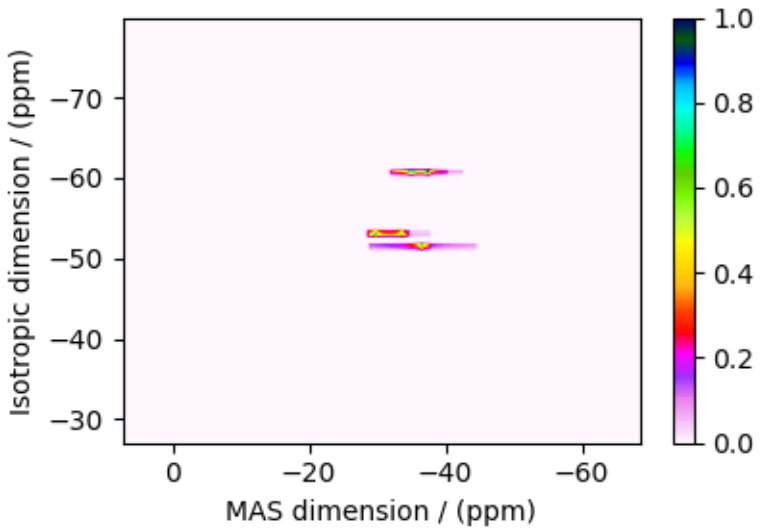
The plot of the simulation.

```
data = sim.methods[0].simulation

plt.figure(figsize=(4.25, 3.0))
```

(continues on next page)

(continued from previous page)

```
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```

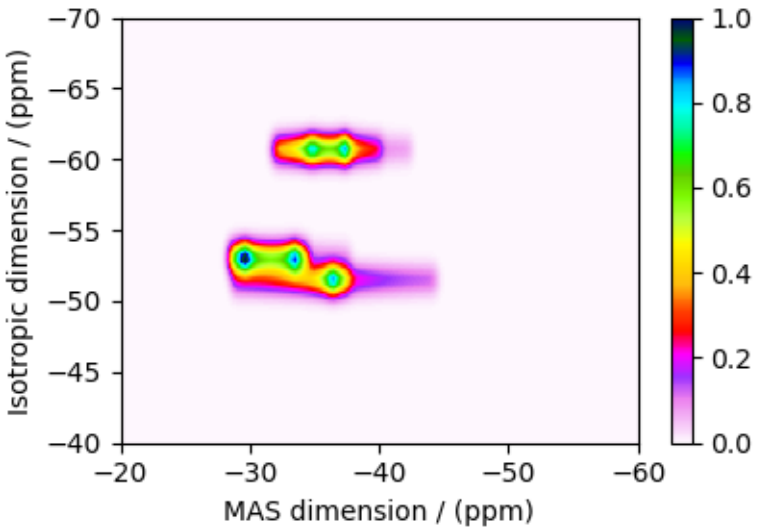


Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.08 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.22 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation)
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.set_ylim(-40, -70)
ax.set_xlim(-20, -60)
plt.tight_layout()
plt.show()
```



See also:

Simulating site disorder (crystalline) (page 156) for RbNO3.

Total running time of the script: (0 minutes 0.488 seconds)

11.3.2 Albite, ^{27}Al (I=5/2) 3QMAS

^{27}Al (I=5/2) triple-quantum magic-angle spinning (3Q-MAS) simulation.

The following is an example of ^{27}Al 3QMAS simulation of albite $NaSi_3AlO_8$. The ^{27}Al tensor parameters were obtained from Massiot *et al.*[1].

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import ThreeQ_VAS
from mrsimulator import signal_processing as sp
```

Generate the site and spin system objects.

```
site = Site(
    isotope="27Al",
    isotropic_chemical_shift=64.7,  # in ppm
    quadrupolar={"Cq": 3.25e6, "eta": 0.68},  # Cq is in Hz
)

spin_systems = [SpinSystem(sites=[site])]
```

Select a Triple Quantum variable-angle spinning method. You may optionally provide a *rotor_angle* to the method. The default *rotor_angle* is the magic-angle.

[1] Massiot, D., Touzoa, B., Trumeaua, D., Coutures, J.P., Virlet, J., Florian, P., Grandinetti, P.J. Two-dimensional magic-angle spinning isotropic reconstruction sequences for quadrupolar nuclei, ssnmr, (1996), **6**, *1*, 73-83. DOI: 10.1016/0926-2040(95)01210-9

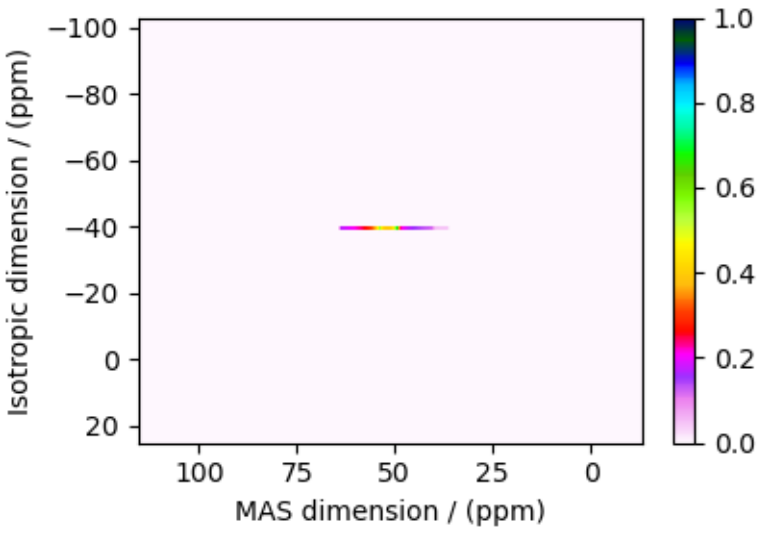
```
method = ThreeQ_VAS(
    channels=["27Al"],
    magnetic_flux_density=7,  # in T
    spectral_dimensions=[
        {
            "count": 256,
            "spectral_width": 1e4,  # in Hz
            "reference_offset": -3e3,  # in Hz
            "label": "Isotropic dimension",
        },
        {
            "count": 512,
            "spectral_width": 1e4,  # in Hz
            "reference_offset": 4e3,  # in Hz
            "label": "MAS dimension",
        },
    ],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [method]  # add the method.
sim.run()
```

The plot of the simulation.

```
data = sim.methods[0].simulation

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```

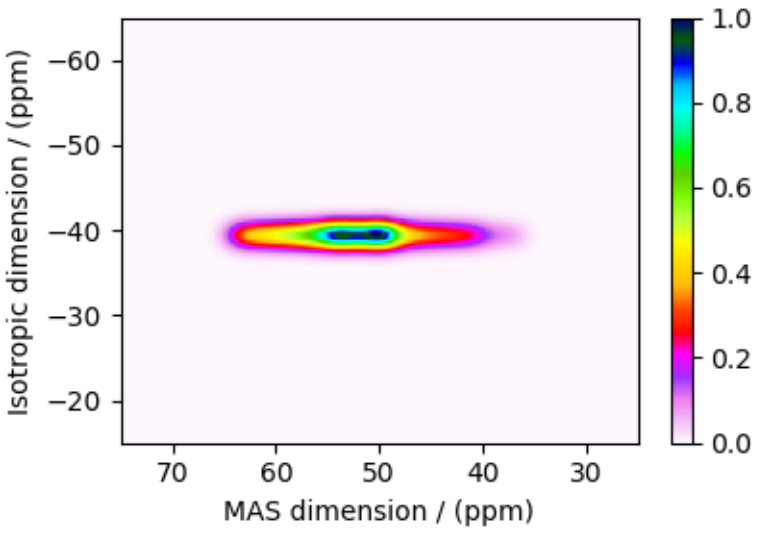


Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.2 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.2 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation)
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.set_xlim(75, 25)
ax.set_ylim(-15, -65)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.487 seconds)

11.3.3 $RbNO_3$, ^{87}Rb (I=3/2) STMAS

^{87}Rb (I=3/2) staellite-transition off magic-angle spinning simulation.

The following is an example of the STMAS simulation of $RbNO_3$. The ^{87}Rb tensor parameters were obtained from Massiot *et al.*[1].

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import ST1_VAS
from mrsimulator import signal_processing as sp
```

Generate the site and spin system objects.

```
Rb87_1 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-27.4,  # in ppm
    quadrupolar={"Cq": 1.68e6, "eta": 0.2},  # Cq is in Hz
)
Rb87_2 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-28.5,  # in ppm
    quadrupolar={"Cq": 1.94e6, "eta": 1.0},  # Cq is in Hz
)
Rb87_3 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-31.3,  # in ppm
    quadrupolar={"Cq": 1.72e6, "eta": 0.5},  # Cq is in Hz
)
```

(continues on next page)

[1] Massiot, D., Touzoa, B., Trumeaua, D., Coutures, J.P., Virlet, J., Florian, P., Grandinetti, P.J. Two-dimensional magic-angle spinning isotropic reconstruction sequences for quadrupolar nuclei, ssnmr, (1996), **6**, *1*, 73-83. DOI: 10.1016/0926-2040(95)01210-9

(continued from previous page)

```
sites = [Rb87_1, Rb87_2, Rb87_3]  # all sites
spin_systems = [SpinSystem(sites=[s]) for s in sites]
```

Step 2: Select a satellite-transition variable-angle spinning method. The following *ST1_ VAS* method correlates the frequencies from the two inner-satellite transitions to the central transition. Note, STMAS measurements are highly suspectable to rotor angle mismatch. In the following, we show two methods, first set to magic-angle and the second deliberately miss-sets by approximately 0.0059 degrees.

```
angles = [54.7359, 54.73]
method = []
for angle in angles:
    method.append(
        ST1_VAS(
            channels=["87Rb"],
            magnetic_flux_density=7,  # in T
            rotor_angle=angle * 3.14159 / 180,  # in rad (magic angle)
            spectral_dimensions=[
                {
                    "count": 256,
                    "spectral_width": 3e3,  # in Hz
                    "reference_offset": -2.4e3,  # in Hz
                    "label": "Isotropic dimension",
                },
                {
                    "count": 512,
                    "spectral_width": 5e3,  # in Hz
                    "reference_offset": -4e3,  # in Hz
                    "label": "MAS dimension",
                },
            ],
        )
    )
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = method  # add the methods.
sim.run()
```

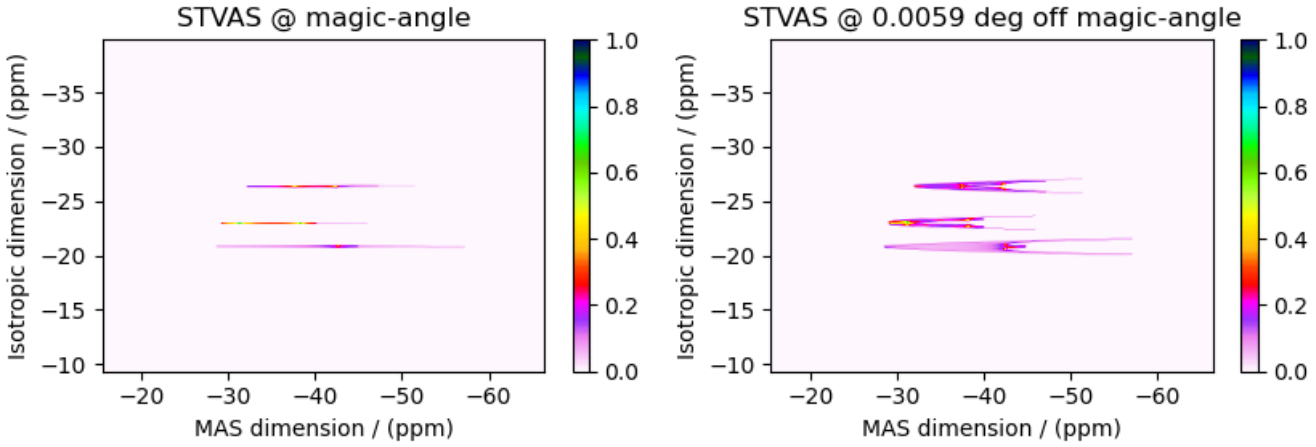
The plot of the simulation.

```
data = [sim.methods[0].simulation, sim.methods[1].simulation]
fig, ax = plt.subplots(1, 2, figsize=(8.5, 3), subplot_kw={"projection": "csdm"})

titles = ["STVAS @ magic-angle", "STVAS @ 0.0059 deg off magic-angle"]
for i, item in enumerate(data):
    cb1 = ax[i].imshow(item / item.max(), aspect="auto", cmap="gist_ncar_r")
    ax[i].set_title(titles[i])
    plt.colorbar(cb1, ax=ax[i])
    ax[i].invert_xaxis()
```

(continues on next page)

(continued from previous page)

```
    ax[i].invert_yaxis()
plt.tight_layout()
plt.show()
```

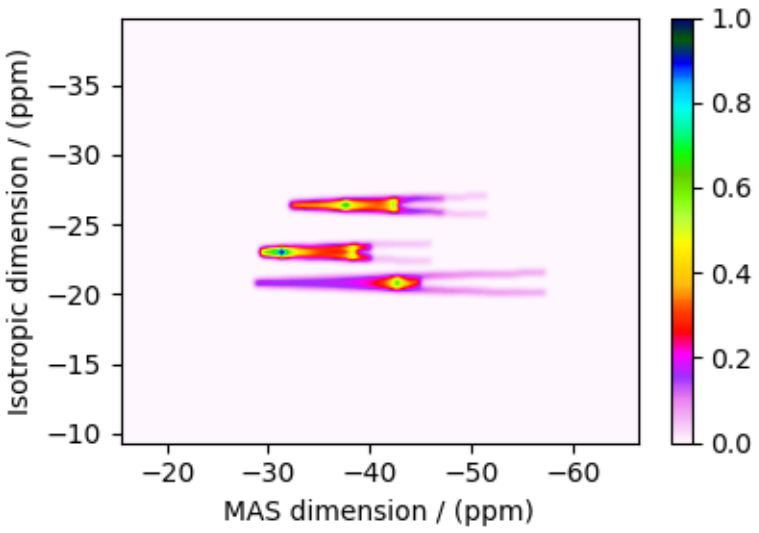


Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="50 Hz", dim_index=0),
        sp.apodization.Gaussian(FWHM="50 Hz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = []
for item in data:
    processed_data.append(processor.apply_operations(data=item))
    processed_data[-1] /= processed_data[-1].max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data[1].real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.775 seconds)

11.3.4 Rb_2SO_4, ^{87}Rb (I=3/2) SAS

^{87}Rb (I=3/2) Switched-angle spinning (SAS) simulation.

The following is an example of Switched-Angle Spinning (SAS) simulation of Rb_2SO_4, which has two distinct rubidium sites. The NMR tensor parameters for these sites are taken from Shore *et al.*[1].

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
```

Generate the site and spin system objects.

```
sites = [
    Site(
        isotope="87Rb",
        isotropic_chemical_shift=16,  # in ppm
        quadrupolar={"Cq": 5.3e6, "eta": 0.1},  # Cq in Hz
    ),
    Site(
        isotope="87Rb",
        isotropic_chemical_shift=40,  # in ppm
        quadrupolar={"Cq": 2.6e6, "eta": 1.0},  # Cq in Hz
    ),
]
spin_systems = [SpinSystem(sites=[s]) for s in sites]
```

Use the generic 2D method, *Method2D*, to simulate a SAS spectrum by customizing the method parameters, as shown below. Note, the Method2D method simulates an infinite spinning speed spectrum.

[1] Shore, J.S., Wang, S.H., Taylor, R.E., Bell, A.T., Pines, A. Determination of quadrupolar and chemical shielding tensors using solid-state two-dimensional NMR spectroscopy, J. Chem. Phys. (1996) **105** *21*, 9412. DOI: 10.1063/1.472776

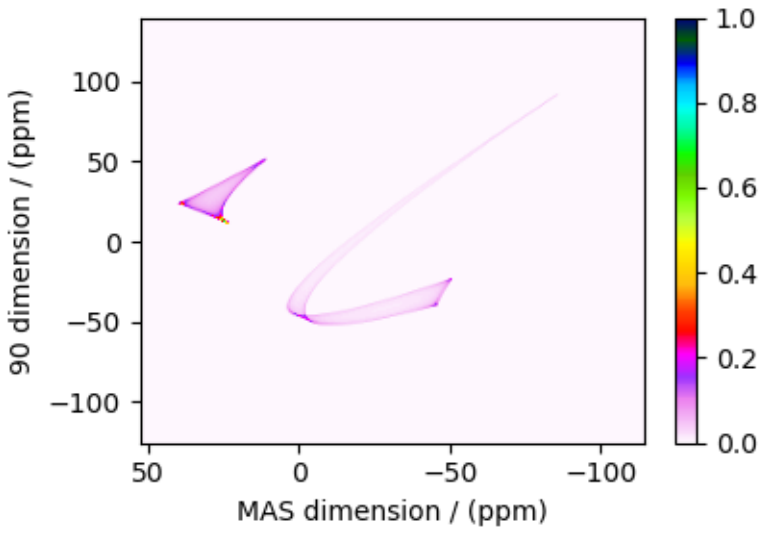
```
sas = Method2D(
    channels=["87Rb"],
    magnetic_flux_density=9.4,  # in T
    spectral_dimensions=[
        {
            "count": 256,
            "spectral_width": 3.5e4,  # in Hz
            "reference_offset": 1e3,  # in Hz
            "label": "90 dimension",
            "events": [
                {
                    "rotor_angle": 90 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],  # in radians
        },
        {
            "count": 256,
            "spectral_width": 22e3,  # in Hz
            "reference_offset": -4e3,  # in Hz
            "label": "MAS dimension",
            "events": [
                {
                    "rotor_angle": 54.74 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],  # in radians
        },
    ],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [sas]  # add the method.
sim.run()
```

The plot of the simulation.

```
data = sim.methods[0].simulation

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```

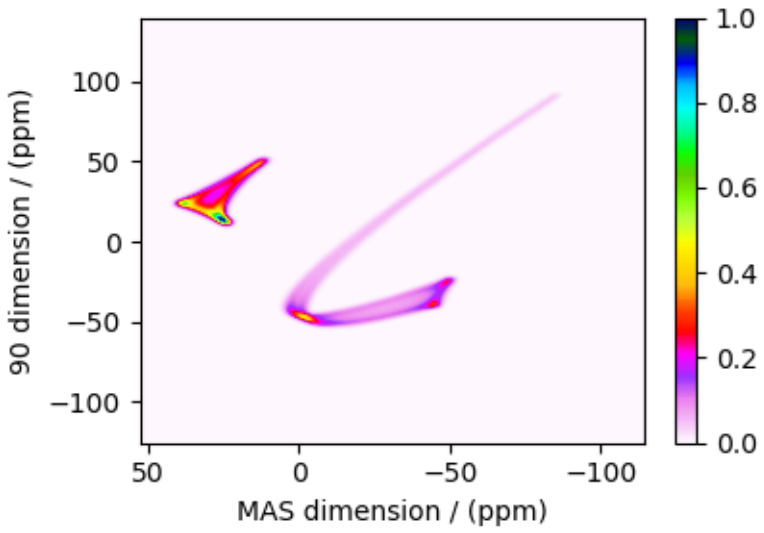


Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.4 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.4 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=data)
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.474 seconds)

11.3.5 Rb_2CrO_4, ^{87}Rb (I=3/2) SAS

^{87}Rb (I=3/2) Switched-angle spinning (SAS) simulation.

The following is a Switched-Angle Spinning (SAS) simulation of Rb_2CrO_4. While Rb_2CrO_4 has two rubidium sites, the site with the smaller quadrupolar interaction was selectively observed and reported by Shore *et al.*[1]. The following is the simulation based on the published tensor parameters.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
```

Generate the site and spin system objects.

```
site = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-7,  # in ppm
    shielding_symmetric={"zeta": 110, "eta": 0},
    quadrupolar={
        "Cq": 3.5e6,  # in Hz
        "eta": 0.3,
        "alpha": 0,  # in rads
        "beta": 70 * 3.14159 / 180,  # in rads
        "gamma": 0,  # in rads
    },
)
spin_system = SpinSystem(sites=[site])
```

[1] Shore, J.S., Wang, S.H., Taylor, R.E., Bell, A.T., Pines, A. Determination of quadrupolar and chemical shielding tensors using solid-state two-dimensional NMR spectroscopy, J. Chem. Phys. (1996) **105** *21*, 9412. DOI: 10.1063/1.472776

Use the generic 2D method, *Method2D*, to simulate a SAS spectrum by customizing the method parameters, as shown below. Note, the Method2D method simulates an infinite spinning speed spectrum.

```
sas = Method2D(
    channels=["87Rb"],
    magnetic_flux_density=4.2,  # in T
    spectral_dimensions=[
        {
            "count": 256,
            "spectral_width": 1.5e4,  # in Hz
            "reference_offset": -5e3,  # in Hz
            "label": "70.12 dimension",
            "events": [
                {
                    "rotor_angle": 70.12 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],  # in radians
        },
        {
            "count": 512,
            "spectral_width": 15e3,  # in Hz
            "reference_offset": -7e3,  # in Hz
            "label": "MAS dimension",
            "events": [
                {
                    "rotor_angle": 54.74 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],  # in radians
        },
    ],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = [spin_system]  # add the spin systems
sim.methods = [sas]  # add the method.

# Configure the simulator object. For non-coincidental tensors, set the value of the
# `integration_volume` attribute to `hemisphere`.
sim.config.integration_volume = "hemisphere"
sim.run()
```

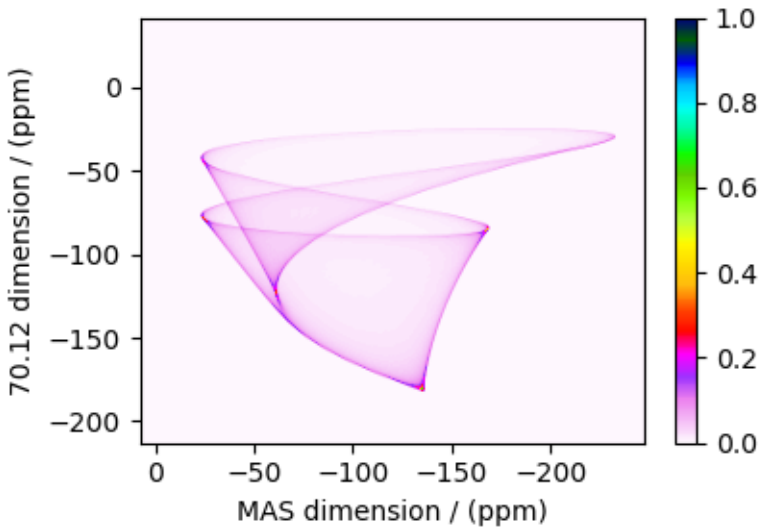
The plot of the simulation.

```
data = sim.methods[0].simulation

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
```

(continues on next page)

(continued from previous page)

```
plt.tight_layout()
plt.show()
```

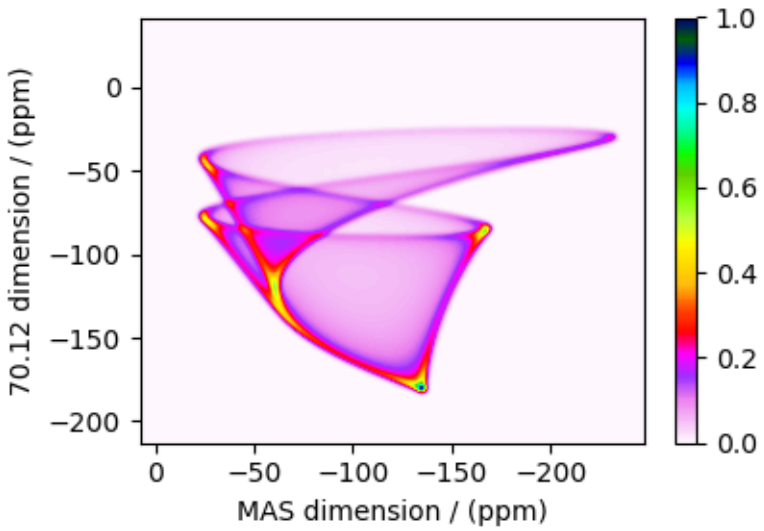


Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.2 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.2 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=data)
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.invert_xaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.555 seconds)

11.3.6 Coesite, ^{17}O (I=5/2) 3QMAS

^{17}O (I=5/2) 3QMAS simulation.

The following is a triple quantum magic angle spinning (3QMAS) simulation of Coesite. The NMR EFG tensor parameters for ^{17}O sites in coesite is obtained from Grandinetti *et al.*[1]

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator
from mrsimulator.methods import ThreeQ_VAS
from mrsimulator import signal_processing as sp
```

Create the Simulator object and load the spin systems database or url address.

```
sim = Simulator()

# load the spin systems from url.
filename = "https://sandbox.zenodo.org/record/687656/files/coesite.mrsys"
sim.load_spin_systems(filename)

method = ThreeQ_VAS(
    channels=["17O"],
    magnetic_flux_density=11.74,  # in T
    spectral_dimensions=[
        {
            "count": 256,
            "spectral_width": 5e3,  # in Hz
            "reference_offset": -2.5e3,  # in Hz
            "label": "Isotropic dimension",
```

(continues on next page)

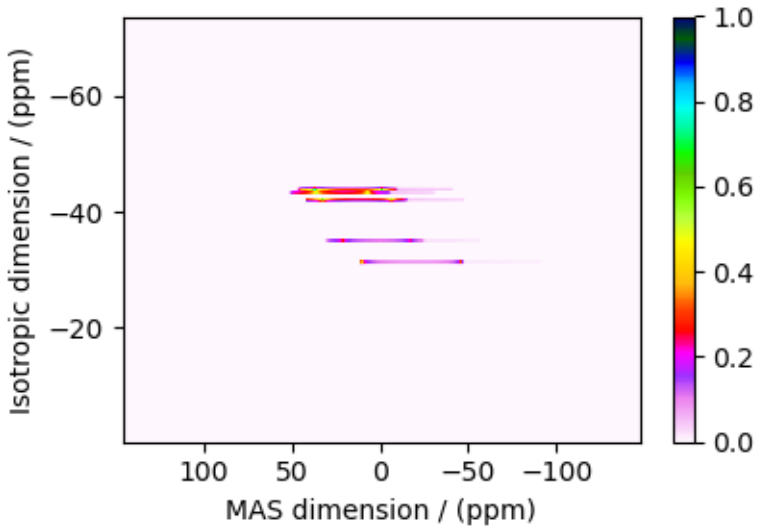
[1] Grandinetti, P. J., Baltisberger, J. H., Farnan, I., Stebbins, J. F., Werner, U. and Pines, A. Solid-State ^{17}O Magic-Angle and Dynamic-Angle Spinning NMR Study of the SiO_2 Polymorph Coesite, J. Phys. Chem. 1995, **99**, *32*, 12341-12348. DOI: 10.1021/j100032a045

(continued from previous page)

```
        },
        # The last spectral dimension block is the direct-dimension
        {
            "count": 256,
            "spectral_width": 2e4,  # in Hz
            "reference_offset": 0,  # in Hz
            "label": "MAS dimension",
        },
    ],
)
sim.methods = [method]  # add the method.
sim.run()  # Run the simulation
```

The plot of the simulation.

```
data = sim.methods[0].simulation

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```



Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.3 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.15 kHz", dim_index=1),
```

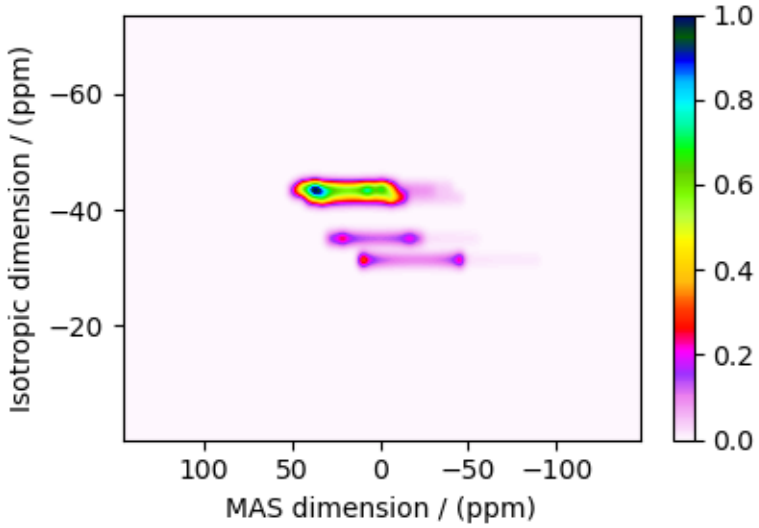
(continues on next page)

(continued from previous page)

```
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=data)
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 1.334 seconds)

11.3.7 Coesite, ^{17}O (I=5/2) DAS

^{17}O (I=5/2) Dynamic-angle spinning (DAS) simulation.

The following is a Dynamic Angle Spinning (DAS) simulation of Coesite. Coesite has five crystallographic ^{17}O sites. In the following, we use the ^{17}O EFG tensor information from Grandinetti *et al.*[1]

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
```

[1] Grandinetti, P. J., Baltisberger, J. H., Farnan, I., Stebbins, J. F., Werner, U. and Pines, A. Solid-State ^{17}O Magic-Angle and Dynamic-Angle Spinning NMR Study of the SiO_2 Polymorph Coesite, J. Phys. Chem. 1995, **99**, *32*, 12341-12348. DOI: 10.1021/j100032a045

Create the Simulator object and load the spin systems database or url address.

```
sim = Simulator()

# load the spin systems from url.
filename = "https://sandbox.zenodo.org/record/687656/files/coesite.mrsys"
sim.load_spin_systems(filename)
```

Use the generic 2D method, *Method2D*, to simulate a DAS spectrum by customizing the method parameters, as shown below. Note, the Method2D method simulates an infinite spinning speed spectrum.

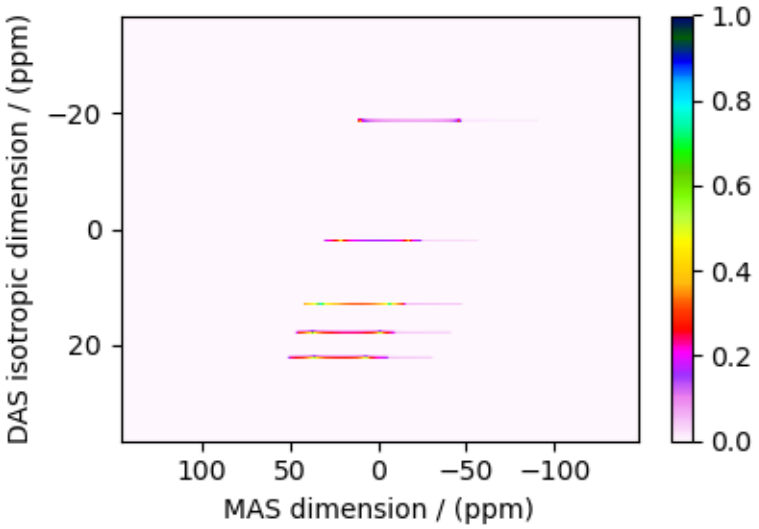
```
das = Method2D(
    channels=["17O"],
    magnetic_flux_density=11.74,  # in T
    spectral_dimensions=[
        {
            "count": 256,
            "spectral_width": 5e3,  # in Hz
            "reference_offset": 0,  # in Hz
            "label": "DAS isotropic dimension",
            "events": [
                {
                    "fraction": 0.5,
                    "rotor_angle": 37.38 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                },
                {
                    "fraction": 0.5,
                    "rotor_angle": 79.19 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                },
            ],
        },
        # The last spectral dimension block is the direct-dimension
        {
            "count": 256,
            "spectral_width": 2e4,  # in Hz
            "reference_offset": 0,  # in Hz
            "label": "MAS dimension",
            "events": [
                {
                    "rotor_angle": 54.735 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],
        },
    ],
)
sim.methods = [das]  # add the method.
```

Run the simulation

```
sim.run()
```

The plot of the simulation.

```
data = sim.methods[0].simulation

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```

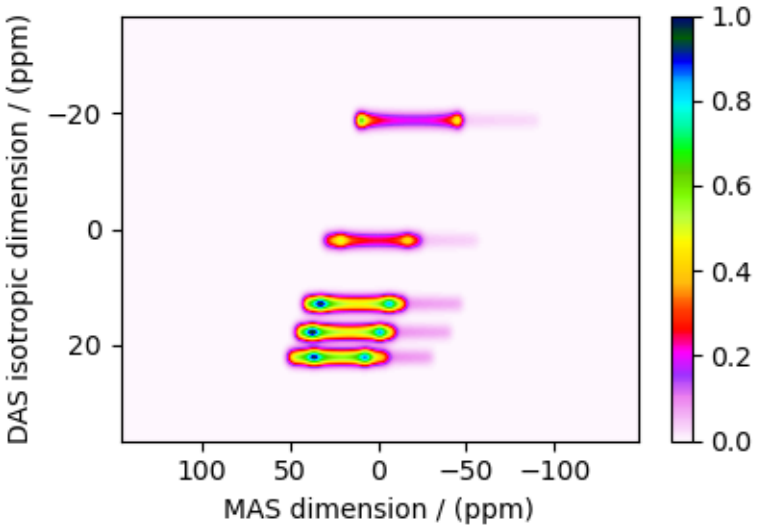


Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.3 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.15 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=data)
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.568 seconds)

11.3.8 Rb_2CrO_4, ^{87}Rb (I=3/2) COASTER

^{87}Rb (I=3/2) Correlation of anisotropies separated through echo refocusing (COASTER) simulation.

The following is a Correlation of Anisotropies Separated Through Echo Refocusing (COASTER) simulation of Rb_2CrO_4. The Rb site with the smaller quadrupolar interaction is selectively observed and reported by Ash *et al.*[1]. The following is the simulation based on the published tensor parameters.

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
```

Generate the site and spin system objects.

```
site = Site(
    isotope="87Rb",
    isotropic_chemical_shift=-9,  # in ppm
    shielding_symmetric={"zeta": 110, "eta": 0},
    quadrupolar={
        "Cq": 3.5e6,  # in Hz
        "eta": 0.36,
        "alpha": 0,  # in rads
        "beta": 70 * 3.14159 / 180,  # in rads
        "gamma": 0,  # in rads
    },
)
spin_system = SpinSystem(sites=[site])
```

[1] Jason T. Ash, Nicole M. Trease, and Philip J. Grandinetti. Separating Chemical Shift and Quadrupolar Anisotropies via Multiple-Quantum NMR Spectroscopy, J. Am. Chem. Soc. (2008) **130**, 10858-10859. DOI: 10.1021/ja802865x

Use the generic 2D method, *Method2D*, to simulate a COASTER spectrum by customizing the method parameters, as shown below. Note, the Method2D method simulates an infinite spinning speed spectrum.

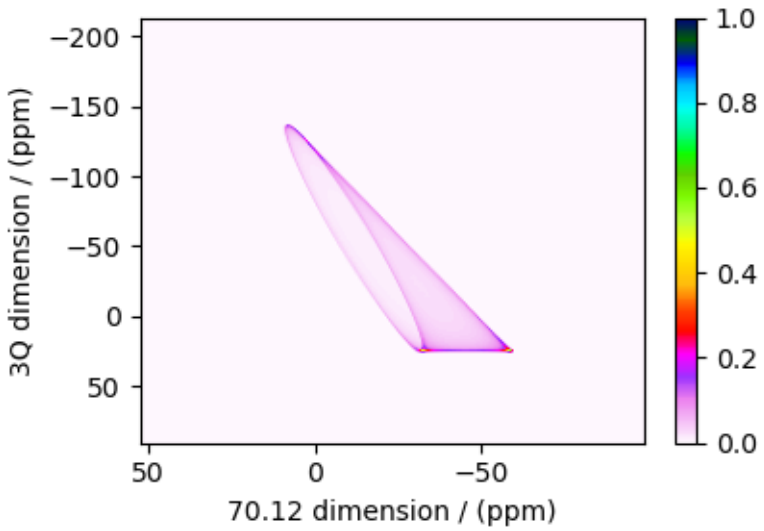
```
coaster = Method2D(
    channels=["87Rb"],
    magnetic_flux_density=9.4,  # in T
    rotor_angle=70.12 * 3.14159 / 180,  # in rads
    spectral_dimensions=[
        {
            "count": 256,
            "spectral_width": 4e4,  # in Hz
            "reference_offset": -8e3,  # in Hz
            "label": "3Q dimension",
            "events": [{"transition_query": {"P": [3], "D": [0]}}],
        },
        # The last spectral dimension block is the direct-dimension
        {
            "count": 256,
            "spectral_width": 2e4,  # in Hz
            "reference_offset": -3e3,  # in Hz
            "label": "70.12 dimension",
            "events": [{"transition_query": {"P": [-1], "D": [0]}}],
        },
    ],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = [spin_system]  # add the spin systems
sim.methods = [coaster]  # add the method.

# configure the simulator object. For non-coincidental tensors, set the value of the
# `integration_volume` attribute to `hemisphere`.
sim.config.integration_volume = "hemisphere"
sim.run()
```

The plot of the simulation.

```
data = sim.methods[0].simulation

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```

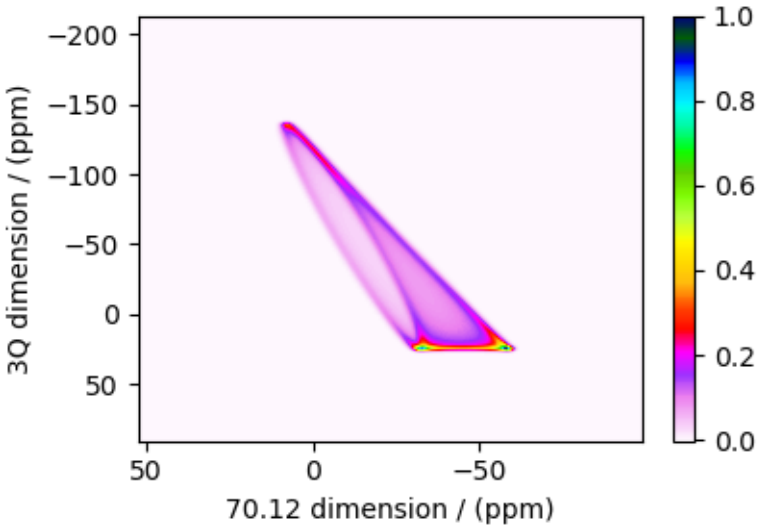


Add post-simulation signal processing.

```
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.3 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.3 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=data)
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.real, cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.507 seconds)

11.3.9 Itraconazole, ^{13}C (I=1/2) PASS

^{13}C (I=1/2) 2D Phase-adjusted spinning sideband (PASS) simulation.

The following is a simulation of a 2D PASS spectrum of itraconazole, a triazole containing drug prescribed for the prevention and treatment of fungal infection. The 2D PASS spectrum is a correlation of finite speed MAS to an infinite speed MAS spectrum. The parameters for the simulation are obtained from Dey *et al.*[1].

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator
from mrsimulator.methods import SSB2D
from mrsimulator import signal_processing as sp
```

There are 41 ^{13}C single-site spin systems partially describing the NMR parameters of itraconazole. We will import the directly import the spin systems to the Simulator object using the *load_spin_systems* method.

```
sim = Simulator()

filename = "https://sandbox.zenodo.org/record/687656/files/itraconazole_13C.mrsys"
sim.load_spin_systems(filename)
```

Use the `SSB2D` method to simulate a PASS, MAT, QPASS, QMAT, or any equivalent sideband separation spectrum. Here, we use the method to generate a PASS spectrum.

```
PASS = SSB2D(
    channels=["13C"],
    magnetic_flux_density=11.74,
    rotor_frequency=2000,
    spectral_dimensions=[
```

(continues on next page)

[1] Dey, K .K, Gayen, S., Ghosh, M., Investigation of the Detailed Internal Structure and Dynamics of Itraconazole by Solid-State NMR Measurements, ACS Omega (2019) **4**, 21627. DOI:10.1021/acsomega.9b03558

(continued from previous page)

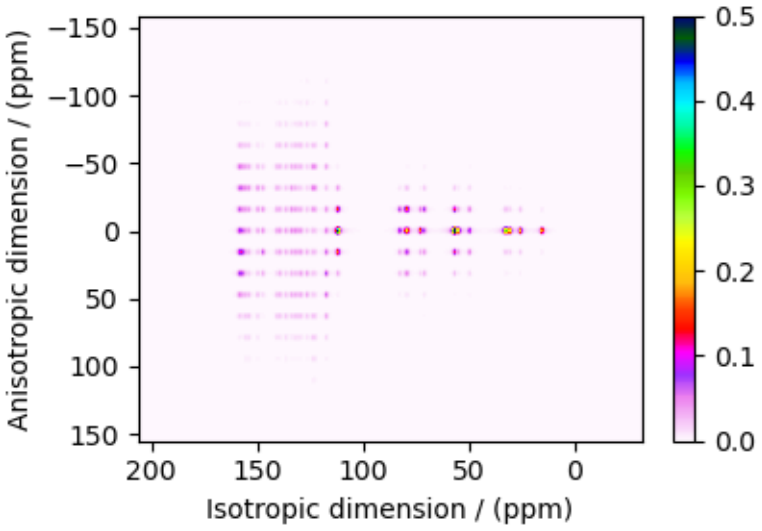
```
        {
            "count": 20 * 4,
            "spectral_width": 2000 * 20,  # value in Hz
            "label": "Anisotropic dimension",
        },
        {
            "count": 1024,
            "spectral_width": 3e4,  # value in Hz
            "reference_offset": 1.1e4,  # value in Hz
            "label": "Isotropic dimension",
        },
    ],
)
sim.methods = [PASS]  # add the method.

# For 2D spinning sideband simulation, set the number of spinning sidebands in the
# Simulator.config object to `spectral_width/rotor_frequency` along the sideband
# dimension.
sim.config.number_of_sidebands = 20

# run the simulation.
sim.run()
```

Apply post-simulation processing. Here, we apply a Lorentzian line broadening to the isotropic dimension.

```
data = sim.methods[0].simulation
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(dim_index=0),
        sp.apodization.Exponential(FWHM="100 Hz", dim_index=0),
        sp.FFT(dim_index=0),
    ]
)
processed_data = processor.apply_operations(data=data).real
processed_data /= processed_data.max()
```

The plot of the simulation.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data, aspect="auto", cmap="gist_ncar_r", vmax=0.5)
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 1.363 seconds)

11.3.10 Rb_2SO_4, ^{87}Rb (I=3/2) QMAT

^{87}Rb (I=3/2) Quadrupolar Magic-angle turning (QMAT) simulation.

The following is a simulation of the QMAT spectrum of Rb_2SiO_4. The 2D QMAT spectrum is a correlation of finite speed MAS to an infinite speed MAS spectrum. The parameters for the simulation are obtained from Walder *et al.*[1].

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import SSB2D
```

Generate the site and spin system objects.

```
sites = [
    Site(
        isotope="87Rb",
        isotropic_chemical_shift=16,  # in ppm
        quadrupolar={"Cq": 5.3e6, "eta": 0.1},  # Cq in Hz
    ),
    Site(
        isotope="87Rb",
        isotropic_chemical_shift=40,  # in ppm
        quadrupolar={"Cq": 2.6e6, "eta": 1.0},  # Cq in Hz
    ),
]
spin_systems = [SpinSystem(sites=[s]) for s in sites]
```

Use the `SSB2D` method to simulate a PASS, MAT, QPASS, QMAT, or any equivalent sideband separation spectrum. Here, we use the method to generate a QMAT spectrum. The QMAT method is created from the `SSB2D` method in

[1] Walder, B. J., Dey, K .K, Kaseman, D. C., Baltisberger, J. H., and Philip J. Grandinetti. Sideband separation experiments in NMR with phase incremented echo train acquisition, J. Chem. Phys. (2013) **138**, 174203. DOI:10.1063/1.4803142

the same as a PASS or MAT method. The difference is that the observed channel is a half-integer quadrupolar spin instead of a spin I=1/2.

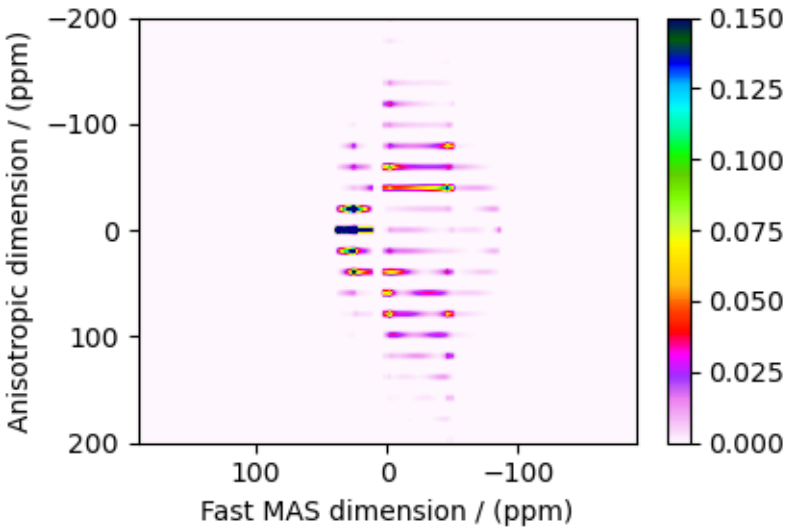
```
qmat = SSB2D(
    channels=["87Rb"],
    magnetic_flux_density=9.4,
    rotor_frequency=2604,
    spectral_dimensions=[
        {
            "count": 32 * 4,
            "spectral_width": 2604 * 32,  # in Hz
            "label": "Anisotropic dimension",
        },
        {
            "count": 512,
            "spectral_width": 50000,  # in Hz
            "label": "Fast MAS dimension",
        },
    ],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [qmat]  # add the method.

# For 2D spinning sideband simulation, set the number of spinning sidebands in the
# Simulator.config object to `spectral_width/rotor_frequency` along the sideband
# dimension.
sim.config.number_of_sidebands = 32
sim.run()
```

The plot of the simulation.

```
plt.figure(figsize=(4.25, 3.0))
data = sim.methods[0].simulation
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), aspect="auto", cmap="gist_ncar_r", vmax=0.15)
plt.colorbar(cb)
ax.invert_xaxis()
ax.set_ylim(200, -200)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.290 seconds)

11.3.11 Wollastonite, ^{29}Si (I=1/2), MAF

^{29}Si (I=1/2) magic angle flipping.

Wollastonite is a high-temperature calcium-silicate, β-$Ca_3Si_3O_9$, with three distinct ^{29}Si sites. The ^{29}Si tensor parameters were obtained from Hansen *et al.*[1]

```
import matplotlib.pyplot as plt

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
```

Create the sites and spin systems

```
sites = [
    Site(
        isotope="29Si",
        isotropic_chemical_shift=-89.0,  # in ppm
        shielding_symmetric={"zeta": 59.8, "eta": 0.62},  # zeta in ppm
    ),
    Site(
        isotope="29Si",
        isotropic_chemical_shift=-89.5,  # in ppm
        shielding_symmetric={"zeta": 52.1, "eta": 0.68},  # zeta in ppm
    ),
    Site(
        isotope="29Si",
        isotropic_chemical_shift=-87.8,  # in ppm
        shielding_symmetric={"zeta": 69.4, "eta": 0.60},  # zeta in ppm
```

(continues on next page)

[1] Hansen, M. R., Jakobsen, H. J., Skibsted, J., ^{29}Si Chemical Shift Anisotropies in Calcium Silicates from High-Field ^{29}Si MAS NMR Spectroscopy, Inorg. Chem. 2003, **42**, 7, 2368-2377. DOI: 10.1021/ic020647f

(continued from previous page)

```
    ),
]

spin_systems = [SpinSystem(sites=[s]) for s in sites]
```

Use the generic 2D method, *Method2D*, to simulate a Magic-Angle Flipping (MAF) spectrum by customizing the method parameters, as shown below. Note, the Method2D method simulates an infinite spinning speed spectrum.

```
maf = Method2D(
    channels=["29Si"],
    magnetic_flux_density=14.1,  # in T
    spectral_dimensions=[
        {
            "count": 128,
            "spectral_width": 2e4,  # in Hz
            "label": "Anisotropic dimension",
            "events": [
                {
                    "rotor_angle": 90 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],
        },
        {
            "count": 128,
            "spectral_width": 3e3,  # in Hz
            "reference_offset": -1.05e4,  # in Hz
            "label": "Isotropic dimension",
            "events": [
                {
                    "rotor_angle": 54.735 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],
        },
    ],
    affine_matrix=[[1, -1], [0, 1]],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [maf]  # add the method
sim.run()
```

Add post-simulation signal processing.

```
csdm_data = sim.methods[0].simulation
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(dim_index=(0, 1)),
```

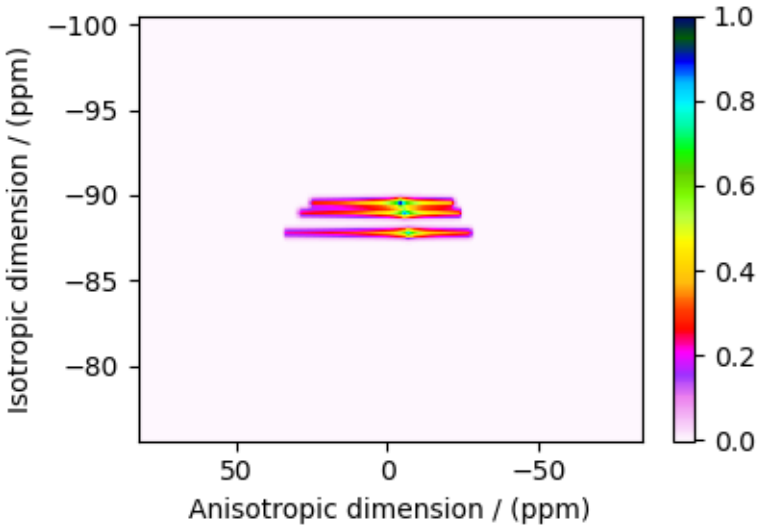
(continues on next page)

(continued from previous page)

```
        sp.apodization.Gaussian(FWHM="50 Hz", dim_index=0),
        sp.apodization.Gaussian(FWHM="50 Hz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=csdm_data).real
processed_data /= processed_data.max()
```

The plot of the simulation after signal processing.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(processed_data.T, aspect="auto", cmap="gist_ncar_r")
plt.colorbar(cb)
ax.invert_xaxis()
ax.invert_yaxis()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.274 seconds)

11.3.12 $MCl_2.2D_2O$, ^2H (I=1) Shifting-d echo

^2H (I=1) 2D NMR CSA-Quad 1st order correlation spectrum simulation.

The following is a static shifting-d echo NMR correlation simulation of $MCl_2 \cdot 2D_2O$ crystalline solid, where $M \in$ [Cu, Ni, Co, Fe, Mn]. The tensor parameters for the simulation and the corresponding spectrum are reported by Walder *et al.*[1].

```
import matplotlib.pyplot as plt
```

(continues on next page)

[1] Walder B.J, Patterson A.M., Baltisberger J.H, and Grandinetti P.J Hydrogen motional disorder in crystalline iron group chloride dihydrates spectroscopy, J. Chem. Phys. (2018) **149**, 084503. DOI: 10.1063/1.5037151

(continued from previous page)

```
from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
```

Generate the site and spin system objects.

```
site_Ni = Site(
    isotope="2H",
    isotropic_chemical_shift=-97,  # in ppm
    shielding_symmetric={
        "zeta": -551,  # in ppm
        "eta": 0.12,
        "alpha": 62 * 3.14159 / 180,  # in rads
        "beta": 114 * 3.14159 / 180,  # in rads
        "gamma": 171 * 3.14159 / 180,  # in rads
    },
    quadrupolar={"Cq": 77.2e3, "eta": 0.9},  # Cq in Hz
)

site_Cu = Site(
    isotope="2H",
    isotropic_chemical_shift=51,  # in ppm
    shielding_symmetric={
        "zeta": 146,  # in ppm
        "eta": 0.84,
        "alpha": 95 * 3.14159 / 180,  # in rads
        "beta": 90 * 3.14159 / 180,  # in rads
        "gamma": 0 * 3.14159 / 180,  # in rads
    },
    quadrupolar={"Cq": 118.2e3, "eta": 0.86},  # Cq in Hz
)

site_Co = Site(
    isotope="2H",
    isotropic_chemical_shift=215,  # in ppm
    shielding_symmetric={
        "zeta": -1310,  # in ppm
        "eta": 0.23,
        "alpha": 180 * 3.14159 / 180,  # in rads
        "beta": 90 * 3.14159 / 180,  # in rads
        "gamma": 90 * 3.14159 / 180,  # in rads
    },
    quadrupolar={"Cq": 114.6e3, "eta": 0.95},  # Cq in Hz
)

site_Fe = Site(
    isotope="2H",
    isotropic_chemical_shift=101,  # in ppm
    shielding_symmetric={
        "zeta": -1187,  # in ppm
        "eta": 0.4,
        "alpha": 122 * 3.14159 / 180,  # in rads
```

(continues on next page)

(continued from previous page)

```
        "beta": 90 * 3.14159 / 180,  # in rads
        "gamma": 90 * 3.14159 / 180,  # in rads
    },
    quadrupolar={"Cq": 114.2e3, "eta": 0.98},  # Cq in Hz
)

site_Mn = Site(
    isotope="2H",
    isotropic_chemical_shift=145,  # in ppm
    shielding_symmetric={
        "zeta": -1236,  # in ppm
        "eta": 0.23,
        "alpha": 136 * 3.14159 / 180,  # in rads
        "beta": 90 * 3.14159 / 180,  # in rads
        "gamma": 90 * 3.14159 / 180,  # in rads
    },
    quadrupolar={"Cq": 1.114e5, "eta": 1.0},  # Cq in Hz
)

spin_systems = [
    SpinSystem(sites=[s], name=f"{n}Cl$_2$.2D$_2$O")
    for s, n in zip(
        [site_Ni, site_Cu, site_Co, site_Fe, site_Mn], ["Ni", "Cu", "Co", "Fe", "Mn"]
    )
]
```

Use the generic 2D method, *Method2D*, to generate a shifting-d echo method. The reported shifting-d 2D sequence is a correlation of the shielding frequencies to the first-order quadrupolar frequencies. Here, we create a correlation method using the `freq_contrib` attribute, which acts as a switch for including the frequency contributions from interaction during the event.

In the following method, we assign the `["Quad1_2"]` and `["Shielding1_0", "Shielding1_2"]` as the value to the `freq_contrib` key. The *Quad1_2* is an enumeration for selecting the first-order second-rank quadrupolar frequency contributions. *Shielding1_0* and *Shielding1_2* are enumerations for the first-order shielding with zeroth and second-rank tensor contributions, respectively. See *FrequencyEnum* (page 316) for details.

```
shifting_d = Method2D(
    channels=["2H"],
    magnetic_flux_density=9.395,  # in T
    spectral_dimensions=[
        {
            "count": 512,
            "spectral_width": 2.5e5,  # in Hz
            "label": "Quadrupolar frequency",
            "events": [
                {
                    "rotor_frequency": 0,
                    "transition_query": {"P": [-1]},
                    "freq_contrib": ["Quad1_2"],
                }
            ],
        },
```

(continues on next page)

(continued from previous page)

```
        {
            "count": 256,
            "spectral_width": 2e5,  # in Hz
            "reference_offset": 2e4,  # in Hz
            "label": "Paramagnetic shift",
            "events": [
                {
                    "rotor_frequency": 0,
                    "transition_query": {"P": [-1]},
                    "freq_contrib": ["Shielding1_0", "Shielding1_2"],
                }
            ],
        },
    ],
)
```

Create the Simulator object, add the method and spin system objects, and run the simulation.

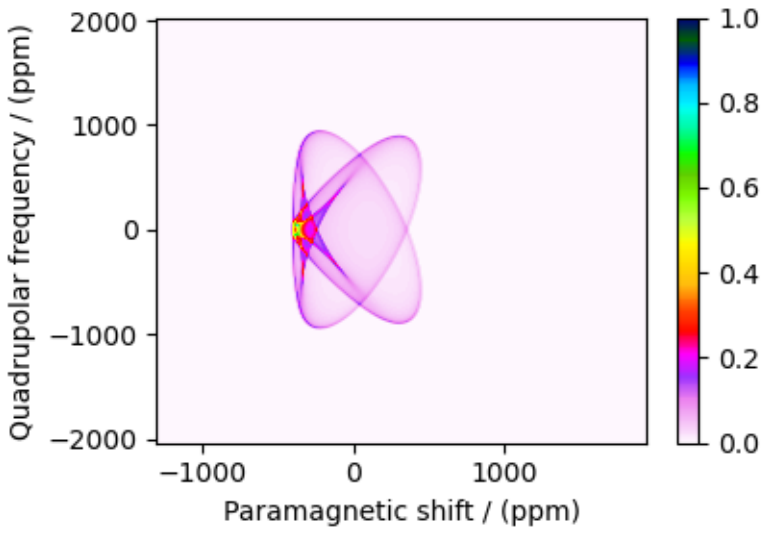
```
sim = Simulator(spin_systems=spin_systems, methods=[shifting_d])
# Configure the simulator object. For non-coincidental tensors, set the value of the
# `integration_volume` attribute to `hemisphere`.
sim.config.integration_volume = "hemisphere"
sim.config.decompose_spectrum = "spin_system"  # simulate spectra per spin system
sim.run()
```

Add post-simulation signal processing.

```
data = sim.methods[0].simulation
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="9 kHz", dim_index=0),  # along dimension 0
        sp.apodization.Gaussian(FWHM="9 kHz", dim_index=1),  # along dimension 1
        sp.FFT(dim_index=(0, 1)),
    ]
)
processed_data = processor.apply_operations(data=data).real
```

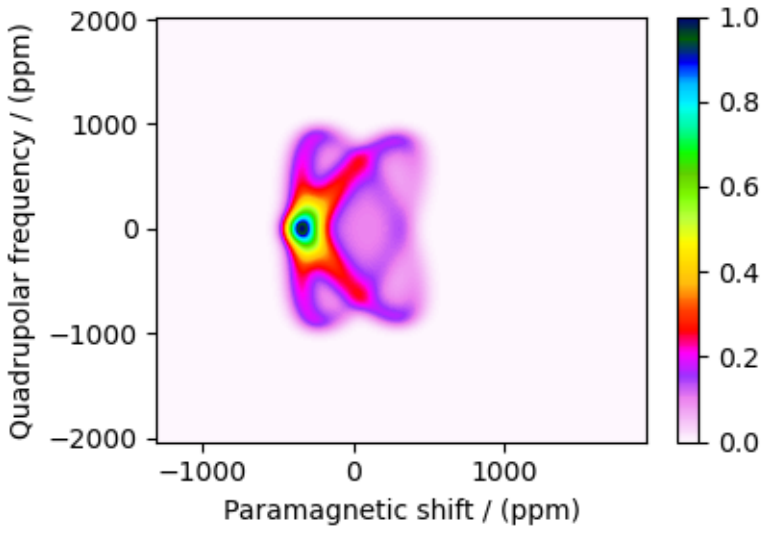
The plot of the simulation. Because we configured the simulator object to simulate spectrum per spin system, the following data is a CSDM object containing five simulations (dependent variables). Let's visualize the first data corresponding to $NiCl_2 \cdot 2D_2O$.

```
data_Ni = data.split()[0]

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data_Ni / data_Ni.max(), aspect="auto", cmap="gist_ncar_r")
plt.title(None)
plt.colorbar(cb)
plt.tight_layout()
plt.show()
```



The plot of the simulation after signal processing.

```
proc_data_Ni = processed_data.split()[0]

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(proc_data_Ni / proc_data_Ni.max(), cmap="gist_ncar_r", aspect="auto")
plt.title(None)
plt.colorbar(cb)
plt.tight_layout()
plt.show()
```



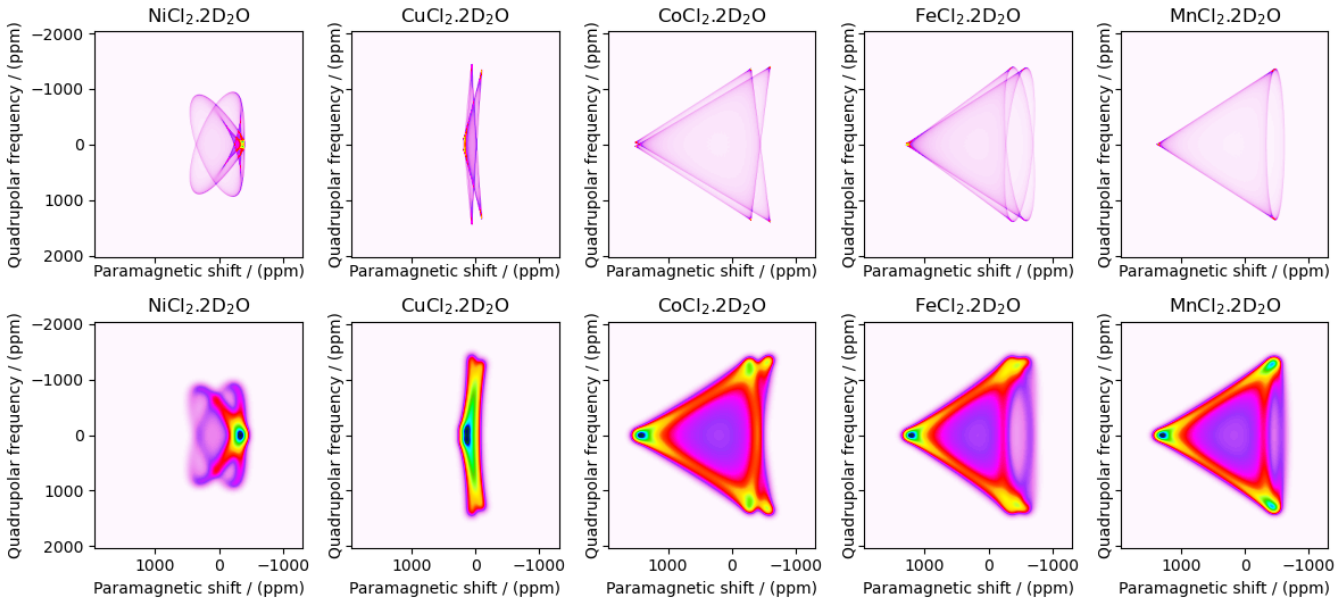
Let's plot all the simulated datasets.

```
fig, ax = plt.subplots(
    2, 5, sharex=True, sharey=True, figsize=(12, 5.5), subplot_kw={"projection": "csdm"}
```

(continues on next page)

(continued from previous page)

```
)
for i, data_obj in enumerate([data, processed_data]):
    for j, datum in enumerate(data_obj.split()):
        ax[i, j].imshow(datum / datum.max(), aspect="auto", cmap="gist_ncar_r")
        ax[i, j].invert_xaxis()
        ax[i, j].invert_yaxis()

plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 1.756 seconds)

11.4 2D NMR simulation (Disordered/Amorphous solids)

The following examples are the NMR spectrum simulation for amorphous solids. The examples include the illustrations for the following methods:

- Triple-quantum variable-angle spinning (*ThreeQ_VAS()* (page 325))

11.4.1 Simulating site disorder (crystalline)

^{87}Rb (I=3/2) 3QMAS simulation with site disorder.

The following example illustrates an NMR simulation of a crystalline solid with site disorders. We model such disorders with Extended Czjzek distribution. The following case study shows an ^{87}Rb 3QMAS simulation of RbNO3.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
```

(continues on next page)

(continued from previous page)

```
from mrsimulator import Simulator
from mrsimulator.methods import ThreeQ_VAS
from mrsimulator.models import ExtCzjzekDistribution
from mrsimulator.utils.collection import single_site_system_generator
```

Generate probability distribution

Create three extended Czjzek distributions for the three sites in RbNO3 about their respective mean tensors.

```
# The range of isotropic chemical shifts, the quadrupolar coupling constant, and
# asymmetry parameters used in generating a 3D grid.
iso_r = np.arange(101) / 6.5 - 35  # in ppm
Cq_r = np.arange(100) / 100 + 1.25  # in MHz
eta_r = np.arange(11) / 10

# The 3D mesh grid over which the distribution amplitudes are evaluated.
iso, Cq, eta = np.meshgrid(iso_r, Cq_r, eta_r, indexing="ij")


def get_prob_dist(iso, Cq, eta, eps, cov):
    pdf = 0
    for i in range(len(iso)):
        # The 2D amplitudes for Cq and eta is sampled from the extended Czjzek model.
        avg_tensor = {"Cq": Cq[i], "eta": eta[i]}
        _, _, amp = ExtCzjzekDistribution(avg_tensor, eps=eps[i]).pdf(pos=[Cq_r, eta_r])

        # The 1D amplitudes for isotropic chemical shifts is sampled as a Gaussian.
        iso_amp = multivariate_normal(mean=iso[i], cov=[cov[i]]).pdf(iso_r)

        # The 3D amplitude grid is generated as an uncorrelated distribution of the
        # above two distribution, which is the product of the two distributions.
        pdf_t = np.repeat(amp, iso_r.size).reshape(eta_r.size, Cq_r.size, iso_r.size)
        pdf_t *= iso_amp
        pdf += pdf_t
    return pdf


iso_0 = [-27.4, -28.5, -31.3]  # isotropic chemical shifts for the three sites in ppm
Cq_0 = [1.68, 1.94, 1.72]  # Cq in MHz for the three sites
eta_0 = [0.2, 1, 0.5]  # eta for the three sites
eps_0 = [0.02, 0.02, 0.02]  # perturbation fractions for extended Czjzek distribution.
var_0 = [0.1, 0.1, 0.1]  # variance for the isotropic chemical shifts in ppm^2.

pdf = get_prob_dist(iso_0, Cq_0, eta_0, eps_0, var_0).T
```

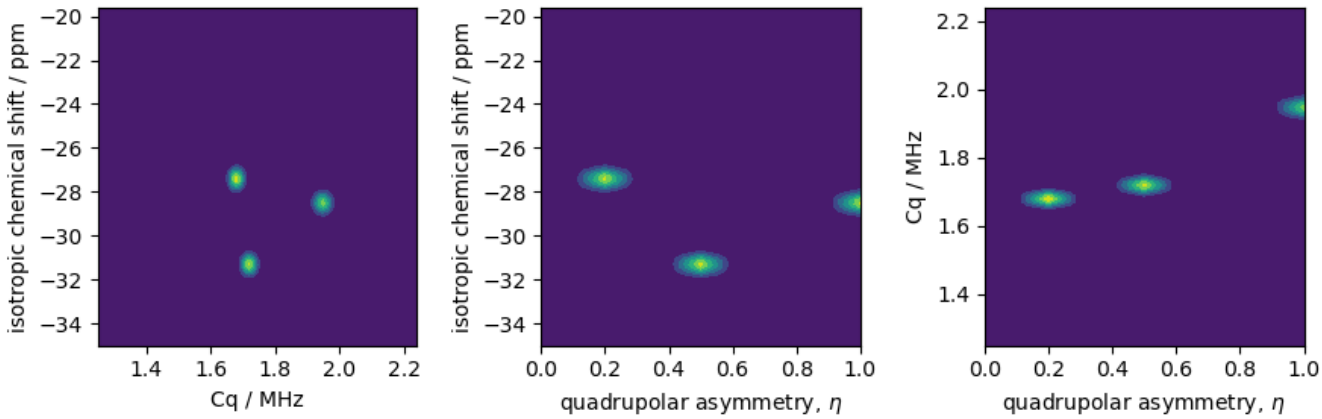
The two-dimensional projections from this three-dimensional distribution are shown below.

```
_, ax = plt.subplots(1, 3, figsize=(9, 3))

# isotropic shift v.s. quadrupolar coupling constant
ax[0].contourf(Cq_r, iso_r, pdf.sum(axis=2))
```

(continues on next page)

(continued from previous page)

```
ax[0].set_xlabel("Cq / MHz")
ax[0].set_ylabel("isotropic chemical shift / ppm")

# isotropic shift v.s. quadrupolar asymmetry
ax[1].contourf(eta_r, iso_r, pdf.sum(axis=1))
ax[1].set_xlabel(r"quadrupolar asymmetry, $\eta$")
ax[1].set_ylabel("isotropic chemical shift / ppm")

# quadrupolar coupling constant v.s. quadrupolar asymmetry
ax[2].contourf(eta_r, Cq_r, pdf.sum(axis=0))
ax[2].set_xlabel(r"quadrupolar asymmetry, $\eta$")
ax[2].set_ylabel("Cq / MHz")
plt.tight_layout()
plt.show()
```



Simulation setup

Generate spin systems from the above probability distribution.

```
spin_systems = single_site_system_generator(
    isotopes="87Rb",
    isotropic_chemical_shifts=iso,
    quadrupolar={"Cq": Cq * 1e6, "eta": eta},  # Cq in Hz
    abundance=pdf,
)
len(spin_systems)
```

Out:

```
518
```

Simulate a ^{27}Al 3Q-MAS spectrum by using the *ThreeQ_MAS* method.

```
method = ThreeQ_VAS(
    channels=["87Rb"],
    magnetic_flux_density=9.4,  # in T
```

(continues on next page)

(continued from previous page)

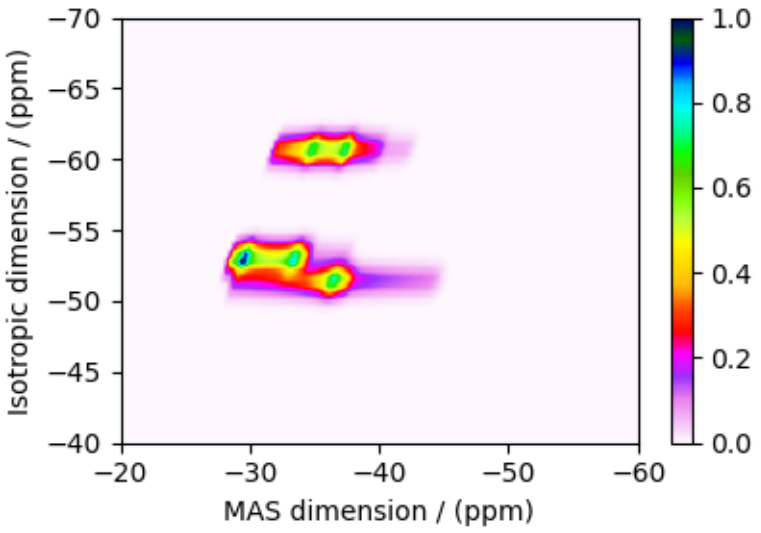
```
    rotor_angle=54.735 * np.pi / 180,
    spectral_dimensions=[
        {
            "count": 96,
            "spectral_width": 7e3,  # in Hz
            "reference_offset": -7e3,  # in Hz
            "label": "Isotropic dimension",
        },
        {
            "count": 256,
            "spectral_width": 1e4,  # in Hz
            "reference_offset": -4e3,  # in Hz
            "label": "MAS dimension",
        },
    ],
)
```

Create the simulator object, add the spin systems and method, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [method]  # add the method
sim.config.number_of_sidebands = 1
sim.run()

data = sim.methods[0].simulation
```

The plot of the corresponding spectrum.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), cmap="gist_ncar_r", aspect="auto")
ax.set_ylim(-40, -70)
ax.set_xlim(-20, -60)
plt.colorbar(cb)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 2.848 seconds)

11.4.2 Czjzek distribution, ^{27}Al (I=5/2) 3QMAS

^{27}Al (I=5/2) 3QMAS simulation of amorphous material.

In this section, we illustrate the simulation of a quadrupolar MQMAS spectrum arising from a distribution of the electric field gradient (EFG) tensors from amorphous material. We proceed by employing the Czjzek distribution model.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

from mrsimulator import Simulator
from mrsimulator.methods import ThreeQ_VAS
from mrsimulator.models import CzjzekDistribution
from mrsimulator.utils.collection import single_site_system_generator
```

Generate probability distribution

```
# The range of isotropic chemical shifts, the quadrupolar coupling constant, and
# asymmetry parameters used in generating a 3D grid.
iso_r = np.arange(101) / 1.5 + 30  # in ppm
Cq_r = np.arange(100) / 4  # in MHz
eta_r = np.arange(10) / 9

# The 3D mesh grid over which the distribution amplitudes are evaluated.
iso, Cq, eta = np.meshgrid(iso_r, Cq_r, eta_r, indexing="ij")

# The 2D amplitude grid of Cq and eta is sampled from the Czjzek distribution model.
Cq_dist, e_dist, amp = CzjzekDistribution(sigma=1).pdf(pos=[Cq_r, eta_r])
```

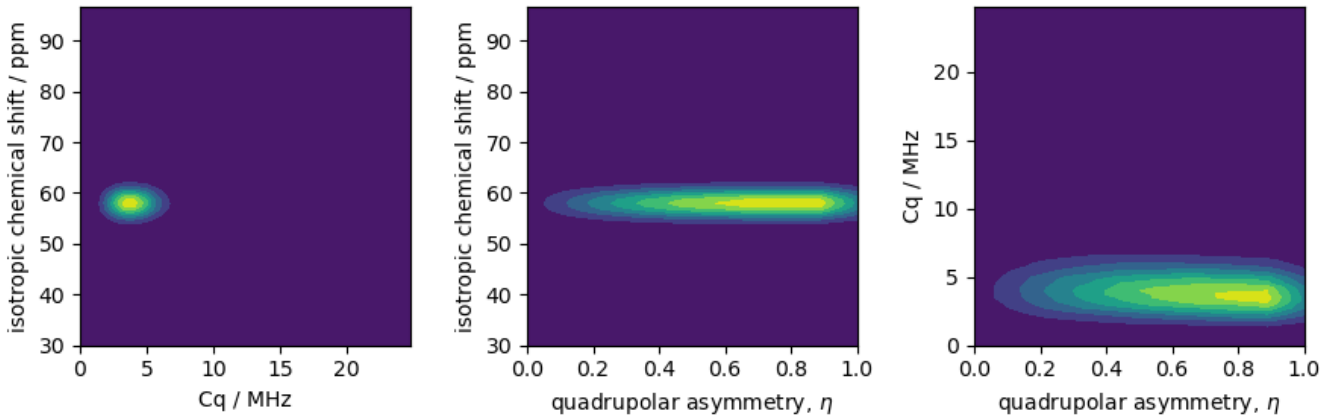
(continues on next page)

(continued from previous page)

```
# The 1D amplitude grid of isotropic chemical shifts is sampled from a Gaussian model.
iso_amp = multivariate_normal(mean=58, cov=[4]).pdf(iso_r)

# The 3D amplitude grid is generated as an uncorrelated distribution of the above two
# distribution, which is the product of the two distributions.
pdf = np.repeat(amp, iso_r.size).reshape(eta_r.size, Cq_r.size, iso_r.size)
pdf *= iso_amp
pdf = pdf.T
```

The two-dimensional projections from this three-dimensional distribution are shown below.

```
_, ax = plt.subplots(1, 3, figsize=(9, 3))

# isotropic shift v.s. quadrupolar coupling constant
ax[0].contourf(Cq_r, iso_r, pdf.sum(axis=2))
ax[0].set_xlabel("Cq / MHz")
ax[0].set_ylabel("isotropic chemical shift / ppm")

# isotropic shift v.s. quadrupolar asymmetry
ax[1].contourf(eta_r, iso_r, pdf.sum(axis=1))
ax[1].set_xlabel(r"quadrupolar asymmetry, $\eta$")
ax[1].set_ylabel("isotropic chemical shift / ppm")

# quadrupolar coupling constant v.s. quadrupolar asymmetry
ax[2].contourf(eta_r, Cq_r, pdf.sum(axis=0))
ax[2].set_xlabel(r"quadrupolar asymmetry, $\eta$")
ax[2].set_ylabel("Cq / MHz")

plt.tight_layout()
plt.show()
```



Simulation setup

Let's create the site and spin system objects from these parameters. Use the *single_site_system_generator()* (page 339) utility function to generate single-site spin systems.

```
spin_systems = single_site_system_generator(
    isotopes="27Al",
    isotropic_chemical_shifts=iso,
    quadrupolar={"Cq": Cq * 1e6, "eta": eta},  # Cq in Hz
    abundance=pdf,
)
len(spin_systems)
```

Out:

```
5787
```

Simulate a ^{27}Al 3Q-MAS spectrum by using the *ThreeQ_MAS* method.

```
mqvas = ThreeQ_VAS(
    channels=["27Al"],
    spectral_dimensions=[
        {
            "count": 512,
            "spectral_width": 26718.475776,  # in Hz
            "reference_offset": -4174.76184,  # in Hz
            "label": "Isotropic dimension",
        },
        {
            "count": 512,
            "spectral_width": 2e4,  # in Hz
            "reference_offset": 2e3,  # in Hz
            "label": "MAS dimension",
        },
    ],
)
```

Create the simulator object, add the spin systems and method, and run the simulation.

```
sim = Simulator()
sim.spin_systems = spin_systems  # add the spin systems
sim.methods = [mqvas]  # add the method
sim.config.number_of_sidebands = 1
sim.run()

data = sim.methods[0].simulation
```

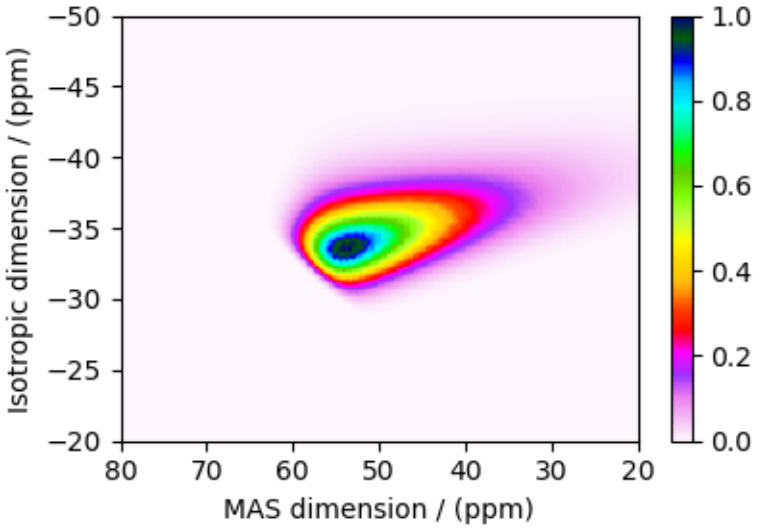
The plot of the corresponding spectrum.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
cb = ax.imshow(data / data.max(), cmap="gist_ncar_r", aspect="auto")
plt.colorbar(cb)
ax.set_ylim(-20, -50)
```

(continues on next page)

(continued from previous page)

```
ax.set_xlim(80, 20)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 12.272 seconds)

CHAPTER

TWELVE

FITTING EXAMPLES (LEAST SQUARES)

The `mrsimulator` library is easily integrable with other python-based libraries. In the following examples, we illustrate the use of LMFIT non-linear least-squares minimization python package to fit a simulation object to experimental data.

12.1 1D Data Fitting

12.1.1 ^{29}Si 1D MAS spinning sideband (CSA)

After acquiring an NMR spectrum, we often require a least-squares analysis to determine site populations and nuclear spin interaction parameters. Generally, this comprises of two steps:

- create a fitting model, and
- determine the model parameters that give the best fit to the spectrum.

Here, we will use the mrsimulator objects to create a fitting model, and use the LMFIT library for performing the least-squares fitting optimization. In this example, we use a synthetic ^{29}Si NMR spectrum of cuspidine, generated from the tensor parameters reported by Hansen *et al.*[1], to demonstrate a simple fitting procedure.

We will begin by importing relevant modules and establishing figure size.

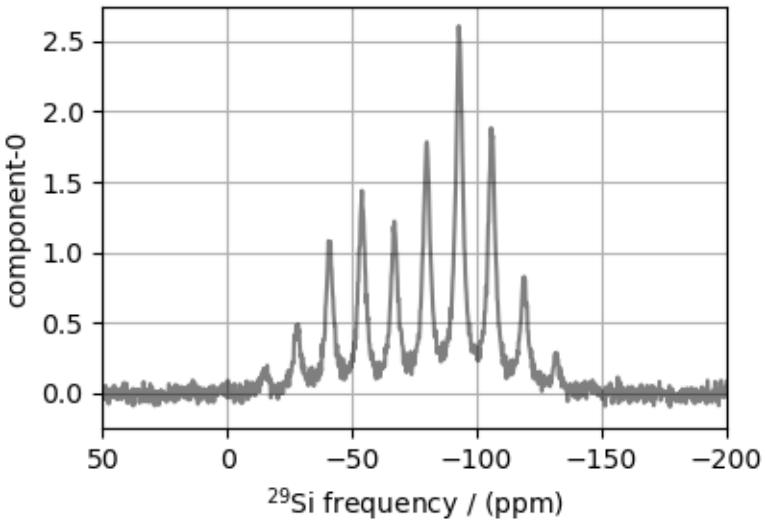
```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, Parameters, fit_report

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
```

[1] Hansen, M. R., Jakobsen, H. J., Skibsted, J., ^{29}Si Chemical Shift Anisotropies in Calcium Silicates from High-Field ^{29}Si MAS NMR Spectroscopy, Inorg. Chem. 2003, **42**, 7, 2368-2377. DOI: 10.1021/ic020647f

Import the dataset

Use the csdmpy module to load the synthetic dataset as a CSDM object.

```
file_ = "https://sandbox.zenodo.org/record/835664/files/synthetic_cuspidine_test.csdf?"
synthetic_experiment = cp.load(file_).real

# standard deviation of noise from the dataset
sigma = 0.03383338

# convert the dimension coordinates from Hz to ppm
synthetic_experiment.x[0].to("ppm", "nmr_frequency_ratio")

# Plot of the synthetic dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(synthetic_experiment, "k", alpha=0.5)
ax.set_xlim(50, -200)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Before you can fit a simulation to an experiment, in this case, the synthetic dataset, you will first need to create a fitting model. We will use the `mrsimulator` objects as tools in creating a model for the least-squares fitting.

Step 1: Create initial guess sites and spin systems.

The initial guess is often based on some prior knowledge about the system under investigation. For the current example, we know that Cuspidine is a crystalline silica polymorph with one crystallographic Si site. Therefore, our initial guess model is a single ^{29}Si site spin system. For non-linear fitting algorithms, as a general recommendation, the initial guess model parameters should be a good starting point for the algorithms to converge.

```
# the guess model comprising of a single site spin system
site = Site(
    isotope="29Si",
    isotropic_chemical_shift=-82.0,  # in ppm,
    shielding_symmetric={"zeta": -63, "eta": 0.4},  # zeta in ppm
)

spin_system = SpinSystem(
    name="Si Site",
    description="A 29Si site in cuspidine",
    sites=[site],  # from the above code
    abundance=100,
)
```

Step 2: Create the method object.

The method should be the same as the one used in the measurement. In this example, we use the *BlochDecaySpectrum* method. Note, when creating the method object, the value of the method parameters must match the respective values used in the experiment.

```
MAS = BlochDecaySpectrum(
    channels=["29Si"],
    magnetic_flux_density=7.1,  # in T
    rotor_frequency=780,  # in Hz
    spectral_dimensions=[
        {
            "count": 2048,
            "spectral_width": 25000,  # in Hz
            "reference_offset": -5000,  # in Hz
        }
    ],
    experiment=synthetic_experiment,  # add the measurement to the method.
)
```

Step 3: Create the Simulator object, add the method and spin system objects, and run the simulation.

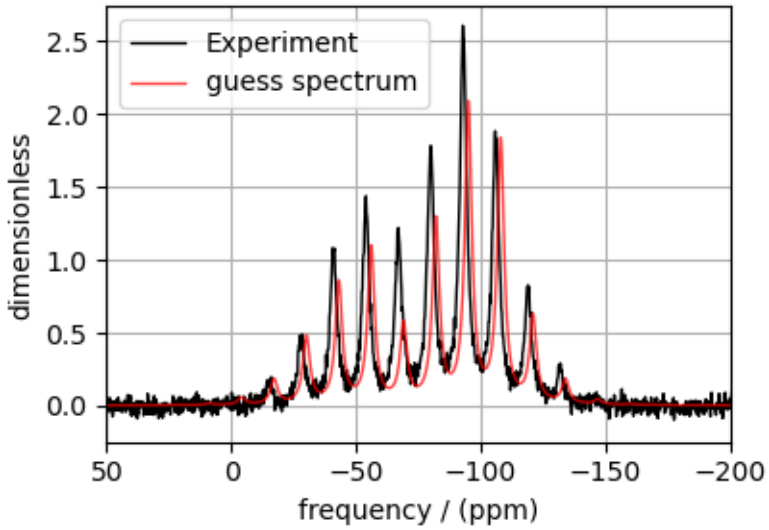
```
sim = Simulator(spin_systems=[spin_system], methods=[MAS])
sim.run()
```

Step 4: Create a SignalProcessor class and apply post simulation processing.

```
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),  # inverse FFT to convert frequency based spectrum to time domain.
        sp.apodization.Exponential(FWHM="200 Hz"),  # apodization of time domain signal.
        sp.FFT(),  # forward FFT to convert time domain signal to frequency spectrum.
        sp.Scale(factor=3),  # scale the frequency spectrum.
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real
```

Step 5: The plot the spectrum. We also plot the synthetic dataset for comparison.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(synthetic_experiment, "k", linewidth=1, label="Experiment")
ax.plot(processed_data, "r", alpha=0.75, linewidth=1, label="guess spectrum")
ax.set_xlim(50, -200)
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



Setup a Least-squares minimization

Now that our model is ready, the next step is to set up a least-squares minimization. You may use any optimization package of choice, here we show an application using LMFIT. You may read more on the LMFIT documentation page.

Create fitting parameters

Next, you will need a list of parameters that will be used in the fit. The *LMFIT* library provides a Parameters class to create a list of parameters.

```
site1 = spin_system.sites[0]
params = Parameters()

params.add(name="iso", value=site1.isotropic_chemical_shift)
params.add(name="eta", value=site1.shielding_symmetric.eta, min=0, max=1)
params.add(name="zeta", value=site1.shielding_symmetric.zeta)
params.add(name="FWHM", value=processor.operations[1].FWHM)
params.add(name="factor", value=processor.operations[3].factor)
```

Create a minimization function

Note, the above set of parameters does not know about the model. You will need to set up a function that will

- update the parameters of the *Simulator* and *SignalProcessor* object based on the LMFIT parameter updates,
- re-simulate the spectrum based on the updated values, and
- return the difference between the experiment and simulation.

```
def minimization_function(params, sim, processor, sigma=1):
    values = params.valuesdict()

    # the experiment data as a Numpy array
    intensity = sim.methods[0].experiment.y[0].components[0].real

    # Here, we update simulation parameters iso, eta, and zeta for the site object
    site = sim.spin_systems[0].sites[0]
    site.isotropic_chemical_shift = values["iso"]
    site.shielding_symmetric.eta = values["eta"]
    site.shielding_symmetric.zeta = values["zeta"]

    # run the simulation
    sim.run()

    # update the SignalProcessor parameter and apply line broadening.
    # update the scaling factor parameter at index 3 of operations list.
    processor.operations[3].factor = values["factor"]
    # update the exponential apodization FWHM parameter at index 1 of operations list.
    processor.operations[1].FWHM = values["FWHM"]

    # apply signal processing
    processed_data = processor.apply_operations(sim.methods[0].simulation)

    # return the difference vector.
    diff = intensity - processed_data.y[0].components[0].real
    return diff / sigma
```

Note: To automate the fitting process, we provide a function to parse the `Simulator` and `SignalProcessor` objects for parameters and construct an *LMFIT* `Parameters` object. Similarly, a minimization function, analogous to the above *minimization_function*, is also included in the *mrsimulator* library. See the next example for usage instructions.

Perform the least-squares minimization

With the synthetic dataset, simulation, and the initial guess parameters, we are ready to perform the fit. To fit, we use the *LMFIT* Minimizer class.

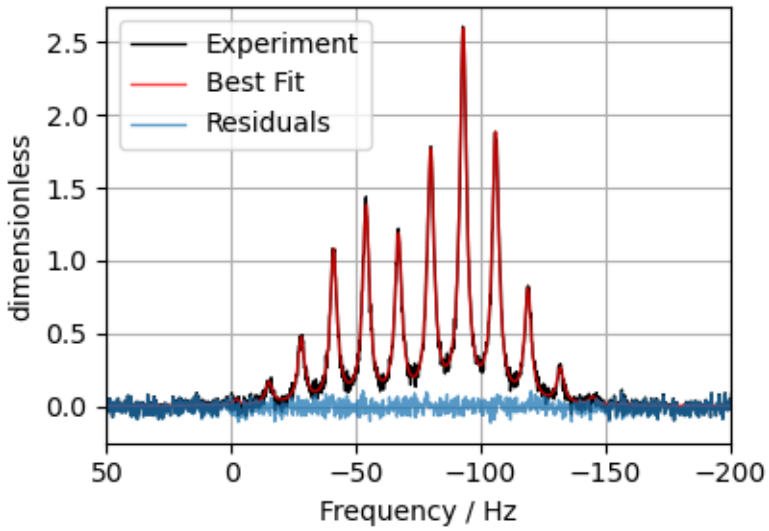
```
minner = Minimizer(minimization_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
print(fit_report(result))
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 38
    # data points      = 2048
    # variables        = 5
    chi-square         = 2386.63679
    reduced chi-square = 1.16820205
    Akaike info crit   = 323.387966
    Bayesian info crit = 351.511061
[[Variables]]
    iso:    -79.9388706 +/- 0.00572867 (0.01%) (init = -82)
    eta:     0.60780315 +/- 0.00350688 (0.58%) (init = 0.4)
    zeta:   -58.2017026 +/- 0.11096611 (0.19%) (init = -63)
    FWHM:    196.532147 +/- 0.84737888 (0.43%) (init = 200)
    factor:  3.69478971 +/- 0.01102742 (0.30%) (init = 3)
[[Correlations]] (unreported correlations are < 0.100)
    C(FWHM, factor) =  0.525
    C(eta, zeta)    =  0.333
    C(zeta, factor) = -0.228
    C(eta, factor)  =  0.158
```

The plot of the fit, measurement and the residuals is shown below.

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(synthetic_experiment, "k", linewidth=1, label="Experiment")
ax.plot(best_fit, "r", alpha=0.75, linewidth=1, label="Best Fit")
ax.plot(residuals, alpha=0.75, linewidth=1, label="Residuals")
ax.set_xlabel("Frequency / Hz")
ax.set_xlim(50, -200)
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 2.335 seconds)

12.1.2 ^{31}P MAS NMR of crystalline Na2PO4 (CSA)

In this example, we illustrate the use of the mrsimulator objects to

- create a CSA fitting model using Simulator and SignalProcessor objects,
- use the fitting model to perform a least-squares analysis, and
- extract the fitting parameters from the model.

We use the LMFIT library to fit the spectrum. The following example shows the least-squares fitting procedure applied to the ^{31}P MAS NMR spectrum of Na_2PO_4. The following experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

Start by importing the relevant modules.

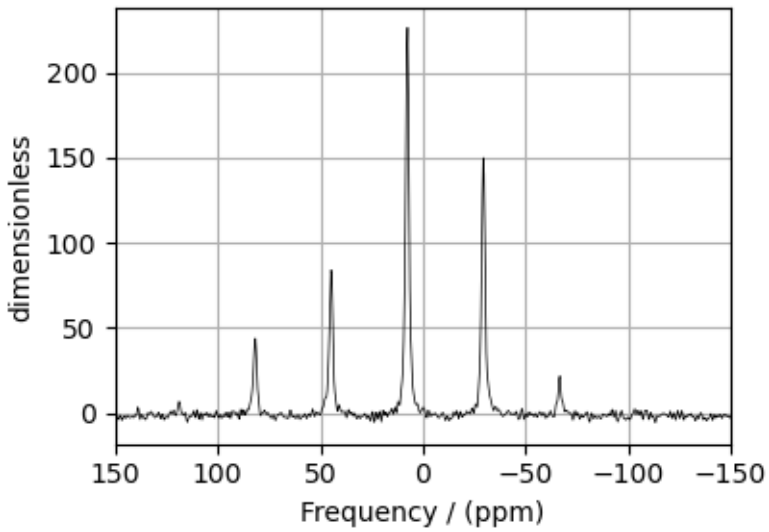
```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984

Import the dataset

Import the experimental data. We use dataset file serialized with the CSDM file-format, using the csdmpy module.

```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "31P Phosphate 6kHz.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 1.523217

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the dimension coordinates from Hz to ppm.
experiment.x[0].to("ppm", "nmr_frequency_ratio")

# plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(150, -150)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

A fitting model is a composite of `Simulator` and `SignalProcessor` objects.

Step 1: Create initial guess sites and spin systems

```
P_31 = Site(
    isotope="31P",
    isotropic_chemical_shift=5.0,  # in ppm,
    shielding_symmetric={"zeta": -80, "eta": 0.5},  # zeta in Hz
)

spin_systems = [SpinSystem(sites=[P_31])]
```

Step 2: Create the method object. Create an appropriate method object that closely resembles the technique used in acquiring the experimental data. The attribute values of this method must meet the experimental conditions, including the acquisition channels, the magnetic flux density, rotor angle, rotor frequency, and the spectral/spectroscopic dimension.

In the following example, we set up a Bloch decay spectrum method where the spectral/spectroscopic dimension information, i.e., count, spectral_width, and the reference_offset, is extracted from the CSDM dimension metadata using the *get_spectral_dimensions()* (page 340) utility function. The remaining attribute values are set to the experimental conditions.

```
# get the count, spectral_width, and reference_offset information from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS = BlochDecaySpectrum(
    channels=["31P"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=6000,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # experimental dataset
)

# A method object queries every spin system for a list of transition pathways that are
# relevant to the given method. Since the method and the number of spin systems remains
# unchanged during the least-squares analysis, a one-time query is sufficient. To avoid
# querying for the transition pathways at every iteration in a least-squares fitting,
# evaluate the transition pathways once and store it as follows
for sys in spin_systems:
    sys.transition_pathways = MAS.get_transition_pathways(sys)
```

Step 3: Create the Simulator object and add the method and spin system objects.

```
sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.run()
```

Step 4: Create a SignalProcessor class object and apply the post-simulation signal processing operations.

```
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="0.3 kHz"),
        sp.FFT(),
```

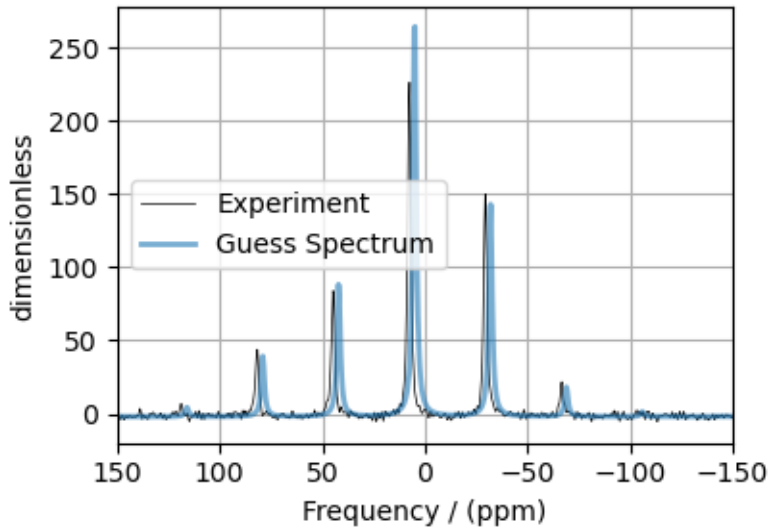
(continues on next page)

(continued from previous page)

```
        sp.Scale(factor=300),
        sp.baseline.ConstantOffset(offset=-2),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real
```

Step 5: The plot of the data and the guess spectrum.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6, label="Guess Spectrum")
ax.set_xlim(150, -150)
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Once you have a fitting model, you need to create the list of parameters to use in the least-squares minimization. For this, you may use the Parameters class from *LMFIT*, as described in the previous example. Here, we make use of a utility function, *make_LMFIT_params()* (page 351), to simplifies the LMFIT parameters generation process. By default, the function only creates parameters from the SpinSystem and SignalProcessor objects. Often, in spectrum with sidebands, spinning speed may not be accurately known; and is, therefore, included as a fitting parameter. To include a keyword from the method object, use the *include* argument of the function, as follows,

Step 6: Create a list of parameters.

```
params = sf.make_LMFIT_params(sim, processor, include={"rotor_frequency"})
```

The *make_ LMFIT_ params* parses the instances of the `Simulator` and the `PostSimulator` objects for parameters and returns a LMFIT *Parameters* object.

Customize the Parameters: You may customize the parameters list, `params`, as desired. Here, we remove the abundance parameter.

```
params.pop("sys_0_abundance")
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM         0.3     -inf      inf     True     None
SP_0_operation_3_Scale_factor             300     -inf      inf     True     None
SP_0_operation_4_ConstantOffset_offset     -2     -inf      inf     True     None
mth_0_rotor_frequency                    6000     5900     6100     True     None
sys_0_site_0_isotropic_chemical_shift       5     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_eta      0.5        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta     -80     -inf      inf     True     None
None
```

Step 7: Perform the least-squares minimization. For the user's convenience, we also provide a utility function, *LMFIT_min_function()* (page 351), for evaluating the difference vector between the simulation and experiment, based on the parameters update. You may use this function directly as the argument of the LMFIT Minimizer class, as follows,

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 99
    # data points      = 2048
    # variables        = 7
    chi-square         = 2364.72004
    reduced chi-square = 1.15860855
    Akaike info crit   = 308.494070
    Bayesian info crit = 347.866403
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:   7.57609723 +/- 0.00347983 (0.05%) (init = 5)
    sys_0_site_0_shielding_symmetric_zeta:  -88.1881083 +/- 0.20973038 (0.24%) (init = -80)
    sys_0_site_0_shielding_symmetric_eta:    0.45591108 +/- 0.01141676 (2.50%) (init = 0.5)
    mth_0_rotor_frequency:                   6012.45935 +/- 0.80711552 (0.01%) (init = 6000)
    SP_0_operation_1_Exponential_FWHM:       0.27280282 +/- 0.00163466 (0.60%) (init = 0.3)
    SP_0_operation_3_Scale_factor:           268.854046 +/- 1.20048486 (0.45%) (init = 300)
    SP_0_operation_4_ConstantOffset_offset: -1.65291840 +/- 0.03734295 (2.26%) (init = -2)
[[Correlations]] (unreported correlations are < 0.100)
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_3_Scale_factor)            =  0.664
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta) =  0.362
    C(SP_0_operation_3_Scale_factor, SP_0_operation_4_ConstantOffset_offset)       = -0.242
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_3_Scale_factor)         =  0.227
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_3_Scale_factor)        = -0.225
    C(sys_0_site_0_isotropic_chemical_shift, mth_0_rotor_frequency)                =  0.203
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_4_ConstantOffset_offset)   = -0.161
```

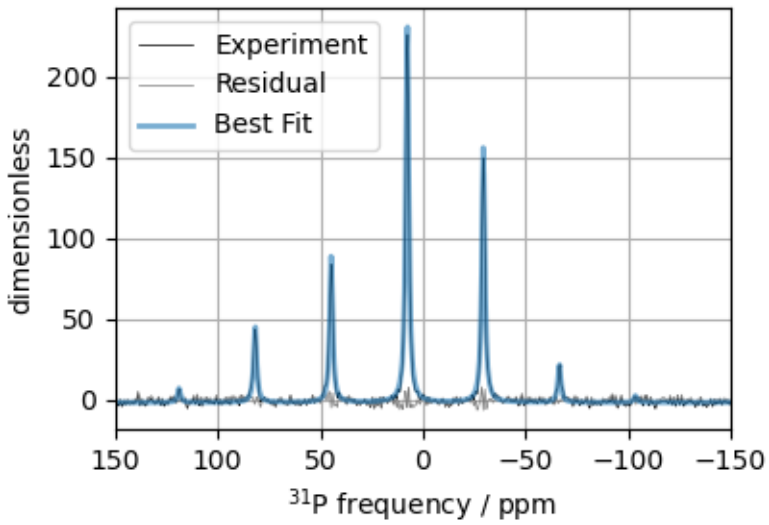
(continues on next page)

(continued from previous page)

```
    C(sys_0_site_0_shielding_symmetric_zeta, mth_0_rotor_frequency)                = -0.108
```

Step 8: The plot of the fit, measurement, and residuals.

```
# Best fit spectrum
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6, label="Best Fit")
ax.set_xlabel(r"$^{31}$P frequency / ppm")
ax.set_xlim(150, -150)
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 2.755 seconds)

12.1.3 ^{31}P static NMR of crystalline Na2PO4 (CSA)

The following is a CSA static least-squares fitting example of a ^{31}P MAS NMR spectrum of Na_2PO_4. The following experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit
```

(continues on next page)

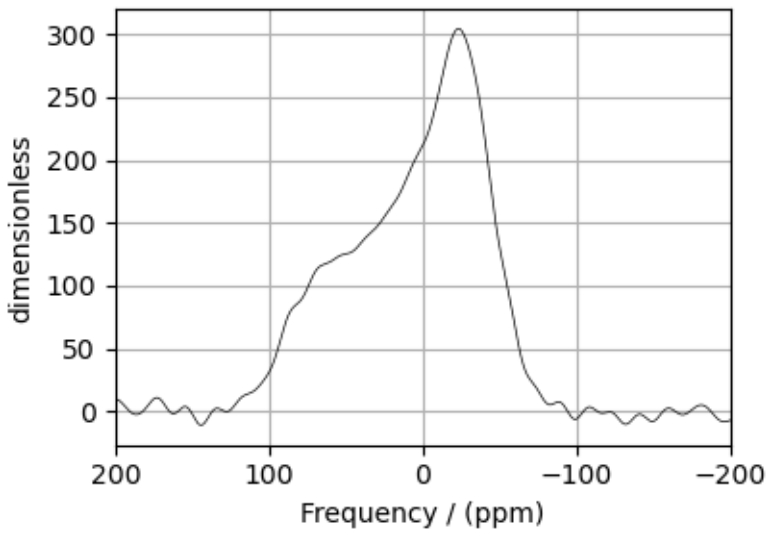
[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984

(continued from previous page)

```
from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "31P Phophonate Static.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 3.258224

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(200, -200)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Spin System

```
P_31 = Site(
    isotope="31P",
    isotropic_chemical_shift=5.0,  # in ppm,
    shielding_symmetric={"zeta": -80, "eta": 0.5},  # zeta in Hz
)

spin_systems = [SpinSystem(sites=[P_31])]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

static1D = BlochDecaySpectrum(
    channels=["31P"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=0,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # experimental dataset
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = static1D.get_transition_pathways(sys)
```

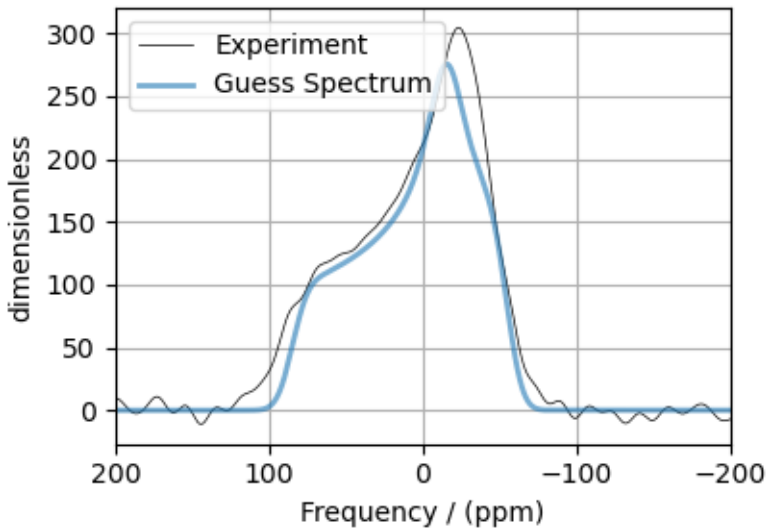
Guess Model Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[static1D])
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Gaussian(FWHM="3000 Hz"),
        sp.FFT(),
        sp.Scale(factor=4000),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
```

(continues on next page)

(continued from previous page)

```
ax.plot(processed_data, linewidth=2, alpha=0.6, label="Guess Spectrum")
ax.set_xlim(200, -200)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
params.pop("sys_0_abundance")
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM           3000     -inf      inf     True     None
SP_0_operation_3_Scale_factor            4000     -inf      inf     True     None
sys_0_site_0_isotropic_chemical_shift       5     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_eta      0.5        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta      -80     -inf      inf     True     None
None
```

Solve the minimizer using LMFIT

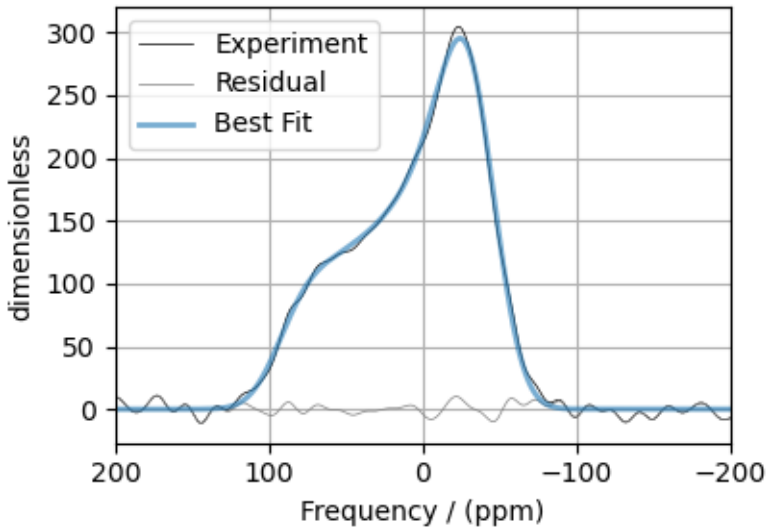
```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 69
    # data points      = 2048
    # variables        = 5
    chi-square         = 3405.48278
    reduced chi-square = 1.66690298
    Akaike info crit   = 1051.45512
    Bayesian info crit = 1079.57821
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  6.32228235 +/- 0.07291393 (1.15%) (init = 5)
    sys_0_site_0_shielding_symmetric_zeta: -86.8324323 +/- 0.14006961 (0.16%) (init = -80)
    sys_0_site_0_shielding_symmetric_eta:   0.18101163 +/- 0.02313105 (12.78%) (init = 0.5)
    SP_0_operation_1_Gaussian_FWHM:         5978.91422 +/- 101.488619 (1.70%) (init = 3000)
    SP_0_operation_3_Scale_factor:          4626.51660 +/- 6.23103807 (0.13%) (init = 4000)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_1_Gaussian_FWHM)         = -0.975
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_shielding_symmetric_zeta) = -0.590
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_1_Gaussian_FWHM)        =  0.325
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_3_Scale_factor)         = -0.308
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta)  = -0.296
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_3_Scale_factor)         =  0.279
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_shielding_symmetric_eta)  = -0.249
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_1_Gaussian_FWHM)        =  0.216
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_3_Scale_factor)                =  0.182
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_3_Scale_factor)          = -0.118
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6, label="Best Fit")
ax.set_xlim(200, -200)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 1.372 seconds)

12.1.4 ^{13}C MAS NMR of Glycine (CSA) [1940 Hz]

The following is a sideband least-squares fitting example of a ^{13}C MAS NMR spectrum of Glycine spinning at 1940 Hz. The following experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

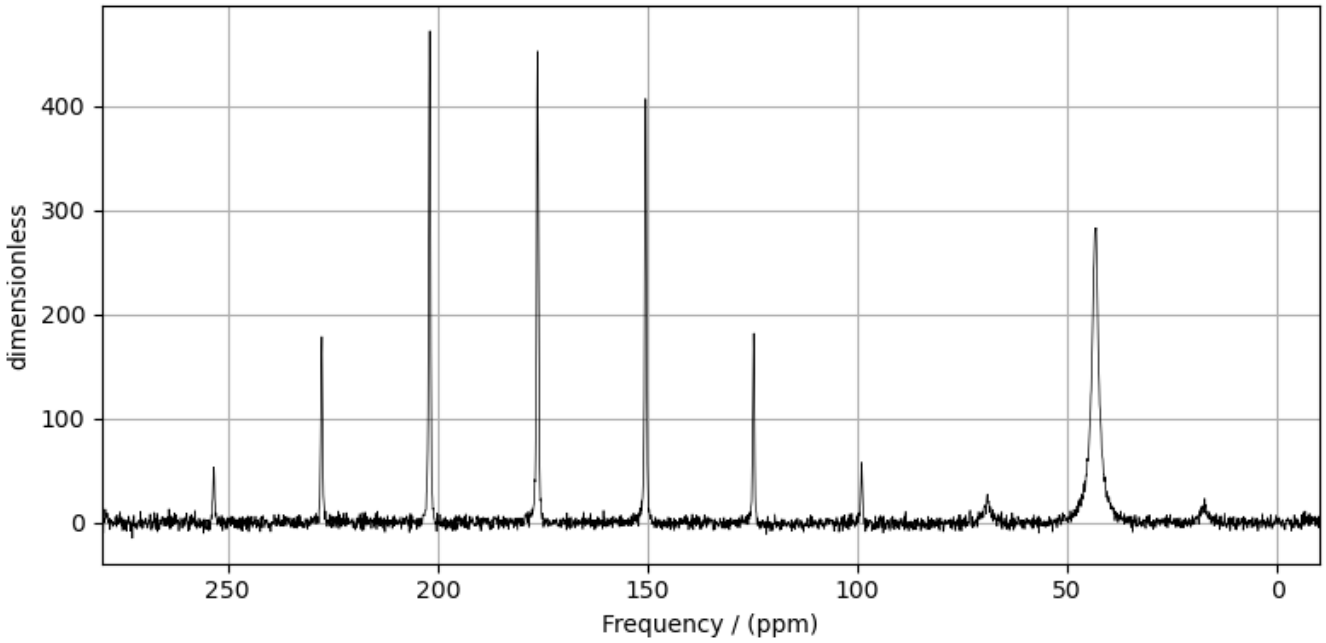
```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "13C MAS 1940Hz - Glycine.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 3.822249

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real
```

(continues on next page)

[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984

(continued from previous page)

```
# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(280, -10)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Spin System

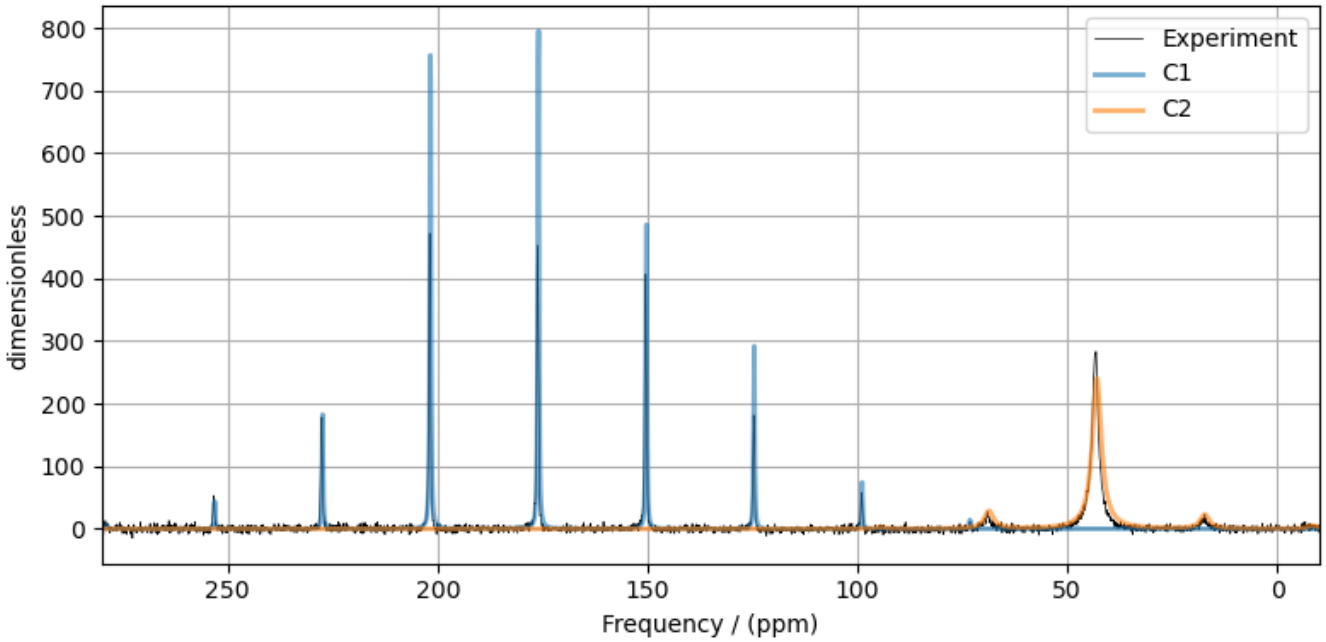
```
C1 = Site(
    isotope="13C",
    isotropic_chemical_shift=176.0,  # in ppm
    shielding_symmetric={"zeta": 70, "eta": 0.6},  # zeta in Hz
)
C2 = Site(
    isotope="13C",
    isotropic_chemical_shift=43.0,  # in ppm
    shielding_symmetric={"zeta": 30, "eta": 0.5},  # zeta in Hz
)

spin_systems = [SpinSystem(sites=[C1], name="C1"), SpinSystem(sites=[C2], name="C2")]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=7.05,  # in T
    rotor_frequency=1940,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # experimental dataset
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS.get_transition_pathways(sys)
```

Guess Model Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.config.decompose_spectrum = "spin_system"
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="20 Hz", dv_index=0),  # spin system 0
        sp.apodization.Exponential(FWHM="200 Hz", dv_index=1),  # spin system 1
        sp.FFT(),
        sp.Scale(factor=100),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6)
ax.set_xlim(280, -10)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor, include={"rotor_frequency"})
params["sys_1_site_0_shielding_symmetric_eta"].vary = False
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM          20     -inf      inf     True     None
SP_0_operation_2_Exponential_FWHM         200     -inf      inf     True     None
SP_0_operation_4_Scale_factor             100     -inf      inf     True     None
mth_0_rotor_frequency                    1940     1840     2040     True     None
sys_0_abundance                            50        0      100     True     None
sys_0_site_0_isotropic_chemical_shift     176     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_eta      0.6        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta      70     -inf      inf     True     None
sys_1_abundance                            50        0      100    False 100-sys_0_abundance
sys_1_site_0_isotropic_chemical_shift      43     -inf      inf     True     None
sys_1_site_0_shielding_symmetric_eta      0.5        0        1    False     None
sys_1_site_0_shielding_symmetric_zeta      30     -inf      inf     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 78
    # data points      = 4096
    # variables        = 10
    chi-square         = 5247.81571
    reduced chi-square = 1.28434060
    Akaike info crit   = 1034.99309
    Bayesian info crit = 1098.17075
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  176.133538 +/- 7.7530e-04 (0.00%) (init = 176)
    sys_0_site_0_shielding_symmetric_zeta:  72.6573659 +/- 0.18735813 (0.26%) (init = 70)
    sys_0_site_0_shielding_symmetric_eta:   0.92348112 +/- 0.00507035 (0.55%) (init = 0.6)
    sys_0_abundance:                        57.5381144 +/- 0.21265458 (0.37%) (init = 50)
    sys_1_site_0_isotropic_chemical_shift:  43.3604050 +/- 0.00500134 (0.01%) (init = 43)
    sys_1_site_0_shielding_symmetric_zeta:  23.1190164 +/- 0.50113151 (2.17%) (init = 30)
    sys_1_site_0_shielding_symmetric_eta:   0.5 (fixed)
    sys_1_abundance:                        42.4618856 +/- 0.21265458 (0.50%) == '100-sys_0_
↪abundance'
    mth_0_rotor_frequency:                  1940.88271 +/- 0.05721410 (0.00%) (init = 1940)
    SP_0_operation_1_Exponential_FWHM:      29.6025777 +/- 0.17069359 (0.58%) (init = 20)
    SP_0_operation_2_Exponential_FWHM:      134.355218 +/- 1.06603885 (0.79%) (init = 200)
    SP_0_operation_4_Scale_factor:          170.972529 +/- 0.68520995 (0.40%) (init = 100)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_abundance, sys_1_site_0_shielding_symmetric_zeta)                    = -0.606
    C(sys_1_site_0_shielding_symmetric_zeta, SP_0_operation_4_Scale_factor)      =  0.555
    C(sys_0_abundance, SP_0_operation_2_Exponential_FWHM)                        = -0.446
    C(sys_0_abundance, SP_0_operation_4_Scale_factor)                            = -0.442
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta) = -0.420
    C(SP_0_operation_2_Exponential_FWHM, SP_0_operation_4_Scale_factor)           =  0.409
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_4_Scale_factor)           =  0.395
    C(sys_0_abundance, SP_0_operation_1_Exponential_FWHM)                        =  0.319
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_4_Scale_factor)      =  0.144
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_abundance)                    =  0.122
```

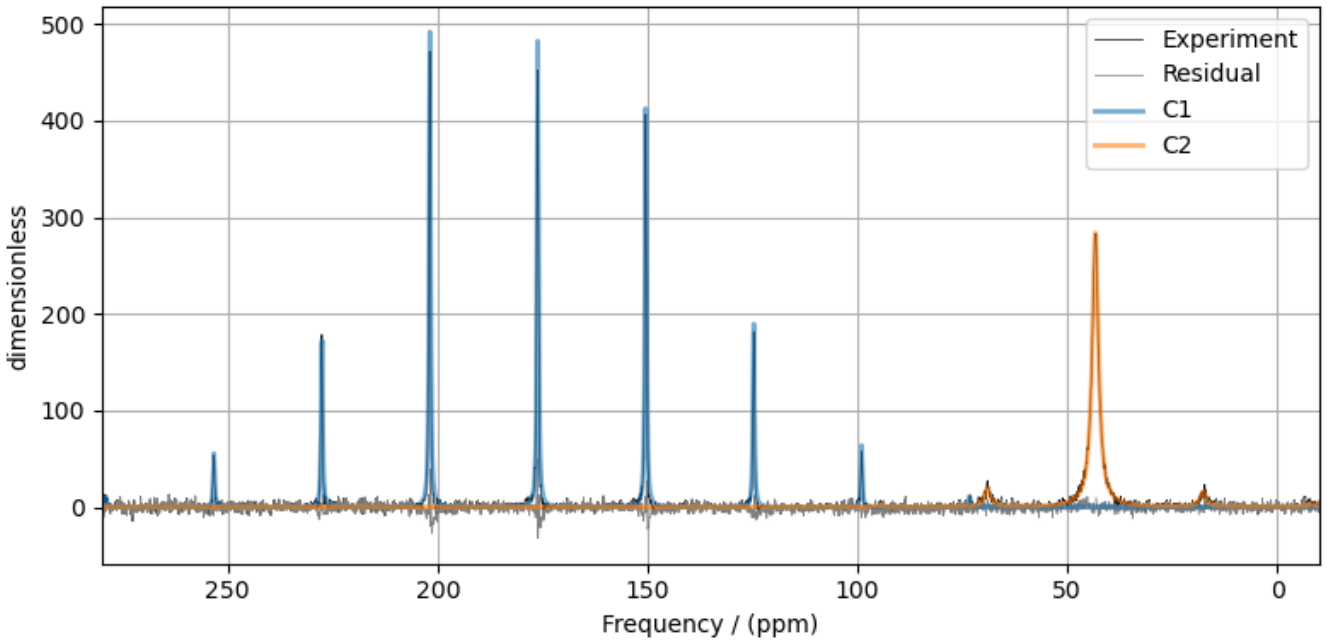
The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6)
ax.set_xlim(280, -10)
plt.grid()
plt.legend()
plt.tight_layout()
```

(continues on next page)

(continued from previous page)

```
plt.show()
```



Total running time of the script: (0 minutes 3.584 seconds)

12.1.5 ^{13}C MAS NMR of Glycine (CSA) [5000 Hz]

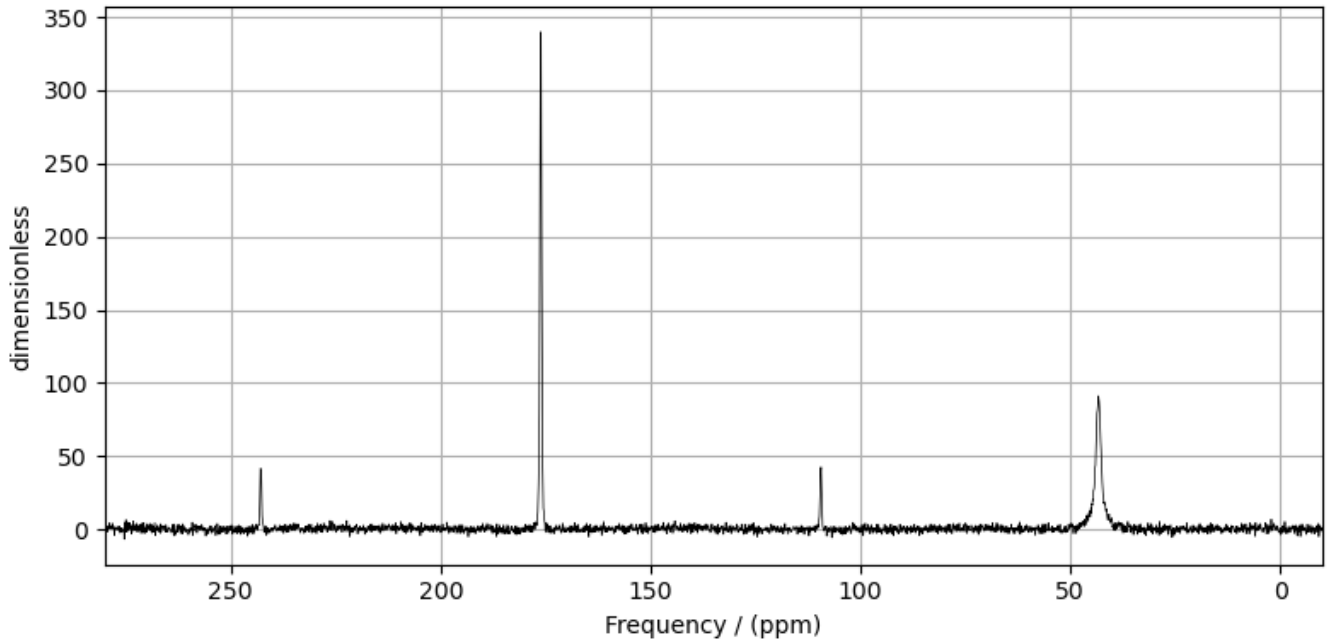
The following is a sideband least-squares fitting example of a ^{13}C MAS NMR spectrum of Glycine spinning at 5000 Hz. The following experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984

Import the dataset

```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "13C MAS 5000Hz - Glycine.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 3.822249

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(280, -10)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Spin System

```
C1 = Site(
    isotope="13C",
    isotropic_chemical_shift=176.0,  # in ppm
    shielding_symmetric={"zeta": 70, "eta": 0.6},  # zeta in Hz
)
C2 = Site(
    isotope="13C",
    isotropic_chemical_shift=43.0,  # in ppm
)

spin_systems = [SpinSystem(sites=[C1], name="C1"), SpinSystem(sites=[C2], name="C2")]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=7.05,  # in T
    rotor_frequency=5000,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # experimental dataset
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS.get_transition_pathways(sys)
```

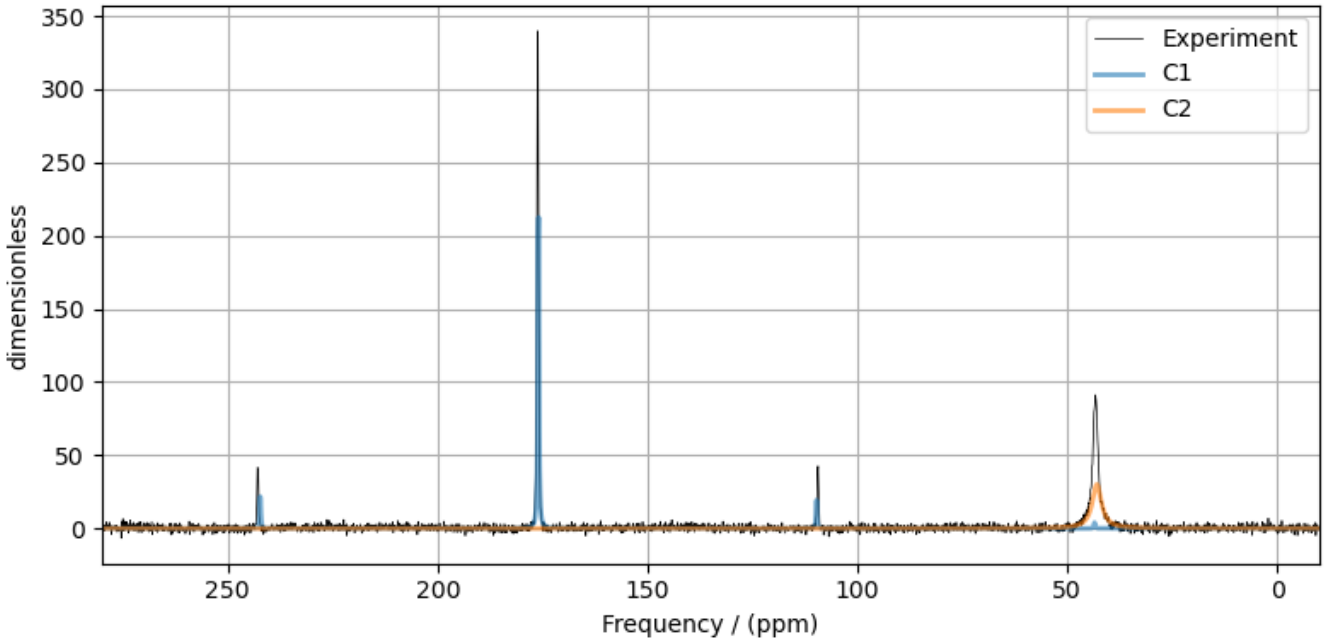
Guess Model Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.config.decompose_spectrum = "spin_system"
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="20 Hz", dv_index=0),  # spin system 0
        sp.apodization.Exponential(FWHM="200 Hz", dv_index=1),  # spin system 1
        sp.FFT(),
        sp.Scale(factor=10),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real
```

(continues on next page)

(continued from previous page)

```
# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6)
ax.set_xlim(280, -10)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor, include={"rotor_frequency"})
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM      20     -inf      inf     True     None
SP_0_operation_2_Exponential_FWHM     200     -inf      inf     True     None
SP_0_operation_4_Scale_factor          10     -inf      inf     True     None
mth_0_rotor_frequency                5000     4900     5100     True     None
sys_0_abundance                        50        0      100     True     None
sys_0_site_0_isotropic_chemical_shift 176     -inf      inf     True     None
```

(continues on next page)

(continued from previous page)

```
sys_0_site_0_shielding_symmetric_eta        0.6        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta        70     -inf      inf     True     None
sys_1_abundance                              50        0      100    False 100-sys_0_abundance
sys_1_site_0_isotropic_chemical_shift        43     -inf      inf     True     None
None
```

Solve the minimizer using LMFIT

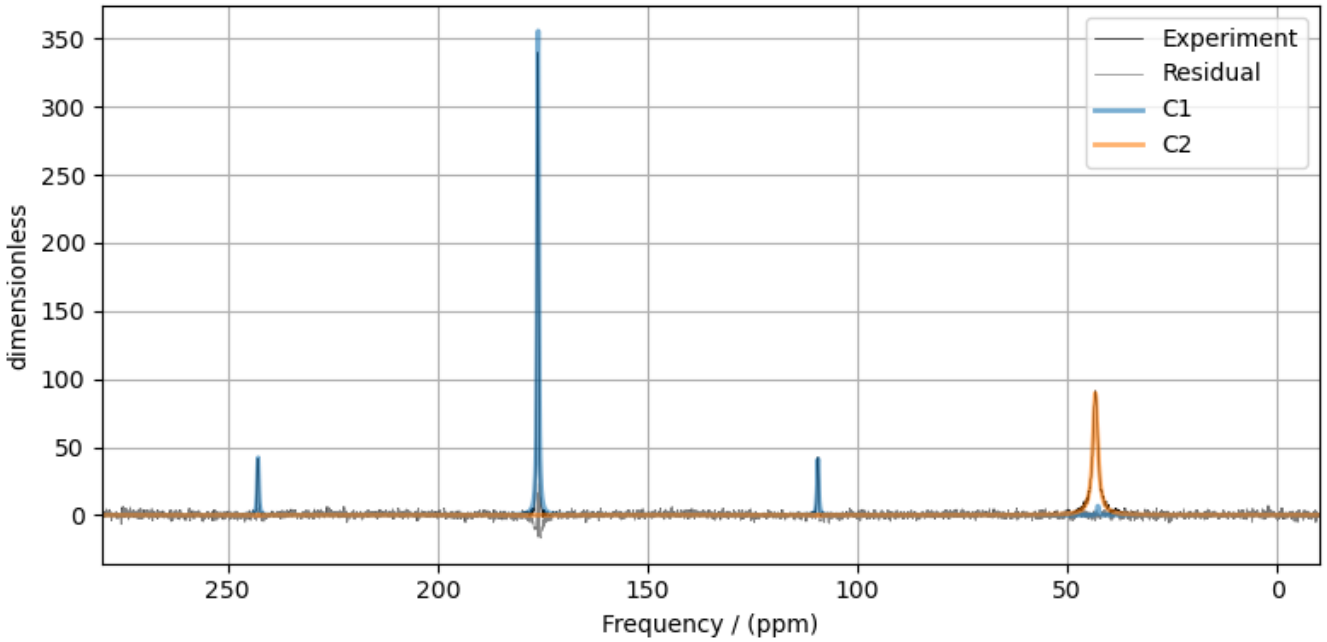
```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 191
    # data points      = 4096
    # variables        = 9
    chi-square         = 1267.95049
    reduced chi-square = 0.31023991
    Akaike info crit   = -4785.00677
    Bayesian info crit = -4728.14688
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  176.093349 +/- 8.2151e-04 (0.00%) (init = 176)
    sys_0_site_0_shielding_symmetric_zeta: -73.6104864 +/- 1.52263789 (2.07%) (init = 70)
    sys_0_site_0_shielding_symmetric_eta:   0.99999893 +/- 0.07549726 (7.55%) (init = 0.6)
    sys_0_abundance:                        58.2535314 +/- 0.29047642 (0.50%) (init = 50)
    sys_1_site_0_isotropic_chemical_shift:  43.3581388 +/- 0.00711644 (0.02%) (init = 43)
    sys_1_abundance:                        41.7464686 +/- 0.29047642 (0.70%) == '100-sys_0_
↪abundance'
    mth_0_rotor_frequency:                  5033.23931 +/- 0.36010260 (0.01%) (init = 5000)
    SP_0_operation_1_Exponential_FWHM:      26.9286152 +/- 0.18909095 (0.70%) (init = 20)
    SP_0_operation_2_Exponential_FWHM:      106.899303 +/- 1.49566542 (1.40%) (init = 200)
    SP_0_operation_4_Scale_factor:          38.3645243 +/- 0.20216190 (0.53%) (init = 10)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta)  =  0.899
    C(sys_0_abundance, SP_0_operation_2_Exponential_FWHM)                           = -0.600
    C(SP_0_operation_2_Exponential_FWHM, SP_0_operation_4_Scale_factor)             =  0.544
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_4_Scale_factor)             =  0.358
    C(sys_0_abundance, SP_0_operation_4_Scale_factor)                               = -0.319
    C(sys_0_abundance, SP_0_operation_1_Exponential_FWHM)                           =  0.281
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_4_Scale_factor)         = -0.266
    C(sys_0_site_0_shielding_symmetric_eta, sys_1_site_0_isotropic_chemical_shift)  =  0.182
    C(sys_0_site_0_shielding_symmetric_eta, sys_0_abundance)                        =  0.150
    C(sys_0_site_0_shielding_symmetric_zeta, sys_1_site_0_isotropic_chemical_shift) =  0.128
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_4_Scale_factor)          = -0.116
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6)
ax.set_xlim(280, -10)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 6.644 seconds)

12.1.6 ^{13}C MAS NMR of Glycine (CSA) [960 Hz]

The following is a sideband least-squares fitting example of a ^{13}C MAS NMR spectrum of Glycine spinning at 960 Hz. The following experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit
```

(continues on next page)

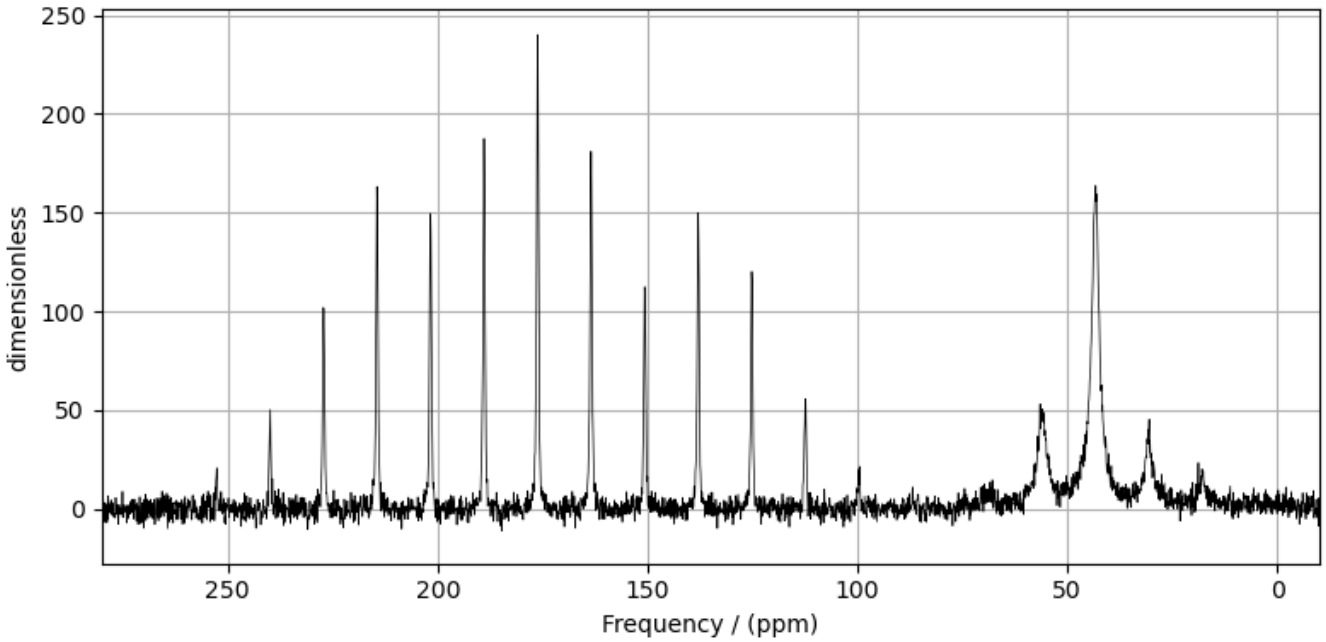
[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984

(continued from previous page)

```
from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "13C MAS 960Hz - Glycine.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 3.822249

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(280, -10)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Spin System

```
C1 = Site(
    isotope="13C",
    isotropic_chemical_shift=176.0,  # in ppm
    shielding_symmetric={"zeta": 70, "eta": 0.6},  # zeta in Hz
)
C2 = Site(
    isotope="13C",
    isotropic_chemical_shift=43.0,  # in ppm
    shielding_symmetric={"zeta": 30, "eta": 0.5},  # zeta in Hz
)

spin_systems = [SpinSystem(sites=[C1], name="C1"), SpinSystem(sites=[C2], name="C2")]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=7.05,  # in T
    rotor_frequency=960,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # experimental dataset
)
```

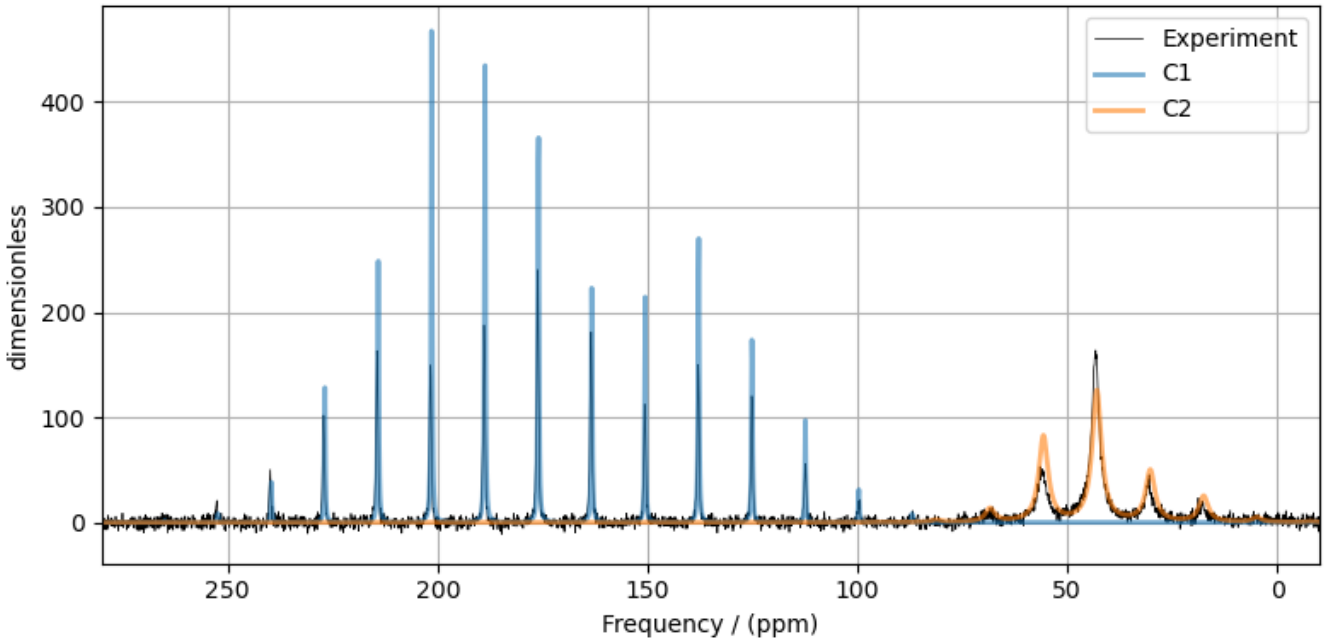
(continues on next page)

(continued from previous page)

```
# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS.get_transition_pathways(sys)
```

Guess Model Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.config.decompose_spectrum = "spin_system"
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="20 Hz", dv_index=0),  # spin system 0
        sp.apodization.Exponential(FWHM="200 Hz", dv_index=1),  # spin system 1
        sp.FFT(),
        sp.Scale(factor=100),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6)
ax.set_xlim(280, -10)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor, include={"rotor_frequency"})
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM          20     -inf      inf     True     None
SP_0_operation_2_Exponential_FWHM         200     -inf      inf     True     None
SP_0_operation_4_Scale_factor             100     -inf      inf     True     None
mth_0_rotor_frequency                     960      860     1060     True     None
sys_0_abundance                            50        0      100     True     None
sys_0_site_0_isotropic_chemical_shift     176     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_eta      0.6        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta      70     -inf      inf     True     None
sys_1_abundance                            50        0      100    False 100-sys_0_abundance
sys_1_site_0_isotropic_chemical_shift      43     -inf      inf     True     None
sys_1_site_0_shielding_symmetric_eta      0.5        0        1     True     None
sys_1_site_0_shielding_symmetric_zeta      30     -inf      inf     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 85
    # data points      = 4096
    # variables        = 11
    chi-square         = 4246.76287
    reduced chi-square = 1.03959923
    Akaike info crit   = 170.054533
    Bayesian info crit = 239.549961
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  176.131413 +/- 0.00127988 (0.00%) (init = 176)
    sys_0_site_0_shielding_symmetric_zeta:  71.2620768 +/- 0.25475162 (0.36%) (init = 70)
    sys_0_site_0_shielding_symmetric_eta:   0.91217925 +/- 0.00555631 (0.61%) (init = 0.6)
    sys_0_abundance:                        54.5029597 +/- 0.24448335 (0.45%) (init = 50)
    sys_1_site_0_isotropic_chemical_shift:  43.3284518 +/- 0.00887254 (0.02%) (init = 43)
    sys_1_site_0_shielding_symmetric_zeta:  23.0234065 +/- 0.18114229 (0.79%) (init = 30)
    sys_1_site_0_shielding_symmetric_eta:   0.50291614 +/- 0.04141365 (8.23%) (init = 0.5)
    sys_1_abundance:                        45.4970403 +/- 0.24448335 (0.54%) == '100-sys_0_
↪abundance'
    mth_0_rotor_frequency:                  962.151219 +/- 0.04197724 (0.00%) (init = 960)
    SP_0_operation_1_Exponential_FWHM:      33.8471657 +/- 0.27755222 (0.82%) (init = 20)
    SP_0_operation_2_Exponential_FWHM:      179.799993 +/- 1.81583654 (1.01%) (init = 200)
    SP_0_operation_4_Scale_factor:          172.469361 +/- 0.82542295 (0.48%) (init = 100)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_1_site_0_shielding_symmetric_zeta, sys_1_site_0_shielding_symmetric_eta) = -0.601
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta) = -0.448
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_4_Scale_factor)            =  0.442
    C(sys_0_abundance, SP_0_operation_2_Exponential_FWHM)                          = -0.441
    C(SP_0_operation_2_Exponential_FWHM, SP_0_operation_4_Scale_factor)            =  0.409
    C(sys_0_abundance, SP_0_operation_1_Exponential_FWHM)                          =  0.398
    C(sys_0_abundance, SP_0_operation_4_Scale_factor)                              = -0.242
    C(sys_0_abundance, sys_1_site_0_shielding_symmetric_zeta)                      = -0.224
    C(sys_1_site_0_shielding_symmetric_zeta, SP_0_operation_4_Scale_factor)        =  0.209
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_1_Exponential_FWHM)    =  0.178
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_4_Scale_factor)        =  0.149
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_abundance)                      =  0.136
    C(sys_0_abundance, sys_1_site_0_shielding_symmetric_eta)                       = -0.134
    C(sys_1_site_0_shielding_symmetric_eta, SP_0_operation_4_Scale_factor)         =  0.125
```

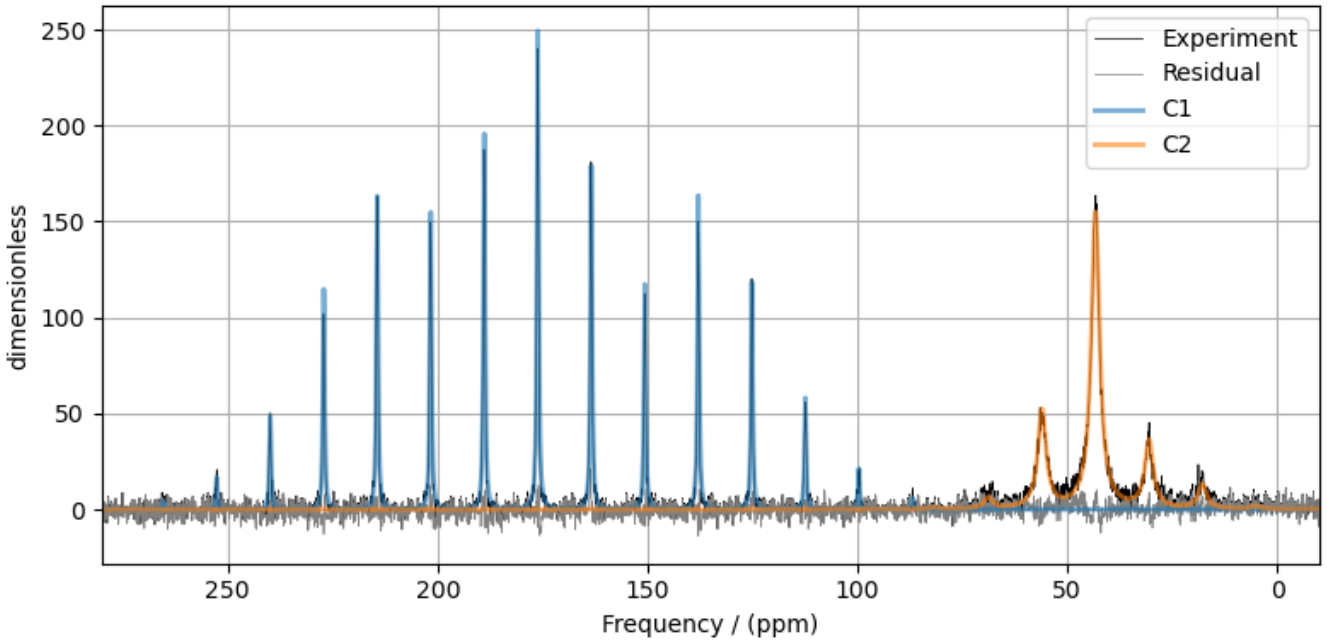
The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6)
ax.set_xlim(280, -10)
plt.grid()
```

(continues on next page)

(continued from previous page)

```
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 4.026 seconds)

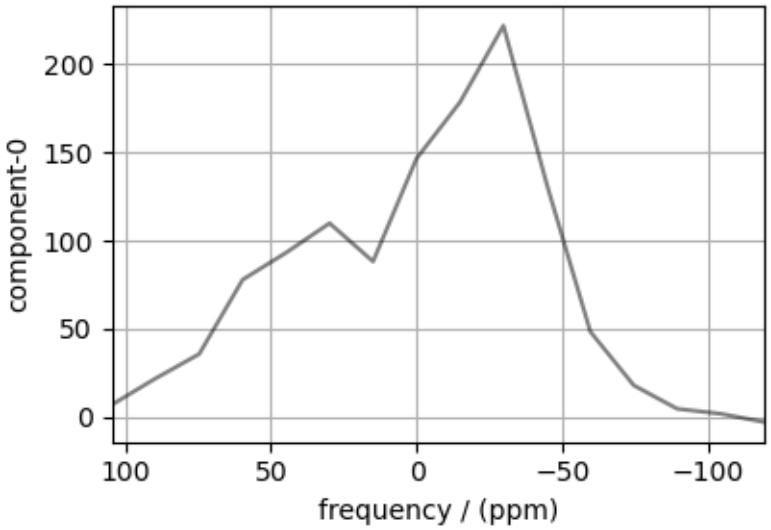
12.1.7 1D PASS/MAT sideband order cross-section

This example illustrates the use of mrsimulator and LMFIT modules in fitting the sideband intensity profile across the isotropic chemical shift cross-section from a PASS/MAT dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
name = "https://sandbox.zenodo.org/record/814455/files/LHistidine_cross_section.csdf"
pass_cross_section = cp.load(name)

# standard deviation of noise from the dataset
sigma = 4.640351

# For the spectral fitting, we only focus on the real part of the complex dataset.
pass_cross_section = pass_cross_section.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in pass_cross_section.dimensions]


# The plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(pass_cross_section, "k", alpha=0.5)
ax.invert_xaxis()
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Guess model

Create a guess list of spin systems. For fitting the sideband profile at an isotropic chemical shift cross-section from PASS/MAT datasets, set the isotropic_chemical_shift parameter of the site object as zero.

```
site = Site(
    isotope="13C",
    isotropic_chemical_shift=0,  #
```

(continues on next page)

(continued from previous page)

```
    shielding_symmetric={"zeta": -70, "eta": 0.8},
)
spin_systems = [SpinSystem(sites=[site])]
```

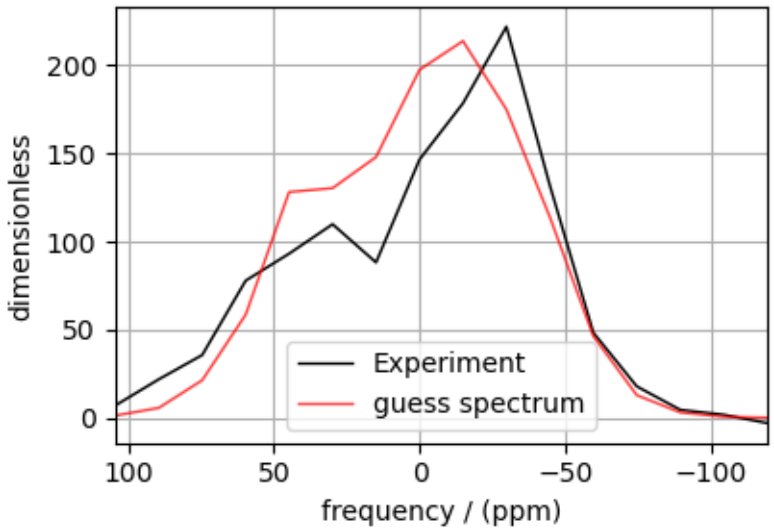
Method

For the sideband-only cross-section, use the BlochDecaySpectrum method.

```
# Get the dimension information from the experiment.
spectral_dims = get_spectral_dimensions(pass_cross_section)

PASS = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=1500,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=pass_cross_section,  # also add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = PASS.get_transition_pathways(sys)
```

Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[PASS])
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(operations=[sp.Scale(factor=2000)])
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(pass_cross_section, color="k", linewidth=1, label="Experiment")
ax.plot(processed_data, color="r", alpha=0.75, linewidth=1, label="guess spectrum")
plt.grid()
ax.invert_xaxis()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

First, create the fitting parameters. Use the *make_LMFIT_params()* (page 351) for a quick setup.

```
params = sf.make_LMFIT_params(sim, processor)

# Fix the value of the isotropic chemical shift to zero for pure anisotropic sideband
# amplitude simulation.
params["sys_0_site_0_isotropic_chemical_shift"].vary = False
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_0_Scale_factor            2000     -inf      inf     True     None
sys_0_abundance                           100        0      100    False      100
sys_0_site_0_isotropic_chemical_shift       0     -inf      inf    False     None
sys_0_site_0_shielding_symmetric_eta      0.8        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta     -70     -inf      inf     True     None
None
```

Run the minimization using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 37
    # data points      = 16
    # variables        = 3
```

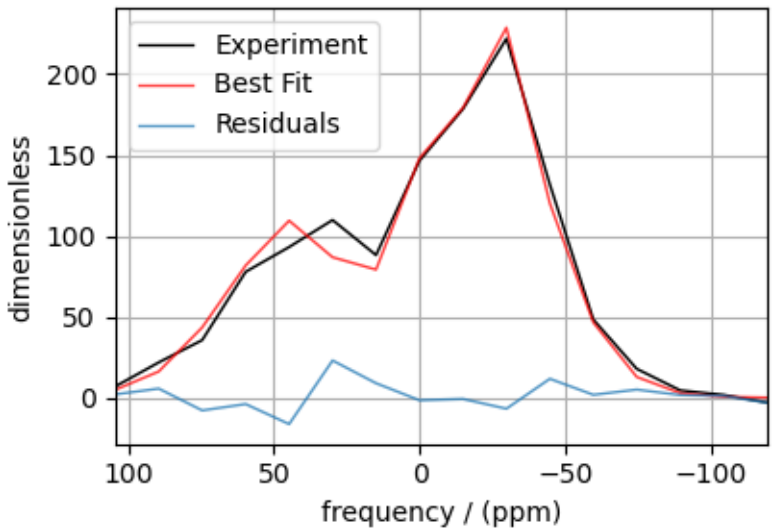
(continues on next page)

(continued from previous page)

```
    chi-square         = 56.5116594
    reduced chi-square = 4.34705072
    Akaike info crit   = 26.1897321
    Bayesian info crit = 28.5074983
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  0 (fixed)
    sys_0_site_0_shielding_symmetric_zeta: -85.1018564 +/- 1.52031809 (1.79%) (init = -70)
    sys_0_site_0_shielding_symmetric_eta:   0.46192300 +/- 0.03083948 (6.68%) (init = 0.8)
    sys_0_abundance:                        100.000000 +/- 0.00000000 (0.00%) == '100'
    SP_0_operation_0_Scale_factor:          1850.35089 +/- 48.9742939 (2.65%) (init = 2000)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta) =  0.374
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_0_Scale_factor)        = -0.231
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_0_Scale_factor)         =  0.206
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(pass_cross_section, color="k", linewidth=1, label="Experiment")
ax.plot(best_fit, "r", alpha=0.75, linewidth=1, label="Best Fit")
ax.plot(residuals, alpha=0.75, linewidth=1, label="Residuals")
ax.invert_xaxis()
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 1.717 seconds)

12.1.8 ^{17}O MAS NMR of crystalline Na_2SiO_3 (2nd order quad)

In this example, we illustrate the use of the mrsimulator objects to

- create a quadrupolar fitting model using Simulator and SignalProcessor objects,
- use the fitting model to perform a least-squares analysis, and
- extract the fitting parameters from the model.

We use the LMFIT library to fit the spectrum. The following example shows the least-squares fitting procedure applied to the ^{17}O MAS NMR spectrum of Na_2SiO_3[1].

Start by importing the relevant modules.

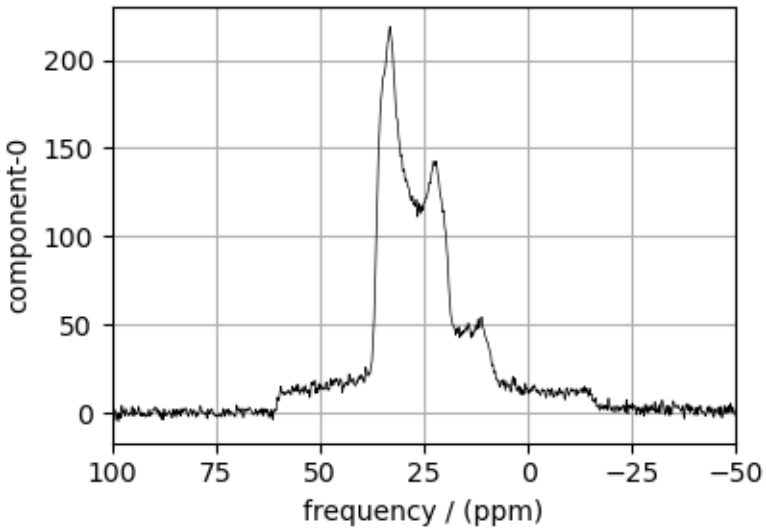
```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecayCTSpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

Import the experimental data. We use dataset file serialized with the CSDM file-format, using the csdmpy module.

```
filename = "https://sandbox.zenodo.org/record/814455/files/Na2SiO3_O17.csdf"
experiment = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 1.931335

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the dimension coordinates from Hz to ppm.
experiment.x[0].to("ppm", "nmr_frequency_ratio")

# plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(100, -50)
plt.grid()
plt.tight_layout()
plt.show()
```

[1] T. M. Clark, P. Florian, J. F. Stebbins, and P. J. Grandinetti, An ^{17}O NMR Investigation of Crystalline Sodium Metasilicate: Implications for the Determination of Local Structure in Alkali Silicates, J. Phys. Chem. B. 2001, **105**, 12257-12265. DOI: 10.1021/jp011289p



Create a fitting model

A fitting model is a composite of `Simulator` and `SignalProcessor` objects.

Step 1: Create initial guess sites and spin systems

```
O1 = Site(
    isotope="17O",
    isotropic_chemical_shift=60.0,  # in ppm,
    quadrupolar={"Cq": 4.2e6, "eta": 0.5},  # Cq in Hz
)

O2 = Site(
    isotope="17O",
    isotropic_chemical_shift=40.0,  # in ppm,
    quadrupolar={"Cq": 2.4e6, "eta": 0},  # Cq in Hz
)

spin_systems = [
    SpinSystem(sites=[O1], abundance=50, name="O1"),
    SpinSystem(sites=[O2], abundance=50, name="O2"),
]
```

Step 2: Create the method object. Create an appropriate method object that closely resembles the technique used in acquiring the experimental data. The attribute values of this method must meet the experimental conditions, including the acquisition channels, the magnetic flux density, rotor angle, rotor frequency, and the spectral/spectroscopic dimension.

In the following example, we set up a central transition selective Bloch decay spectrum method where the spectral/spectroscopic dimension information, i.e., count, spectral_width, and the reference_offset, is extracted from the CSDM dimension metadata using the *get_spectral_dimensions()* (page 340) utility function. The remaining attribute values are set to the experimental conditions.

```
# get the count, spectral_width, and reference_offset information from the experiment.
spectral_dims = get_spectral_dimensions(experiment)
```

(continues on next page)

(continued from previous page)

```
MAS_CT = BlochDecayCTSpectrum(
    channels=["17O"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=14000,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # experimental dataset
)

# A method object queries every spin system for a list of transition pathways that are
# relevant for the given method. Since the method and the number of spin systems remain
# the same during the least-squares fit, a one-time query is sufficient. To avoid
# querying for the transition pathways at every iteration in a least-squares fitting,
# evaluate the transition pathways once and store it as follows
for sys in spin_systems:
    sys.transition_pathways = MAS_CT.get_transition_pathways(sys)
```

Step 3: Create the Simulator object and add the method and spin system objects.

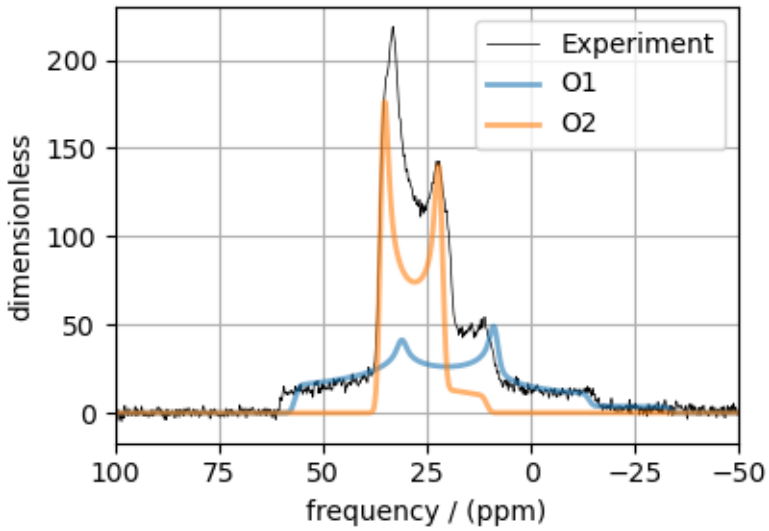
```
sim = Simulator(spin_systems=spin_systems, methods=[MAS_CT])
sim.config.decompose_spectrum = "spin_system"
sim.run()
```

Step 4: Create a SignalProcessor class object and apply the post-simulation signal processing operations.

```
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Gaussian(FWHM="100 Hz"),
        sp.FFT(),
        sp.Scale(factor=200.0),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real
```

Step 5: The plot of the data and the guess spectrum.

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6)
ax.set_xlim(100, -50)
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Once you have a fitting model, you need to create the list of parameters to use in the least-squares fitting. For this, you may use the Parameters class from *LMFIT*, as described in the previous example. Here, we make use of a utility function, *make_LMFIT_params()* (page 351), that considerably simplifies the LMFIT parameters generation process.

Step 6: Create a list of parameters.

```
params = sf.make_LMFIT_params(sim, processor)
```

The *make_LMFIT_params* parses the instances of the `Simulator` and the `PostSimulator` objects for parameters and returns a LMFIT *Parameters* object.

Customize the Parameters: You may customize the parameters list, `params`, as desired. Here, we remove the abundance of the two spin systems and constrain it to the initial value of 50% each, and constrain *eta=0* for spin system at index 1.

```
params.pop("sys_0_abundance")
params.pop("sys_1_abundance")
params["sys_1_site_0_quadrupolar_eta"].vary = False
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM            100     -inf      inf     True     None
SP_0_operation_3_Scale_factor             200     -inf      inf     True     None
sys_0_site_0_isotropic_chemical_shift      60     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq           4.2e+06     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta              0.5        0        1     True     None
sys_1_site_0_isotropic_chemical_shift      40     -inf      inf     True     None
sys_1_site_0_quadrupolar_Cq           2.4e+06     -inf      inf     True     None
sys_1_site_0_quadrupolar_eta                0        0        1    False     None
None
```

Step 7: Perform least-squares minimization. For the user's convenience, we also provide a utility function, *LMFIT_min_function()* (page 351), for evaluating the difference vector between the simulation and experiment, based on the parameters update. You may use this function directly as the argument of the LMFIT Minimizer class, as follows,

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 1453
    # data points      = 4096
    # variables        = 7
    chi-square         = 18809.7754
    reduced chi-square = 4.60009181
    Akaike info crit   = 6257.80238
    Bayesian info crit = 6302.02675
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  63.5770881 +/- 0.15628358 (0.25%) (init = 60)
    sys_0_site_0_quadrupolar_Cq:            4272350.04 +/- 7204.55337 (0.17%) (init = 4200000)
    sys_0_site_0_quadrupolar_eta:           0.52518337 +/- 0.00394588 (0.75%) (init = 0.5)
    sys_1_site_0_isotropic_chemical_shift:  39.3485493 +/- 0.02270983 (0.06%) (init = 40)
    sys_1_site_0_quadrupolar_Cq:            2400520.35 +/- 2100.59889 (0.09%) (init = 2400000)
    sys_1_site_0_quadrupolar_eta:           0 (fixed)
    SP_0_operation_1_Gaussian_FWHM:         176.959365 +/- 1.63340922 (0.92%) (init = 100)
    SP_0_operation_3_Scale_factor:          234.753247 +/- 0.51823251 (0.22%) (init = 200)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_Cq)           =  0.906
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_Cq)           =  0.831
    C(sys_0_site_0_quadrupolar_eta, sys_1_site_0_isotropic_chemical_shift)          =  0.549
    C(sys_0_site_0_quadrupolar_eta, sys_1_site_0_quadrupolar_Cq)                    =  0.395
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_1_Gaussian_FWHM)                 = -0.383
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_3_Scale_factor)                   =  0.378
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_3_Scale_factor)         =  0.341
    C(sys_1_site_0_isotropic_chemical_shift, SP_0_operation_1_Gaussian_FWHM)        = -0.327
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_3_Scale_factor)                =  0.279
    C(sys_0_site_0_isotropic_chemical_shift, sys_1_site_0_isotropic_chemical_shift) = -0.274
    C(sys_0_site_0_quadrupolar_Cq, sys_1_site_0_isotropic_chemical_shift)           = -0.272
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_1_Gaussian_FWHM)        =  0.264
    C(sys_1_site_0_quadrupolar_Cq, SP_0_operation_1_Gaussian_FWHM)                  = -0.260
    C(sys_0_site_0_quadrupolar_Cq, sys_0_site_0_quadrupolar_eta)                    = -0.257
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_1_Gaussian_FWHM)                  =  0.257
    C(sys_0_site_0_quadrupolar_Cq, sys_1_site_0_quadrupolar_Cq)                     = -0.189
    C(sys_0_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_Cq)           = -0.184
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_eta)          = -0.135
```

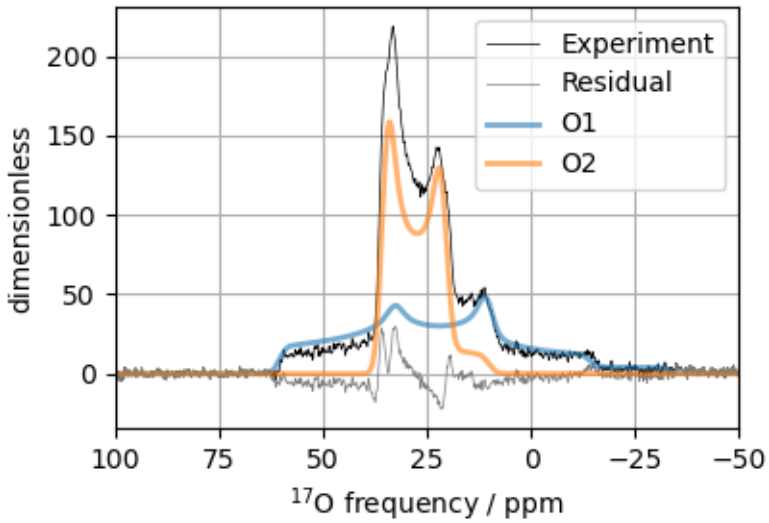
Step 8: The plot of the fit and the measurement data.

```
# Best fit spectrum
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]
```

(continues on next page)

(continued from previous page)

```
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6)
ax.set_xlabel("$^{17}$O frequency / ppm")
ax.set_xlim(100, -50)
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 48.098 seconds)

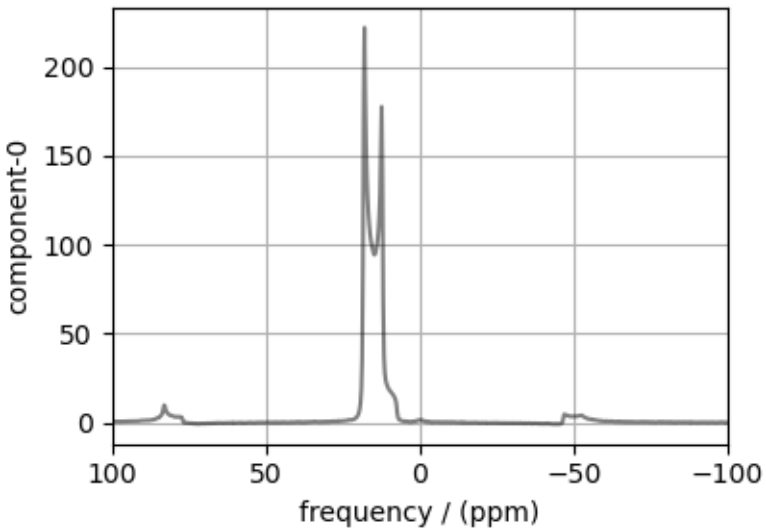
12.1.9 ^{11}B MAS NMR of Lithium orthoborate crystal

The following is a quadrupolar lineshape fitting example for the 11B MAS NMR of lithium orthoborate crystal. The dataset was shared by Dr. Nathan Barrow.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, Site, SpinSystem
from mrsimulator.methods import BlochDecayCTSpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/814455/files/11B_lithum_orthoborate.csdf"
experiment = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 0.08078374

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, "k", alpha=0.5)
ax.set_xlim(100, -100)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Spin System

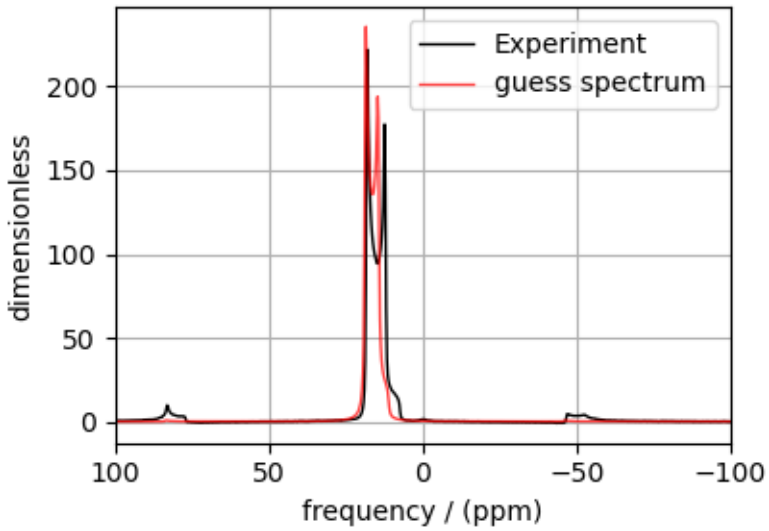
```
B11 = Site(
    isotope="11B",
    isotropic_chemical_shift=20.0,  # in ppm
    quadrupolar={"Cq": 2.3e6, "eta": 0.03},  # Cq in Hz
)
spin_systems = [SpinSystem(sites=[B11])]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS_CT = BlochDecayCTSpectrum(
    channels=["11B"],
    magnetic_flux_density=14.1,  # in T
    rotor_frequency=12500,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS_CT.get_transition_pathways(sys)
```

Guess Model Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MAS_CT])
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="100 Hz"),
        sp.FFT(),
        sp.Scale(factor=200),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, "k", linewidth=1, label="Experiment")
ax.plot(processed_data, "r", alpha=0.75, linewidth=1, label="guess spectrum")
ax.set_xlim(100, -100)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
params.pop("sys_0_abundance")
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                      Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM           100     -inf      inf     True     None
SP_0_operation_3_Scale_factor               200     -inf      inf     True     None
sys_0_site_0_isotropic_chemical_shift        20     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq             2.3e+06     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta               0.03        0        1     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 74
    # data points      = 4096
    # variables        = 5
    chi-square         = 203147.651
    reduced chi-square = 49.6572113
    Akaike info crit   = 16000.4652
```

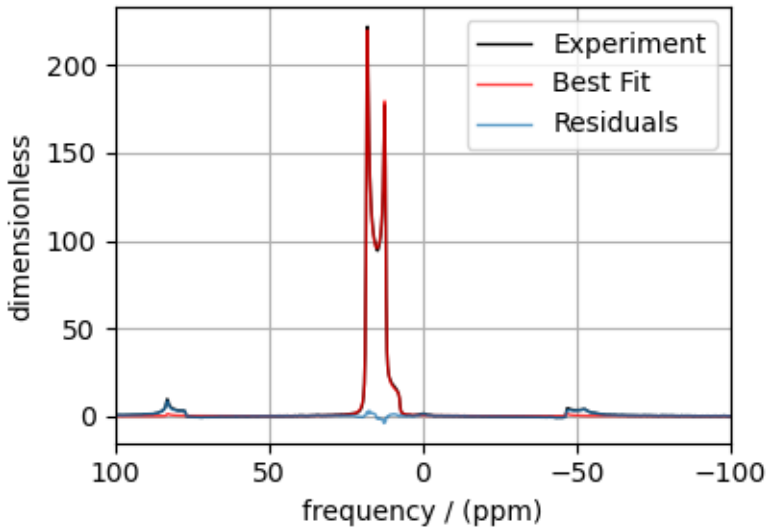
(continues on next page)

(continued from previous page)

```
    Bayesian info crit = 16032.0541
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  20.0547392 +/- 0.00152259 (0.01%) (init = 20)
    sys_0_site_0_quadrupolar_Cq:            2732039.56 +/- 431.874598 (0.02%) (init = 2300000)
    sys_0_site_0_quadrupolar_eta:           0.05154982 +/- 8.5913e-04 (1.67%) (init = 0.03)
    SP_0_operation_1_Exponential_FWHM:      68.1956038 +/- 0.70874729 (1.04%) (init = 100)
    SP_0_operation_3_Scale_factor:          199.046699 +/- 0.21335695 (0.11%) (init = 200)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_1_Exponential_FWHM)          = -0.824
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_Cq)       =  0.806
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_3_Scale_factor)         =  0.561
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_3_Scale_factor)              = -0.367
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_1_Exponential_FWHM) = -0.127
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, "k", linewidth=1, label="Experiment")
ax.plot(best_fit, "r", alpha=0.75, linewidth=1, label="Best Fit")
ax.plot(residuals, alpha=0.75, linewidth=1, label="Residuals")
ax.set_xlim(100, -100)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 3.488 seconds)

12.1.10 ^{23}Na MAS NMR of Nasicon

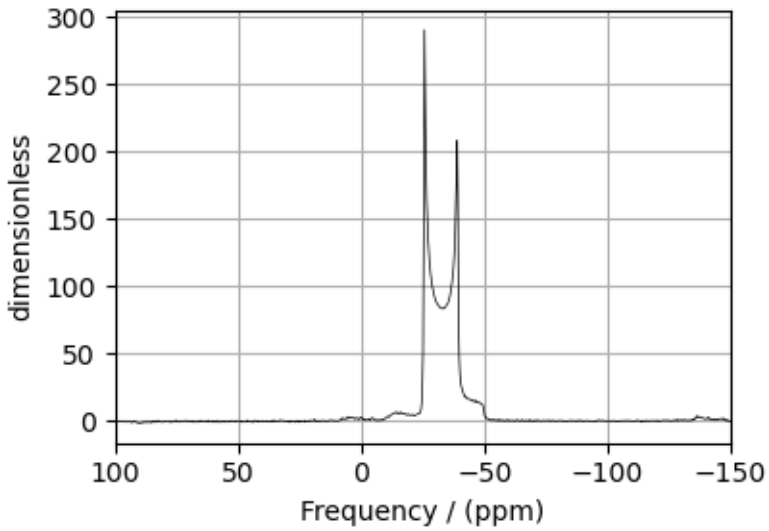
The following is a least-squares fitting example of a ^{23}Na MAS NMR spectrum of Nasicon, $NaZr_2(PO_4)_3$. The following experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, Site, SpinSystem
from mrsimulator.methods import BlochDecayCTSpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "23Na QUAD MAS Nasicon.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 0.2368151

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(100, -150)
plt.grid()
plt.tight_layout()
plt.show()
```

[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984



Create a fitting model

Spin System

```
Na23 = Site(
    isotope="23Na",
    isotropic_chemical_shift=-20.0,  # in ppm
    quadrupolar={"Cq": 2.3e6, "eta": 0.03},  # Cq in Hz
)
spin_systems = [SpinSystem(sites=[Na23])]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS_CT = BlochDecayCTSpectrum(
    channels=["23Na"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=15000,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS_CT.get_transition_pathways(sys)
```

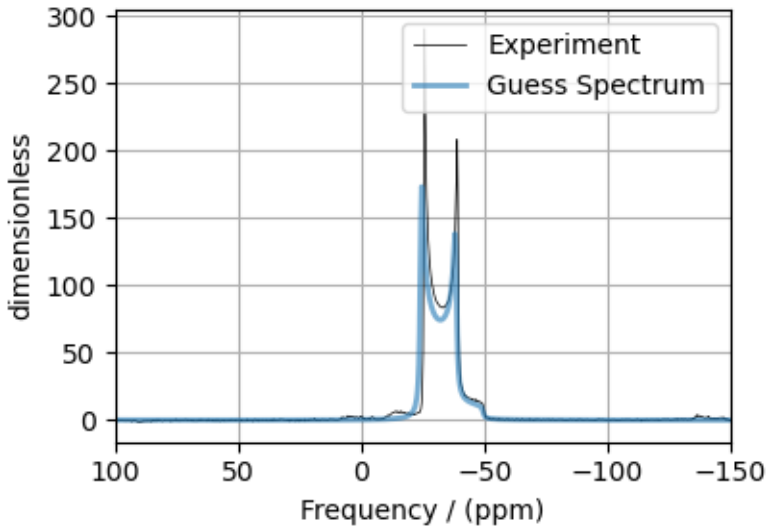
Guess Model Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MAS_CT])
```

(continues on next page)

(continued from previous page)

```
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="100 Hz"),
        sp.FFT(),
        sp.Scale(factor=200),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6, label="Guess Spectrum")
ax.set_xlim(100, -150)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                     Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM          100     -inf      inf     True     None
SP_0_operation_3_Scale_factor              200     -inf      inf     True     None
sys_0_abundance                            100        0      100    False      100
sys_0_site_0_isotropic_chemical_shift      -20     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq            2.3e+06     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta              0.03        0        1     True     None
None
```

Solve the minimizer using LMFIT

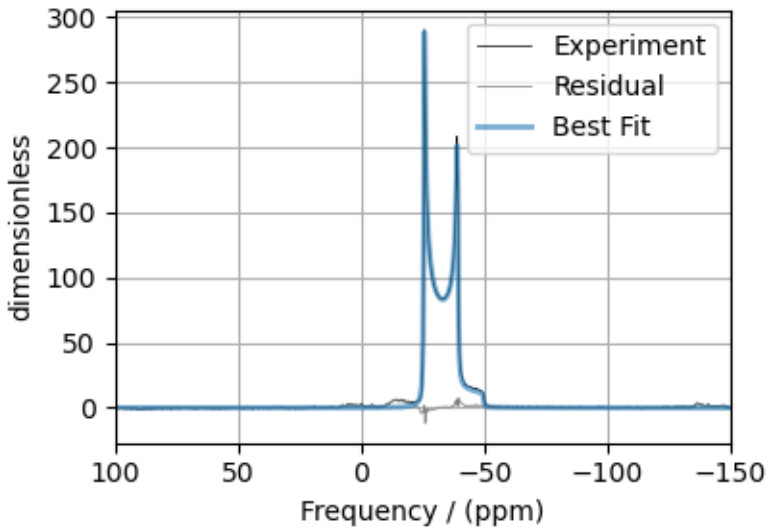
```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 56
    # data points      = 4096
    # variables        = 5
    chi-square         = 50387.1068
    reduced chi-square = 12.3165746
    Akaike info crit   = 10289.8313
    Bayesian info crit = 10321.4201
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift: -21.4706356 +/- 0.00298717 (0.01%) (init = -20)
    sys_0_site_0_quadrupolar_Cq:            2252000.71 +/- 205.098216 (0.01%) (init = 2300000)
    sys_0_site_0_quadrupolar_eta:           0.00341673 +/- 6.9412e-04 (20.32%) (init = 0.03)
    sys_0_abundance:                        100.000000 +/- 0.00000000 (0.00%) == '100'
    SP_0_operation_1_Exponential_FWHM:      45.7664986 +/- 0.38445046 (0.84%) (init = 100)
    SP_0_operation_3_Scale_factor:          213.721815 +/- 0.23474326 (0.11%) (init = 200)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_eta)      = -0.877
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_Cq)       =  0.876
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_1_Exponential_FWHM)          = -0.722
    C(sys_0_site_0_quadrupolar_Cq, sys_0_site_0_quadrupolar_eta)                = -0.680
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_1_Exponential_FWHM) =  0.505
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_3_Scale_factor)         =  0.470
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_1_Exponential_FWHM)           =  0.384
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_3_Scale_factor)              = -0.240
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_3_Scale_factor)     =  0.187
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_3_Scale_factor)               =  0.168
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6, label="Best Fit")
ax.set_xlim(100, -150)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 2.441 seconds)

12.1.11 ^{27}Al MAS NMR of YAG (1st and 2nd order Quad)

The following is a quadrupolar lineshape fitting example for the 27Al MAS NMR of Yttrium aluminum garnet (YAG) crystal. The following experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, Site, SpinSystem
from mrsimulator.methods import BlochDecaySpectrum
```

(continues on next page)

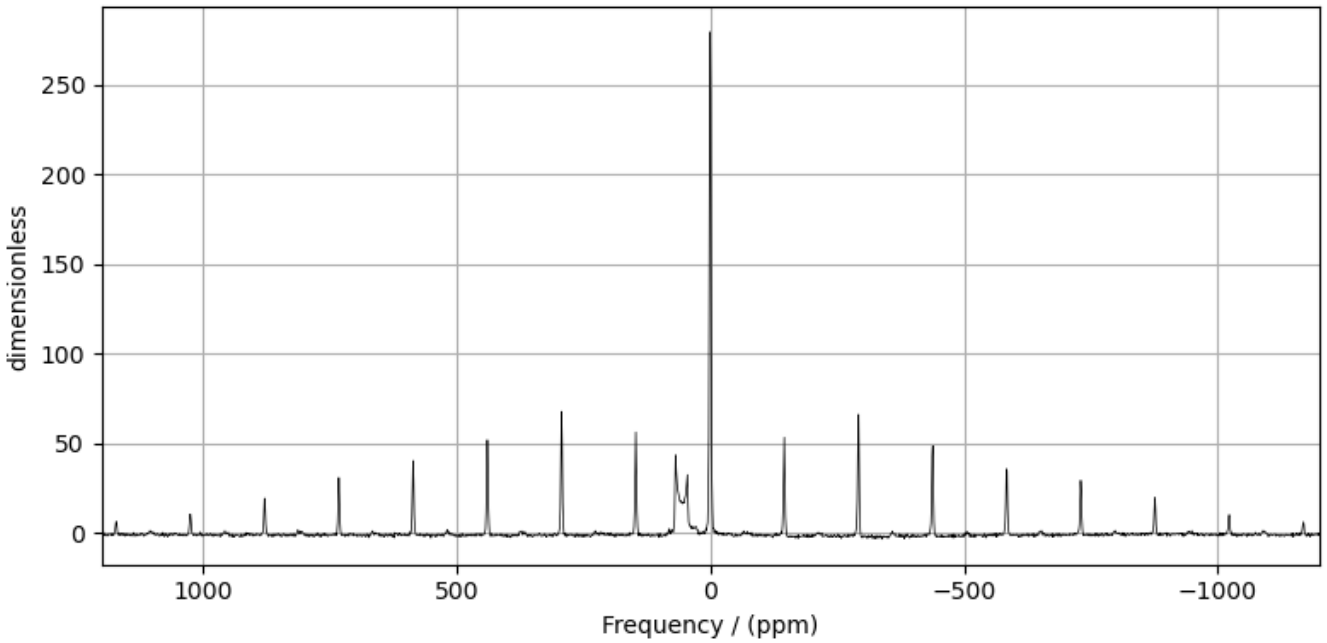
[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984

(continued from previous page)

```
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "27Al Quad MAS YAG 400MHz.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 0.4895381

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(1200, -1200)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Guess model

Create a guess list of spin systems.

```
Al_1 = Site(
    isotope="27Al",
    isotropic_chemical_shift=76,  # in ppm
    quadrupolar={"Cq": 6e6, "eta": 0},  # Cq in Hz
)

Al_2 = Site(
    isotope="27Al",
    isotropic_chemical_shift=1,  # in ppm
    quadrupolar={"Cq": 5e5, "eta": 0.3},  # Cq in Hz
)
spin_systems = [
    SpinSystem(sites=[Al_1], name="AlO4"),
    SpinSystem(sites=[Al_2], name="AlO6"),
]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS = BlochDecaySpectrum(
    channels=["27Al"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=15250,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS.get_transition_pathways(sys)
```

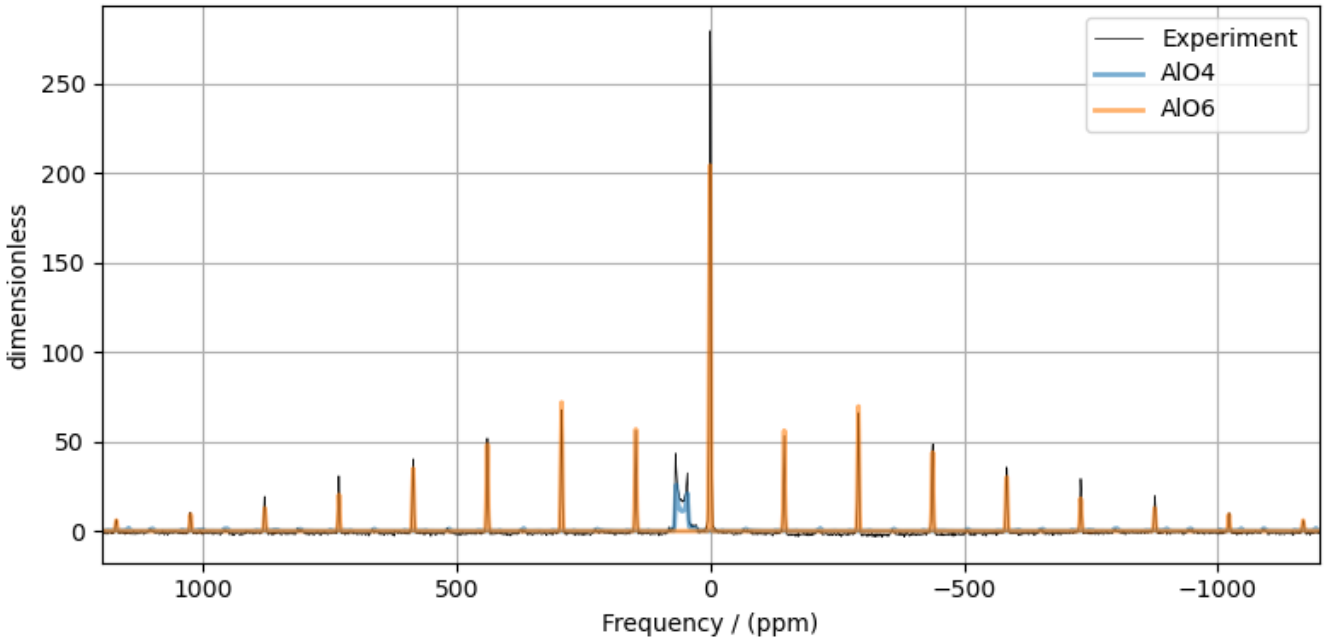
Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.config.decompose_spectrum = "spin_system"
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Gaussian(FWHM="300 Hz"),
```

(continues on next page)

(continued from previous page)

```
        sp.FFT(),
        sp.Scale(factor=50),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6)
ax.set_xlim(1200, -1200)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor, include={"rotor_frequency"})
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM            300     -inf      inf     True     None
```

(continues on next page)

(continued from previous page)

```
SP_0_operation_3_Scale_factor                50     -inf      inf     True     None
mth_0_rotor_frequency                 1.525e+04 1.515e+04 1.535e+04     True     None
sys_0_abundance                              50        0      100     True     None
sys_0_site_0_isotropic_chemical_shift        76     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq                6e+06     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta                  0        0        1     True     None
sys_1_abundance                              50        0      100    False 100-sys_0_abundance
sys_1_site_0_isotropic_chemical_shift         1     -inf      inf     True     None
sys_1_site_0_quadrupolar_Cq                5e+05     -inf      inf     True     None
sys_1_site_0_quadrupolar_eta                 0.3        0        1     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 80
    # data points      = 4096
    # variables        = 10
    chi-square         = 53146.1058
    reduced chi-square = 13.0068786
    Akaike info crit   = 10518.1870
    Bayesian info crit = 10581.3647
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  76.8006363 +/- 0.09331740 (0.12%) (init = 76)
    sys_0_site_0_quadrupolar_Cq:            6013377.50 +/- 13886.0906 (0.23%) (init = 6000000)
    sys_0_site_0_quadrupolar_eta:           0.04554465 +/- 0.00876802 (19.25%) (init = 0)
    sys_0_abundance:                        47.7990936 +/- 0.45739973 (0.96%) (init = 50)
    sys_1_site_0_isotropic_chemical_shift:  0.84460059 +/- 0.00321030 (0.38%) (init = 1)
    sys_1_site_0_quadrupolar_Cq:            586585.801 +/- 4319.84316 (0.74%) (init = 500000)
    sys_1_site_0_quadrupolar_eta:           0.77209876 +/- 0.01784074 (2.31%) (init = 0.3)
    sys_1_abundance:                        52.2009064 +/- 0.45739973 (0.88%) == '100-sys_0_
↪abundance'
    mth_0_rotor_frequency:                  15251.3275 +/- 0.19496683 (0.00%) (init = 15250)
    SP_0_operation_1_Gaussian_FWHM:         319.813723 +/- 2.04528263 (0.64%) (init = 300)
    SP_0_operation_3_Scale_factor:          141.804979 +/- 1.22714045 (0.87%) (init = 50)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_abundance, SP_0_operation_3_Scale_factor)                    =  0.813
    C(sys_1_site_0_quadrupolar_Cq, sys_1_site_0_quadrupolar_eta)         = -0.729
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_Cq)  =  0.609
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_Cq)  =  0.563
    C(sys_1_site_0_isotropic_chemical_shift, mth_0_rotor_frequency)      = -0.472
    C(sys_1_site_0_quadrupolar_Cq, mth_0_rotor_frequency)                = -0.348
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_eta) = -0.199
    C(sys_1_site_0_quadrupolar_eta, mth_0_rotor_frequency)               =  0.168
    C(sys_0_abundance, sys_1_site_0_quadrupolar_Cq)                      = -0.165
```

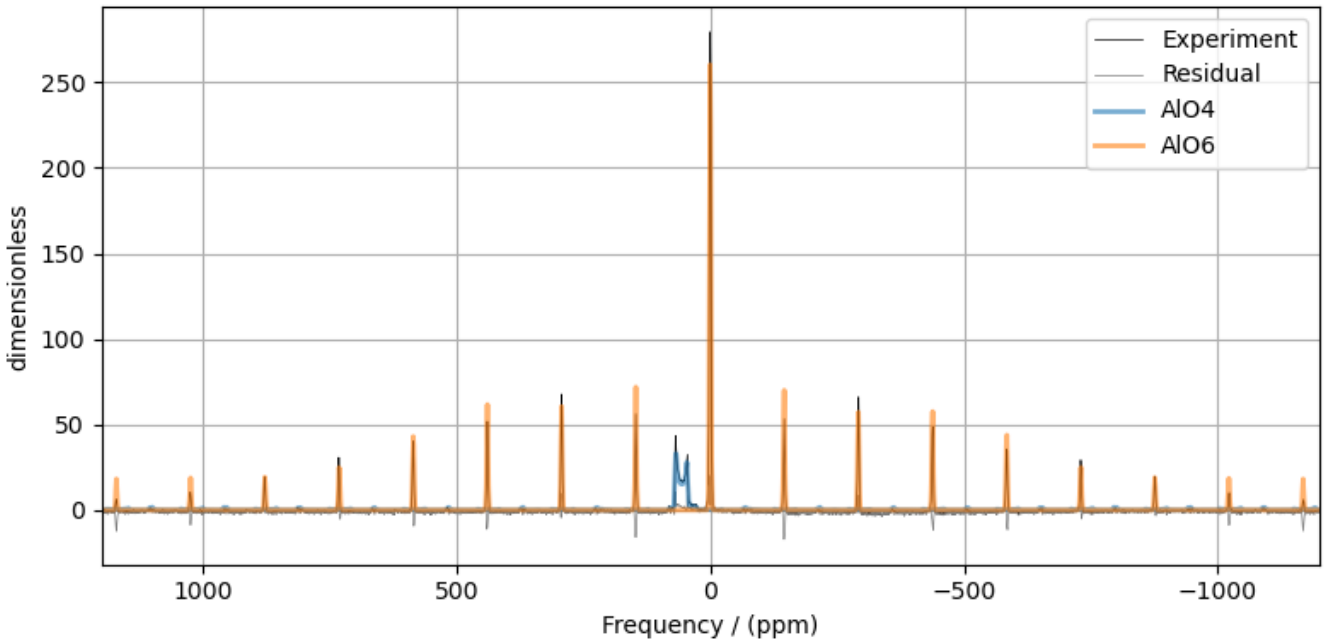
(continues on next page)

(continued from previous page)

```
    C(sys_1_site_0_quadrupolar_Cq, SP_0_operation_1_Gaussian_FWHM)          = -0.158
    C(sys_0_site_0_quadrupolar_eta, sys_0_abundance)                        =  0.146
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_3_Scale_factor)        =  0.146
    C(sys_0_abundance, sys_1_site_0_quadrupolar_eta)                        =  0.143
    C(sys_1_site_0_quadrupolar_Cq, SP_0_operation_3_Scale_factor)           =  0.141
    C(sys_0_site_0_quadrupolar_Cq, sys_0_abundance)                         =  0.133
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_3_Scale_factor)           =  0.132
    C(sys_1_site_0_quadrupolar_eta, SP_0_operation_3_Scale_factor)          = -0.128
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_3_Scale_factor)          =  0.123
    C(sys_0_abundance, SP_0_operation_1_Gaussian_FWHM)                      = -0.109
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(8, 4))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6)
ax.set_xlim(1200, -1200)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 10.692 seconds)

12.1.12 ^{2}H MAS NMR of Methionine

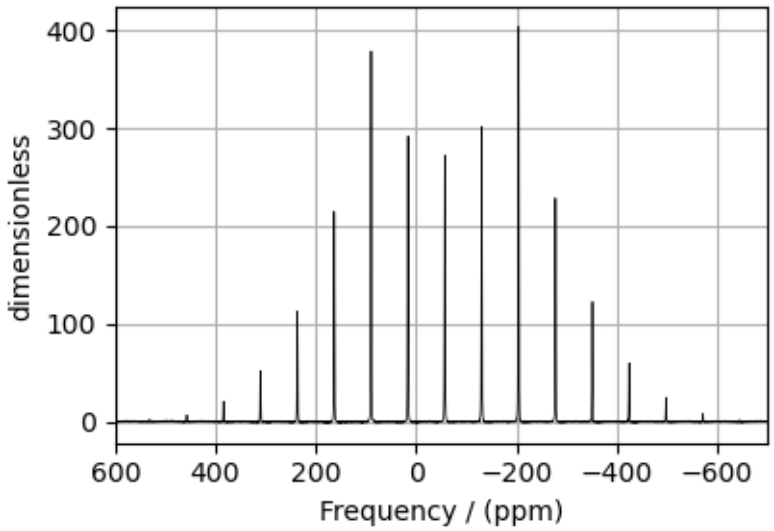
The following is a least-squares fitting example of a ^{2}H MAS NMR spectrum of Methionine. The experimental dataset is a part of DMFIT[1] examples. We thank Dr. Dominique Massiot for sharing the dataset.

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "2H methiodine MAS.csdf"
experiment = cp.load(host + filename)

# standard deviation of noise from the dataset
sigma = 0.3026282

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.set_xlim(600, -700)
plt.grid()
plt.tight_layout()
plt.show()
```

[1] D.Massiot, F.Fayon, M.Capron, I.King, S.Le Calvé, B.Alonso, J.O.Durand, B.Bujoli, Z.Gan, G.Hoatson, 'Modelling one and two-dimensional solid-state NMR spectra.', Magn. Reson. Chem. **40** 70-76 (2002) DOI: 10.1002/mrc.984



Create a fitting model

Spin System

```
H_2 = Site(
    isotope="2H",
    isotropic_chemical_shift=-57.12,  # in ppm,
    quadrupolar={"Cq": 3e4, "eta": 0},  # Cq in Hz
)

spin_systems = [SpinSystem(sites=[H_2])]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS = BlochDecaySpectrum(
    channels=["2H"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=4517.1,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # experimental dataset
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS.get_transition_pathways(sys)
```

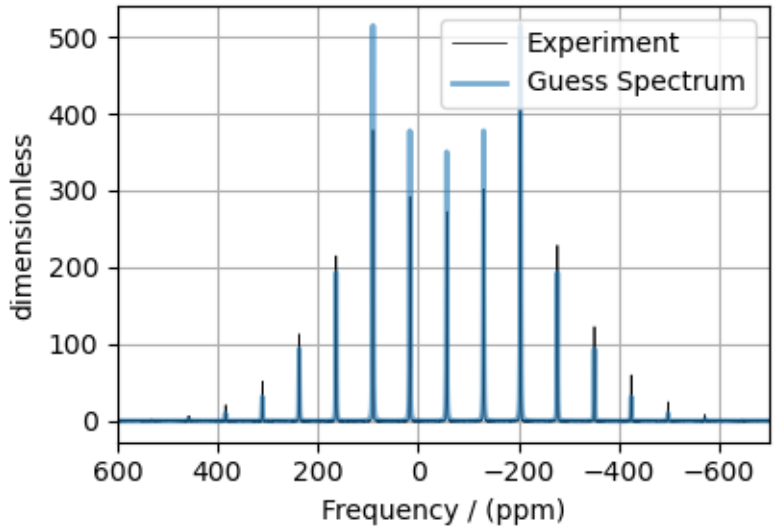
Guess Model Spectrum

```
# Simulation
# ----------
```

(continues on next page)

(continued from previous page)

```
sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="60 Hz"),
        sp.FFT(),
        sp.Scale(factor=140),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(processed_data, linewidth=2, alpha=0.6, label="Guess Spectrum")
ax.set_xlim(600, -700)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
params["sys_0_site_0_isotropic_chemical_shift"].vary = False
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM          60     -inf      inf     True     None
SP_0_operation_3_Scale_factor             140     -inf      inf     True     None
sys_0_abundance                           100        0      100    False      100
sys_0_site_0_isotropic_chemical_shift  -57.12     -inf      inf    False     None
sys_0_site_0_quadrupolar_Cq             3e+04     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta                0        0        1     True     None
None
```

Solve the minimizer using LMFIT

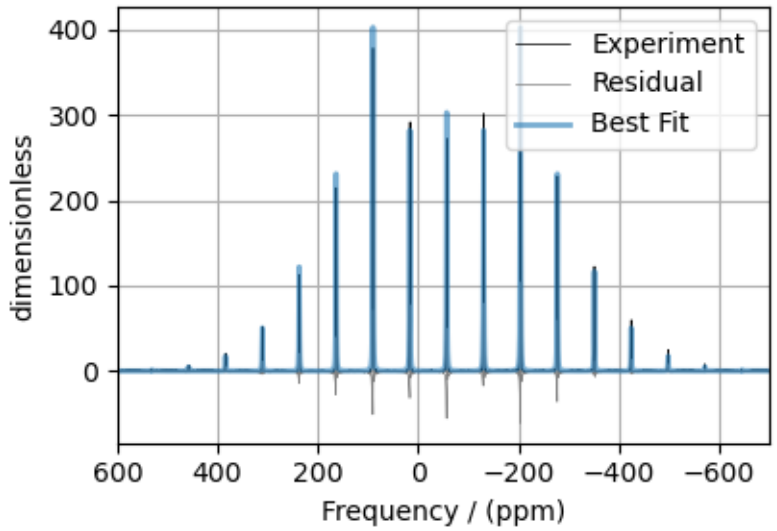
```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 32
    # data points      = 8192
    # variables        = 4
    chi-square         = 1350081.30
    reduced chi-square = 164.885356
    Akaike info crit   = 41826.2105
    Bayesian info crit = 41854.2541
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift: -57.12 (fixed)
    sys_0_site_0_quadrupolar_Cq:            33590.4942 +/- 79.9716279 (0.24%) (init = 30000)
    sys_0_site_0_quadrupolar_eta:           0.32187196 +/- 0.00445150 (1.38%) (init = 0)
    sys_0_abundance:                        100.000000 +/- 0.00000000 (0.00%) == '100'
    SP_0_operation_1_Exponential_FWHM:      62.9466148 +/- 0.32110568 (0.51%) (init = 60)
    SP_0_operation_3_Scale_factor:          132.772217 +/- 0.50413058 (0.38%) (init = 140)
[[Correlations]] (unreported correlations are < 0.100)
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_3_Scale_factor) =  0.666
    C(sys_0_site_0_quadrupolar_Cq, sys_0_site_0_quadrupolar_eta)        = -0.274
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_3_Scale_factor)       =  0.262
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_3_Scale_factor)      =  0.112
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, color="black", linewidth=0.5, label="Experiment")
ax.plot(residuals, color="gray", linewidth=0.5, label="Residual")
ax.plot(best_fit, linewidth=2, alpha=0.6, label="Best Fit")
ax.set_xlim(600, -700)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 2.499 seconds)

12.1.13 ^{119}Sn MAS NMR of SnO

The following is a spinning sideband manifold fitting example for the 119Sn MAS NMR of SnO. The dataset was acquired and shared by Altenhof *et al.*[1].

```
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site, Coupling
from mrsimulator.methods import BlochDecaySpectrum
```

(continues on next page)

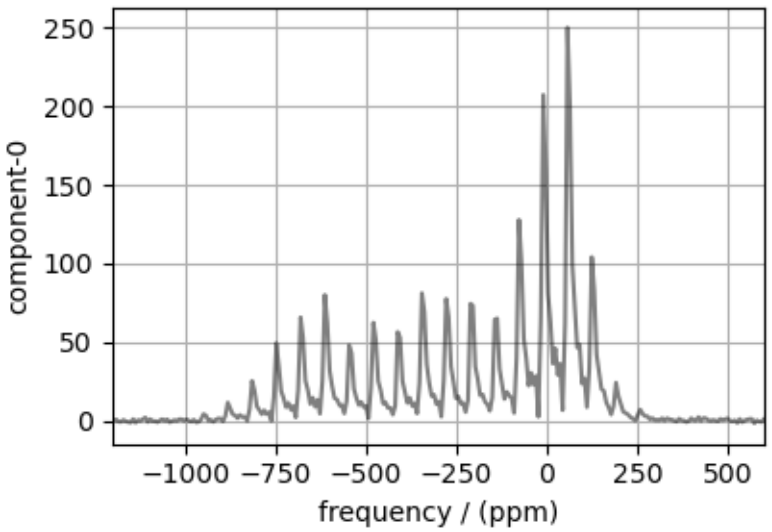
[1] Altenhof A. R., Jaroszewicz M. J., Lindquist A. W., Foster L. D. D., Veinberg S. L., and Schurko R. W. Practical Aspects of Recording Ultra-Wideline NMR Patterns under Magic-Angle Spinning Conditions. J. Phys. Chem. C. 2020, **124**, 27, 14730–14744 DOI: 10.1021/acs.jpcc.0c04510

(continued from previous page)

```
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/814455/files/119Sn_SnO.csdf"
experiment = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 0.6410905

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, "k", alpha=0.5)
ax.set_xlim(-1200, 600)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Guess model

Create a guess list of spin systems. There are two spin systems present in this example, - 1) an uncoupled ^{119}Sn and - 2) a coupled ^{119}Sn-^{117}Sn spin systems.

```
sn119 = Site(
    isotope="119Sn",
    isotropic_chemical_shift=-210,
    shielding_symmetric={"zeta": 700, "eta": 0.1},
)
sn117 = Site(
    isotope="117Sn",
    isotropic_chemical_shift=0,
)
j_sn = Coupling(
    site_index=[0, 1],
    isotropic_j=8150.0,
)

sn117_abundance = 7.68  # in %
spin_systems = [
    # uncoupled spin system
    SpinSystem(sites=[sn119], abundance=100 - sn117_abundance),
    # coupled spin systems
    SpinSystem(sites=[sn119, sn117], couplings=[j_sn], abundance=sn117_abundance),
]
```

Method

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MAS = BlochDecaySpectrum(
    channels=["119Sn"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=10000,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = MAS.get_transition_pathways(sys)
```

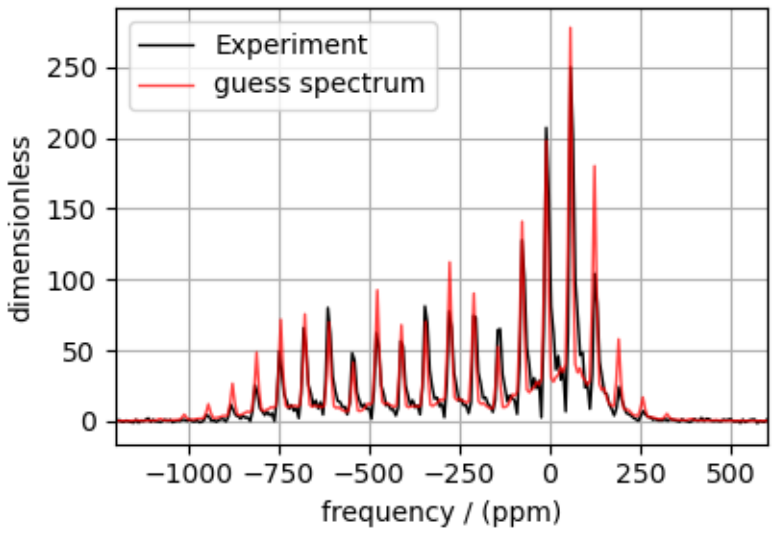
Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.run()
```

(continues on next page)

(continued from previous page)

```
# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="1500 Hz"),
        sp.FFT(),
        sp.Scale(factor=5000),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, "k", linewidth=1, label="Experiment")
ax.plot(processed_data, "r", alpha=0.75, linewidth=1, label="guess spectrum")
ax.set_xlim(-1200, 600)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor, include={"rotor_frequency"})

# Remove the abundance parameters from params. Since the measurement detects 119Sn, we
# also remove the isotropic chemical shift parameter of 117Sn site from params. The
# 117Sn is the site at index 1 of the spin system at index 1.
params.pop("sys_0_abundance")
params.pop("sys_1_abundance")
params.pop("sys_1_site_1_isotropic_chemical_shift")

# Since the 119Sn site is shared between the two spin systems, we add constraints to the
# 119Sn site parameters from the spin system at index 1 to be the same as 119Sn site
# parameters from the spin system at index 0.
lst = [
    "isotropic_chemical_shift",
    "shielding_symmetric_zeta",
    "shielding_symmetric_eta",
]
for item in lst:
    params[f"sys_1_site_0_{item}"].expr = f"sys_0_site_0_{item}"

print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM        1500     -inf      inf     True     None
SP_0_operation_3_Scale_factor            5000     -inf      inf     True     None
mth_0_rotor_frequency                   1e+04     9900 1.01e+04     True     None
sys_0_site_0_isotropic_chemical_shift    -210     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_eta      0.1        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta     700     -inf      inf     True     None
sys_1_coupling_0_isotropic_j             8150     -inf      inf     True     None
sys_1_site_0_isotropic_chemical_shift    -210     -inf      inf    False sys_0_site_0_
↪isotropic_chemical_shift
sys_1_site_0_shielding_symmetric_eta      0.1        0        1    False sys_0_site_0_
↪shielding_symmetric_eta
sys_1_site_0_shielding_symmetric_zeta     700     -inf      inf    False sys_0_site_0_
↪shielding_symmetric_zeta
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
```

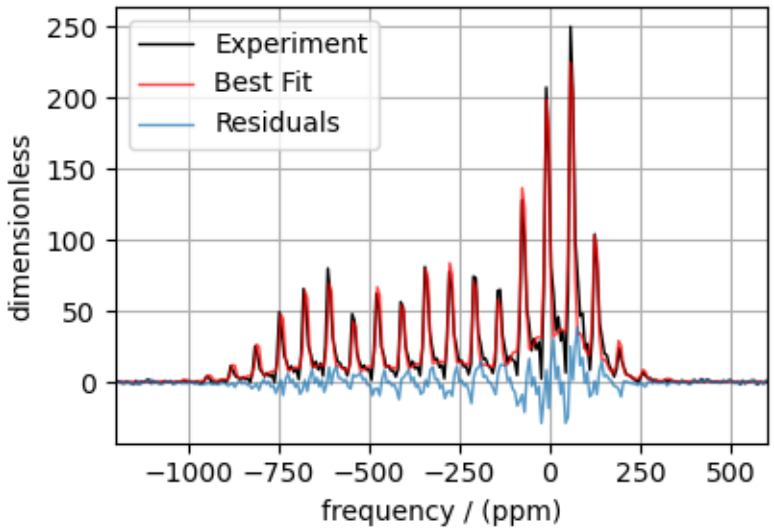
(continues on next page)

(continued from previous page)

```
    # function evals   = 65
    # data points      = 1000
    # variables        = 7
    chi-square         = 33637.4508
    reduced chi-square = 33.8745728
    Akaike info crit   = 3529.64005
    Bayesian info crit = 3563.99434
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift: -207.087914 +/- 0.05617017 (0.03%) (init = -210)
    sys_0_site_0_shielding_symmetric_zeta:  646.836036 +/- 1.65393478 (0.26%) (init = 700)
    sys_0_site_0_shielding_symmetric_eta:   0.10812411 +/- 0.01161823 (10.75%) (init = 0.1)
    sys_1_site_0_isotropic_chemical_shift: -207.087914 +/- 0.05617017 (0.03%) == 'sys_0_site_0_
↪isotropic_chemical_shift'
    sys_1_site_0_shielding_symmetric_zeta:  646.836036 +/- 1.65393478 (0.26%) == 'sys_0_site_0_
↪shielding_symmetric_zeta'
    sys_1_site_0_shielding_symmetric_eta:   0.10812411 +/- 0.01161823 (10.75%) == 'sys_0_site_0_
↪shielding_symmetric_eta'
    sys_1_coupling_0_isotropic_j:           8056.43478 +/- 111.179576 (1.38%) (init = 8150)
    mth_0_rotor_frequency:                  9998.34698 +/- 2.21701251 (0.02%) (init = 10000)
    SP_0_operation_1_Exponential_FWHM:      1349.03588 +/- 24.7137331 (1.83%) (init = 1500)
    SP_0_operation_3_Scale_factor:          5039.54079 +/- 41.6397898 (0.83%) (init = 5000)
[[Correlations]] (unreported correlations are < 0.100)
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_3_Scale_factor)         =  0.566
    C(sys_0_site_0_isotropic_chemical_shift, mth_0_rotor_frequency)            = -0.449
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_3_Scale_factor)      =  0.288
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_3_Scale_factor)     =  0.125
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_1_Exponential_FWHM) = -0.114
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]
residuals = sf.residuals(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.plot(experiment, "k", linewidth=1, label="Experiment")
ax.plot(best_fit, "r", alpha=0.75, linewidth=1, label="Best Fit")
ax.plot(residuals, alpha=0.75, linewidth=1, label="Residuals")
ax.set_xlim(-1200, 600)
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 3.816 seconds)

12.2 2D Data Fitting

12.2.1 ^{13}C 2D MAT NMR of L-Histidine

The following is an illustration for fitting 2D MAT/PASS datasets. The example dataset is a ^{13}C 2D MAT spectrum of L-Histidine from Walder *et al.*[1]

```
import numpy as np
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator
from mrsimulator.methods import SSB2D
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
from mrsimulator.utils.collection import single_site_system_generator
```

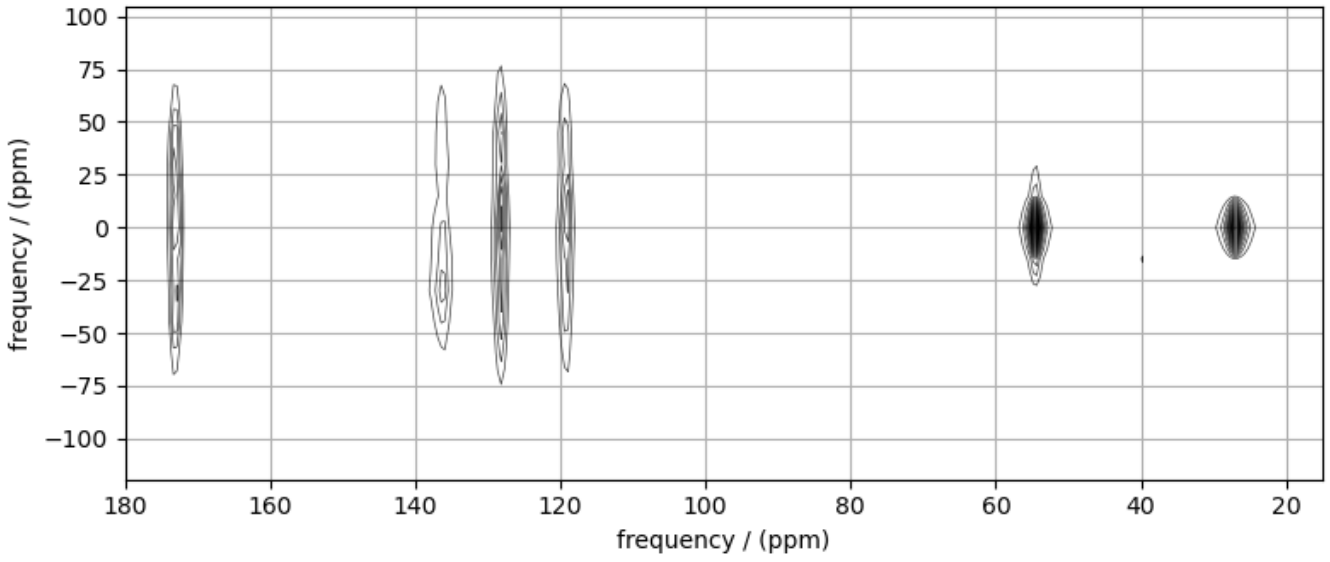
[1] B. J. Walder, K. K. Dey, D. C. Kaseman, J. H. Baltisberger, and P. J. Grandinetti, Sideband separation experiments in NMR with phase incremented echo train acquisition, J. Phys. Chem. 2013, **138**, 174203-1-12. DOI: 10.1063/1.4803142

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/814455/files/1H13C_CPPASS_LHistidine.csdf"
mat_data = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 0.4192854

# For the spectral fitting, we only focus on the real part of the complex dataset.
mat_data = mat_data.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in mat_data.dimensions]
```

When using the SSB2D method, ensure the horizontal dimension of the dataset is the isotropic dimension. Here, we apply an appropriate transpose operation to the dataset.

```
mat_data = mat_data.T  # transpose

# plot of the dataset.
max_amp = mat_data.max()
levels = (np.arange(24) + 1) * max_amp / 25  # contours are drawn at these levels.
options = dict(levels=levels, alpha=0.75, linewidths=0.5)  # plot options

plt.figure(figsize=(8, 3.5))
ax = plt.subplot(projection="csdm")
ax.contour(mat_data, colors="k", **options)
ax.set_xlim(180, 15)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Guess model

Create a guess list of spin systems.

```
shifts = [120, 128, 135, 175, 55, 25]  # in ppm
zeta = [-70, -65, -60, -60, -10, -10]  # in ppm
eta = [0.8, 0.4, 0.9, 0.3, 0.0, 0.0]

spin_systems = single_site_system_generator(
    isotopes="13C",
    isotropic_chemical_shifts=shifts,
    shielding_symmetric={"zeta": zeta, "eta": eta},
    abundance=100 / 6,
)
```

Method

Create the SSB2D method.

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(mat_data)

PASS = SSB2D(
    channels=["13C"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=1500,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=mat_data,  # add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = PASS.get_transition_pathways(sys)
```

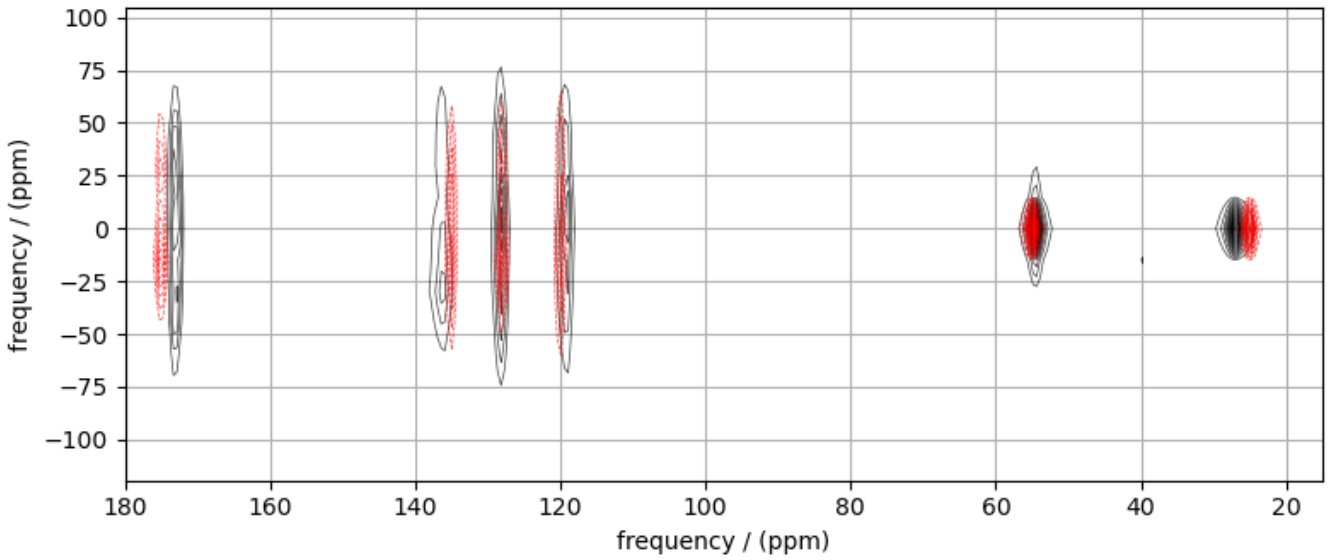
Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[PASS])
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        # Lorentzian convolution along the isotropic dimensions.
        sp.FFT(axis=0),
        sp.apodization.Exponential(FWHM="50 Hz"),
        sp.IFFT(axis=0),
        sp.Scale(factor=60),
    ]
)
```

(continues on next page)

(continued from previous page)

```
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(8, 3.5))
ax = plt.subplot(projection="csdm")
ax.contour(mat_data, colors="k", **options)
ax.contour(processed_data, colors="r", linestyles="--", **options)
ax.set_xlim(180, 15)
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Exponential_FWHM          50     -inf      inf     True     None
SP_0_operation_3_Scale_factor              60     -inf      inf     True     None
sys_0_abundance                         16.67        0      100     True     None
sys_0_site_0_isotropic_chemical_shift     120     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_eta      0.8        0        1     True     None
sys_0_site_0_shielding_symmetric_zeta      -70     -inf      inf     True     None
sys_1_abundance                         16.67        0      100     True     None
sys_1_site_0_isotropic_chemical_shift     128     -inf      inf     True     None
sys_1_site_0_shielding_symmetric_eta      0.4        0        1     True     None
```

(continues on next page)

(continued from previous page)

```
sys_1_site_0_shielding_symmetric_zeta       -65     -inf      inf     True     None
sys_2_abundance                          16.67        0      100     True     None
sys_2_site_0_isotropic_chemical_shift      135     -inf      inf     True     None
sys_2_site_0_shielding_symmetric_eta       0.9        0        1     True     None
sys_2_site_0_shielding_symmetric_zeta      -60     -inf      inf     True     None
sys_3_abundance                          16.67        0      100     True     None
sys_3_site_0_isotropic_chemical_shift      175     -inf      inf     True     None
sys_3_site_0_shielding_symmetric_eta       0.3        0        1     True     None
sys_3_site_0_shielding_symmetric_zeta      -60     -inf      inf     True     None
sys_4_abundance                          16.67        0      100     True     None
sys_4_site_0_isotropic_chemical_shift       55     -inf      inf     True     None
sys_4_site_0_shielding_symmetric_eta         0        0        1     True     None
sys_4_site_0_shielding_symmetric_zeta      -10     -inf      inf     True     None
sys_5_abundance                          16.67        0      100    False 100-sys_0_abundance-
↪sys_1_abundance-sys_2_abundance-sys_3_abundance-sys_4_abundance
sys_5_site_0_isotropic_chemical_shift       25     -inf      inf     True     None
sys_5_site_0_shielding_symmetric_eta         0        0        1     True     None
sys_5_site_0_shielding_symmetric_zeta      -10     -inf      inf     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 183
    # data points      = 32768
    # variables        = 25
    chi-square         = 54771.5124
    reduced chi-square = 1.67277013
    Akaike info crit   = 16883.5043
    Bayesian info crit = 17093.4345
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  119.168697 +/- 0.00372790 (0.00%) (init = 120)
    sys_0_site_0_shielding_symmetric_zeta: -72.1291854 +/- 0.32647311 (0.45%) (init = -70)
    sys_0_site_0_shielding_symmetric_eta:   0.98544438 +/- 0.00762029 (0.77%) (init = 0.8)
    sys_0_abundance:                        16.2139175 +/- 0.07758057 (0.48%) (init = 16.66667)
    sys_1_site_0_isotropic_chemical_shift:  128.195863 +/- 0.00312967 (0.00%) (init = 128)
    sys_1_site_0_shielding_symmetric_zeta: -75.6251701 +/- 0.27375960 (0.36%) (init = -65)
    sys_1_site_0_shielding_symmetric_eta:   0.94619261 +/- 0.00580552 (0.61%) (init = 0.4)
    sys_1_abundance:                        20.4664816 +/- 0.07809427 (0.38%) (init = 16.66667)
    sys_2_site_0_isotropic_chemical_shift:  136.194877 +/- 0.00476933 (0.00%) (init = 135)
    sys_2_site_0_shielding_symmetric_zeta: -86.3262707 +/- 0.38698036 (0.45%) (init = -60)
    sys_2_site_0_shielding_symmetric_eta:   0.42642729 +/- 0.00815945 (1.91%) (init = 0.9)
    sys_2_abundance:                        12.3222669 +/- 0.07828852 (0.64%) (init = 16.66667)
    sys_3_site_0_isotropic_chemical_shift:  172.997746 +/- 0.00307179 (0.00%) (init = 175)
    sys_3_site_0_shielding_symmetric_zeta: -69.1130708 +/- 0.25253811 (0.37%) (init = -60)
```

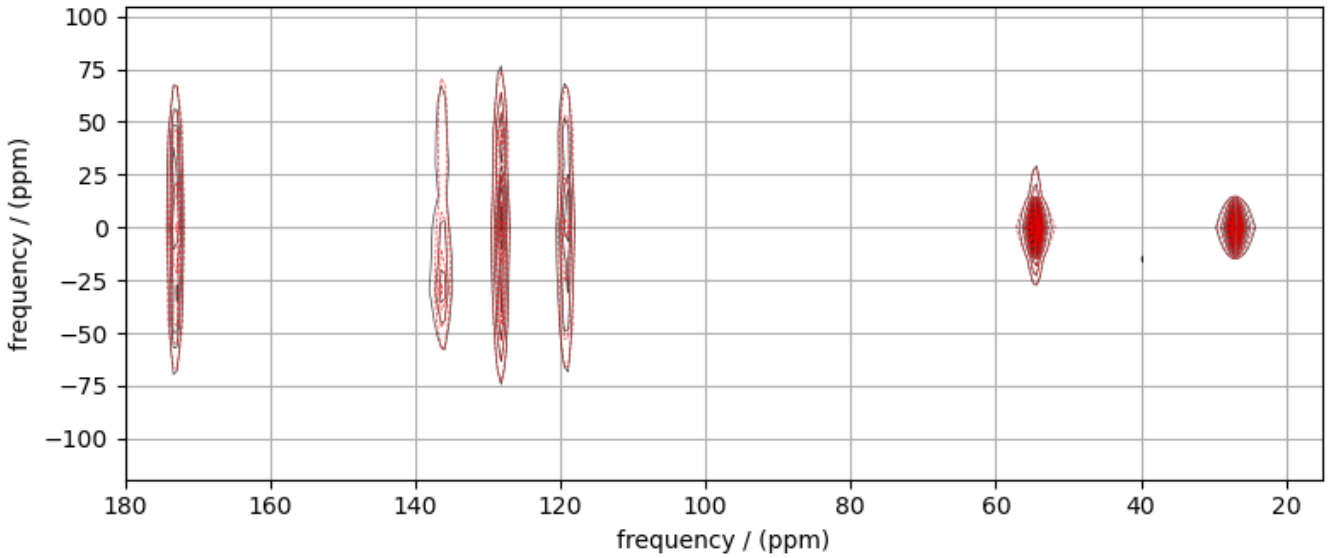
(continues on next page)

(continued from previous page)

```
    sys_3_site_0_shielding_symmetric_eta:   0.99996947 +/- 0.00633576 (0.63%) (init = 0.3)
    sys_3_abundance:                        19.3490339 +/- 0.07587813 (0.39%) (init = 16.66667)
    sys_4_site_0_isotropic_chemical_shift:  54.5170718 +/- 0.00145926 (0.00%) (init = 55)
    sys_4_site_0_shielding_symmetric_zeta: -20.0877990 +/- 0.12450012 (0.62%) (init = -10)
    sys_4_site_0_shielding_symmetric_eta:   0.16387265 +/- 0.13564086 (82.77%) (init = 0)
    sys_4_abundance:                        18.1300502 +/- 0.05430808 (0.30%) (init = 16.66667)
    sys_5_site_0_isotropic_chemical_shift:  26.9919519 +/- 0.00163183 (0.01%) (init = 25)
    sys_5_site_0_shielding_symmetric_zeta: -10.3631453 +/- 0.54821268 (5.29%) (init = -10)
    sys_5_site_0_shielding_symmetric_eta:   0.84480483 +/- 0.21592244 (25.56%) (init = 0)
    sys_5_abundance:                        13.5182498 +/- 0.05061883 (0.37%) == '100-sys_0_
↪abundance-sys_1_abundance-sys_2_abundance-sys_3_abundance-sys_4_abundance'
    SP_0_operation_1_Exponential_FWHM:      99.1258201 +/- 0.31273906 (0.32%) (init = 50)
    SP_0_operation_3_Scale_factor:          101.312086 +/- 0.22835689 (0.23%) (init = 60)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_5_site_0_shielding_symmetric_zeta, sys_5_site_0_shielding_symmetric_eta) =  0.940
    C(sys_4_site_0_shielding_symmetric_zeta, sys_4_site_0_shielding_symmetric_eta) =  0.682
    C(SP_0_operation_1_Exponential_FWHM, SP_0_operation_3_Scale_factor)            =  0.562
    C(sys_1_site_0_shielding_symmetric_zeta, sys_1_site_0_shielding_symmetric_eta) =  0.438
    C(sys_3_site_0_shielding_symmetric_zeta, sys_3_site_0_shielding_symmetric_eta) =  0.434
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta) =  0.430
    C(sys_2_site_0_shielding_symmetric_zeta, sys_2_site_0_shielding_symmetric_eta) =  0.340
    C(sys_4_site_0_shielding_symmetric_zeta, sys_4_abundance)                      = -0.291
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_abundance)                      = -0.291
    C(sys_3_site_0_shielding_symmetric_zeta, sys_3_abundance)                      = -0.285
    C(sys_4_abundance, SP_0_operation_3_Scale_factor)                              = -0.279
    C(sys_0_abundance, sys_1_abundance)                                    = -0.274
    C(sys_1_site_0_shielding_symmetric_zeta, sys_1_abundance)                      = -0.270
    C(sys_1_abundance, sys_2_abundance)                                    = -0.270
    C(sys_1_abundance, sys_3_abundance)                                    = -0.263
    C(sys_0_abundance, sys_3_abundance)                                    = -0.257
    C(sys_2_abundance, sys_3_abundance)                                    = -0.247
    C(sys_0_abundance, sys_2_abundance)                                    = -0.232
    C(sys_2_site_0_shielding_symmetric_eta, sys_2_abundance)                       =  0.223
    C(sys_2_site_0_shielding_symmetric_zeta, sys_2_abundance)                      = -0.218
    C(sys_2_abundance, sys_4_abundance)                                    = -0.207
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_abundance)                      =  0.200
    C(sys_4_site_0_shielding_symmetric_eta, sys_4_abundance)                       =  0.191
    C(sys_0_abundance, sys_4_abundance)                                    = -0.183
    C(sys_4_site_0_isotropic_chemical_shift, SP_0_operation_1_Exponential_FWHM)    = -0.163
    C(sys_2_abundance, SP_0_operation_3_Scale_factor)                              =  0.162
    C(sys_1_site_0_isotropic_chemical_shift, SP_0_operation_1_Exponential_FWHM)    = -0.160
    C(sys_3_abundance, sys_4_abundance)                                    = -0.155
    C(sys_1_abundance, sys_4_abundance)                                    = -0.152
    C(sys_3_site_0_shielding_symmetric_zeta, SP_0_operation_3_Scale_factor)        = -0.119
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_3_Scale_factor)        = -0.117
    C(sys_1_site_0_shielding_symmetric_zeta, SP_0_operation_3_Scale_factor)        = -0.114
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]

# Plot of the best fit solution
plt.figure(figsize=(8, 3.5))
ax = plt.subplot(projection="csdm")
ax.contour(mat_data, colors="k", **options)
ax.contour(best_fit, colors="r", linestyles="--", **options)
ax.set_xlim(180, 15)
plt.grid()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 22.131 seconds)

12.2.2 ^{87}Rb 2D QMAT NMR of Rb_2SO_4

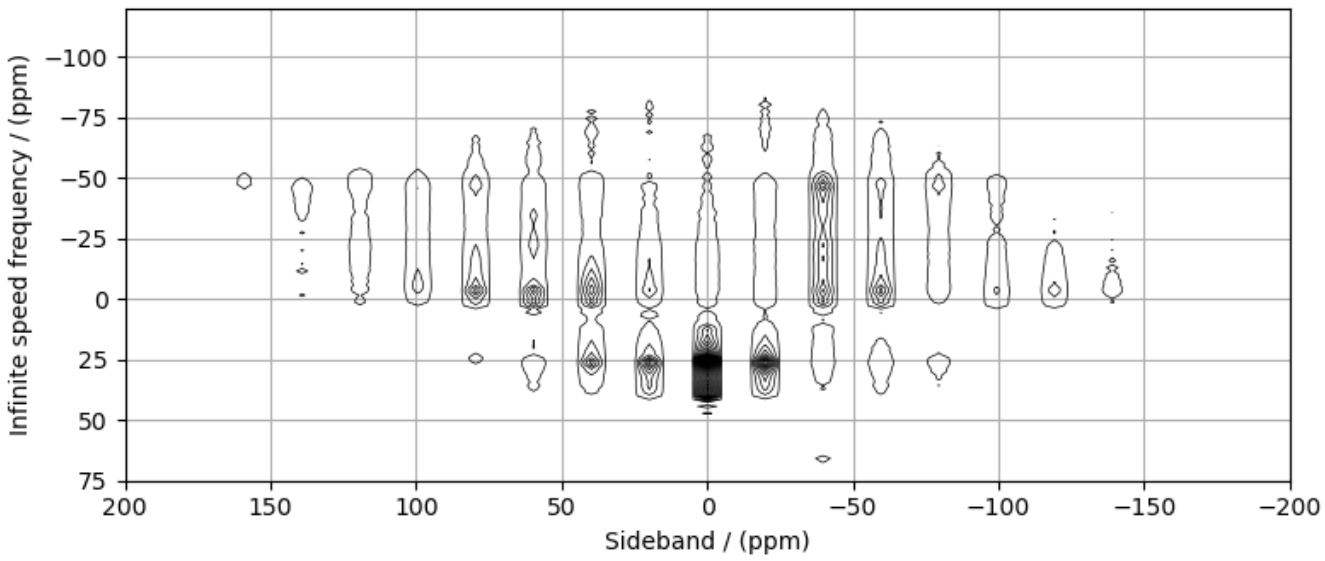
The following is an illustration for fitting 2D QMAT/QPASS datasets. The example dataset is a ^{87}Rb 2D QMAT spectrum of Rb_2SO_4 from Walder *et al.*[1]

```
import numpy as np
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, SpinSystem, Site
from mrsimulator.methods import SSB2D
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

[1] B. J. Walder, K. K. Dey, D. C. Kaseman, J. H. Baltisberger, and P. J. Grandinetti, Sideband separation experiments in NMR with phase incremented echo train acquisition, J. Phys. Chem. 2013, **138**, 174203-1-12. DOI: 10.1063/1.4803142

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/834704/files/Rb2SO4_QMAT.csdf"
qmat_data = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 6.530634

# For the spectral fitting, we only focus on the real part of the complex dataset.
qmat_data = qmat_data.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in qmat_data.dimensions]

# plot of the dataset.
max_amp = qmat_data.max()
levels = (np.arange(31) + 0.15) * max_amp / 32  # contours are drawn at these levels.
options = dict(levels=levels, alpha=1, linewidths=0.5)  # plot options

plt.figure(figsize=(8, 3.5))
ax = plt.subplot(projection="csdm")
ax.contour(qmat_data.T, colors="k", **options)
ax.set_xlim(200, -200)
ax.set_ylim(75, -120)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Guess model

Create a guess list of spin systems.

```
Rb_1 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=16,  # in ppm
    quadrupolar={"Cq": 5.5e6, "eta": 0.1},  # Cq in Hz
)
Rb_2 = Site(
    isotope="87Rb",
    isotropic_chemical_shift=40,  # in ppm
    quadrupolar={"Cq": 2.1e6, "eta": 0.95},  # Cq in Hz
)

spin_systems = [SpinSystem(sites=[s]) for s in [Rb_1, Rb_2]]
```

Method

Create the SSB2D method.

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(qmat_data)

PASS = SSB2D(
    channels=["87Rb"],
    magnetic_flux_density=9.395,  # in T
    rotor_frequency=2604,  # in Hz
    spectral_dimensions=spectral_dims,
    experiment=qmat_data,  # add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = PASS.get_transition_pathways(sys)
```

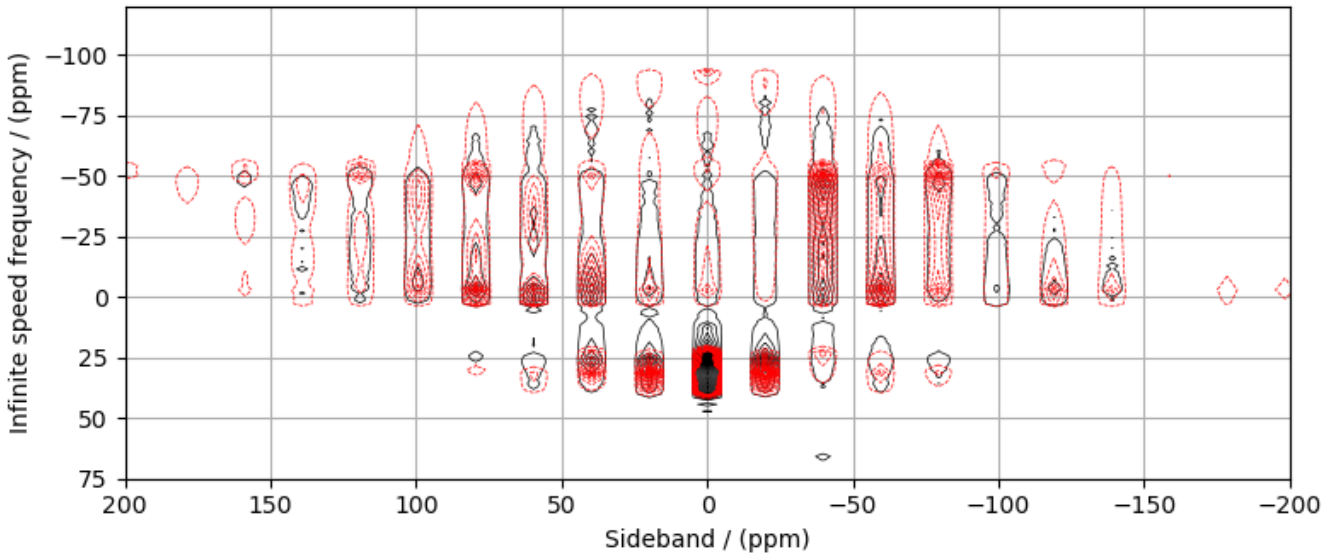
Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[PASS])
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        # Lorentzian convolution along the isotropic dimensions.
        sp.FFT(axis=0),
        sp.apodization.Gaussian(FWHM="50 Hz"),
        sp.IFFT(axis=0),
        sp.Scale(factor=1e4),
```

(continues on next page)

(continued from previous page)

```
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(8, 3.5))
ax = plt.subplot(projection="csdm")
ax.contour(qmat_data.T, colors="k", **options)
ax.contour(processed_data.T, colors="r", linestyles="--", **options)
ax.set_xlim(200, -200)
ax.set_ylim(75, -120)
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM             50     -inf      inf     True     None
SP_0_operation_3_Scale_factor           1e+04     -inf      inf     True     None
sys_0_abundance                            50        0      100     True     None
sys_0_site_0_isotropic_chemical_shift      16     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq           5.5e+06     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta              0.1        0        1     True     None
```

(continues on next page)

(continued from previous page)

```
sys_1_abundance                              50        0      100    False 100-sys_0_abundance
sys_1_site_0_isotropic_chemical_shift        40     -inf      inf     True     None
sys_1_site_0_quadrupolar_Cq             2.1e+06     -inf      inf     True     None
sys_1_site_0_quadrupolar_eta               0.95        0        1     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 133
    # data points      = 65536
    # variables        = 9
    chi-square         = 217831.432
    reduced chi-square = 3.32430040
    Akaike info crit   = 78734.7253
    Bayesian info crit = 78816.5385
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  13.8247093 +/- 0.01883732 (0.14%) (init = 16)
    sys_0_site_0_quadrupolar_Cq:            5229369.47 +/- 989.315964 (0.02%) (init = 5500000)
    sys_0_site_0_quadrupolar_eta:           0.12875964 +/- 3.8584e-04 (0.30%) (init = 0.1)
    sys_0_abundance:                        55.8944405 +/- 0.05573973 (0.10%) (init = 50)
    sys_1_site_0_isotropic_chemical_shift:  40.4488902 +/- 0.00658995 (0.02%) (init = 40)
    sys_1_site_0_quadrupolar_Cq:            2697016.78 +/- 996.758989 (0.04%) (init = 2100000)
    sys_1_site_0_quadrupolar_eta:           0.86990373 +/- 0.00143241 (0.16%) (init = 0.95)
    sys_1_abundance:                        44.1055595 +/- 0.05573973 (0.13%) == '100-sys_0_
↪abundance'
    SP_0_operation_1_Gaussian_FWHM:        -142.754756 +/- 2.83847927 (1.99%) (init = 50)
    SP_0_operation_3_Scale_factor:          6277.07909 +/- 7.75261506 (0.12%) (init = 10000)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_Cq)    =  0.827
    C(sys_1_site_0_quadrupolar_Cq, sys_1_site_0_quadrupolar_eta)             = -0.818
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_Cq)    =  0.633
    C(sys_0_abundance, SP_0_operation_3_Scale_factor)                        =  0.630
    C(sys_1_site_0_quadrupolar_eta, SP_0_operation_1_Gaussian_FWHM)          = -0.356
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_eta)   = -0.263
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_3_Scale_factor)           =  0.240
    C(sys_0_site_0_quadrupolar_eta, sys_0_abundance)                         =  0.235
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_eta)   = -0.198
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_3_Scale_factor)         = -0.197
    C(sys_0_site_0_quadrupolar_Cq, sys_0_site_0_quadrupolar_eta)             = -0.191
    C(sys_0_abundance, sys_1_site_0_isotropic_chemical_shift)                = -0.178
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_3_Scale_factor)            =  0.164
    C(sys_1_site_0_isotropic_chemical_shift, SP_0_operation_3_Scale_factor)  =  0.162
    C(sys_0_abundance, sys_1_site_0_quadrupolar_Cq)                          = -0.157
    C(sys_1_site_0_quadrupolar_Cq, SP_0_operation_1_Gaussian_FWHM)           =  0.156
```

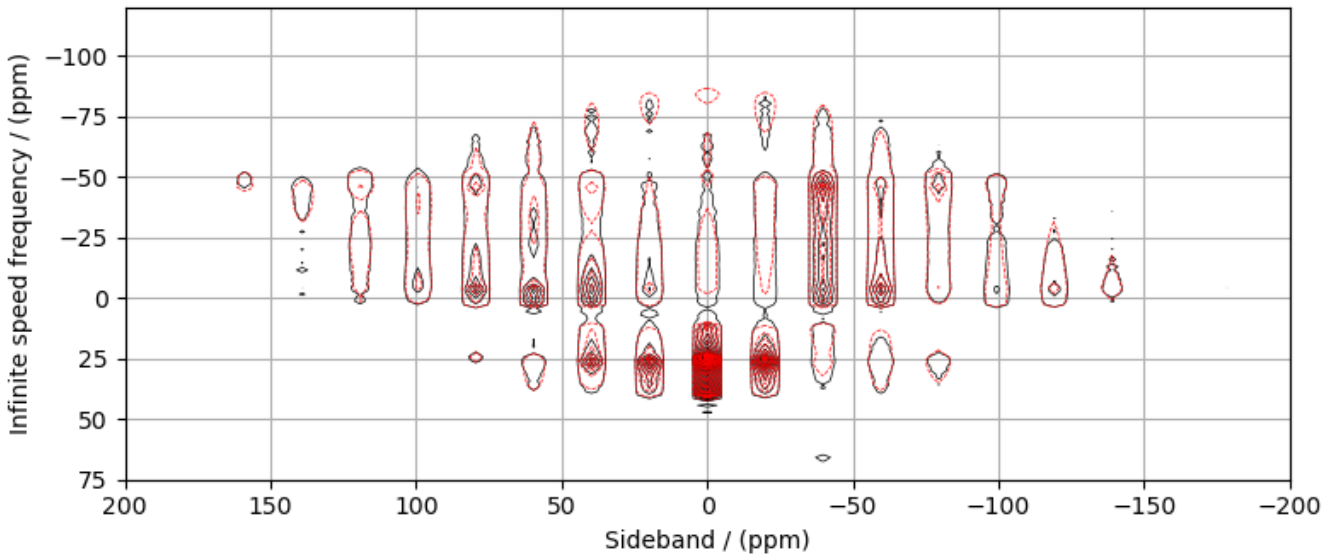
(continues on next page)

(continued from previous page)

```
    C(sys_0_site_0_quadrupolar_Cq, sys_0_abundance)                          =  0.138
    C(sys_1_site_0_isotropic_chemical_shift, SP_0_operation_1_Gaussian_FWHM) = -0.124
    C(sys_1_site_0_quadrupolar_Cq, SP_0_operation_3_Scale_factor)            =  0.104
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_3_Scale_factor)  =  0.104
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]

# Plot of the best fit solution
plt.figure(figsize=(8, 3.5))
ax = plt.subplot(projection="csdm")
ax.contour(qmat_data.T, colors="k", **options)
ax.contour(best_fit.T, colors="r", linestyles="--", **options)
ax.set_xlim(200, -200)
ax.set_ylim(75, -120)
plt.grid()
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 12.516 seconds)

12.2.3 ^{17}O 2D DAS NMR of Coesite

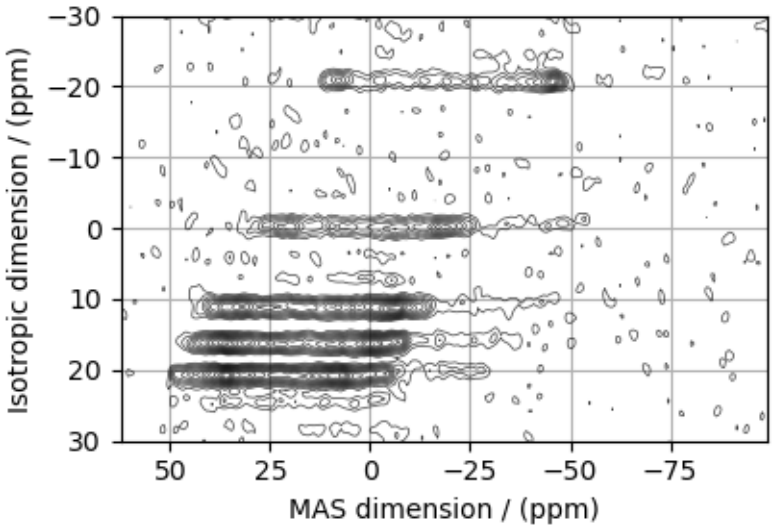
Coesite is a high-pressure (2-3 GPa) and high-temperature (700°C) polymorph of silicon dioxide SiO_2. Coesite has five crystallographic ^{17}O sites. The experimental dataset used in this example is published in Grandinetti *et al.*[1]

```
import numpy as np
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
from mrsimulator.utils.collection import single_site_system_generator
```

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/814455/files/DASCoesite.csdf"
experiment = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 921.6698

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
max_amp = experiment.max()
levels = (np.arange(14) + 1) * max_amp / 15  # contours are drawn at these levels.
options = dict(levels=levels, alpha=0.75, linewidths=0.5)  # plot options

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.invert_xaxis()
ax.set_ylim(30, -30)
plt.grid()
plt.tight_layout()
plt.show()
```

[1] Grandinetti, P. J., Baltisberger, J. H., Farnan, I., Stebbins, J. F., Werner, U. and Pines, A. Solid-State ^{17}O Magic-Angle and Dynamic-Angle Spinning NMR Study of the SiO_2 Polymorph Coesite, J. Phys. Chem. 1995, **99**, *32*, 12341-12348. DOI: 10.1021/j100032a045



Create a fitting model

Guess model

Create a guess list of spin systems.

```
shifts = [29, 39, 54.8, 51, 56]  # in ppm
Cq = [6.1e6, 5.4e6, 5.5e6, 5.5e6, 5.1e6]  # in  Hz
eta = [0.1, 0.2, 0.15, 0.15, 0.3]
abundance_ratio = [1, 1, 2, 2, 2]
abundance = np.asarray(abundance_ratio) / 8 * 100  # in %

spin_systems = single_site_system_generator(
    isotopes="17O",
    isotropic_chemical_shifts=shifts,
    quadrupolar={"Cq": Cq, "eta": eta},
    abundance=abundance,
)
```

Method

Create the DAS method.

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

DAS = Method2D(
    channels=["17O"],
    magnetic_flux_density=11.744,  # in T
    spectral_dimensions=[
        {
            **spectral_dims[0],
            "events": [
                {
                    "fraction": 0.5,
```

(continues on next page)

(continued from previous page)

```
                    "rotor_angle": 37.38 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                },
                {
                    "fraction": 0.5,
                    "rotor_angle": 79.19 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                },
            ],
        },
        # The last spectral dimension block is the direct-dimension
        {
            **spectral_dims[1],
            "events": [
                {
                    "rotor_angle": 54.735 * 3.14159 / 180,
                    "transition_query": {"P": [-1], "D": [0]},
                }
            ],
        },
    ],
    experiment=experiment,  # also add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the das method.
for sys in spin_systems:
    sys.transition_pathways = DAS.get_transition_pathways(sys)
```

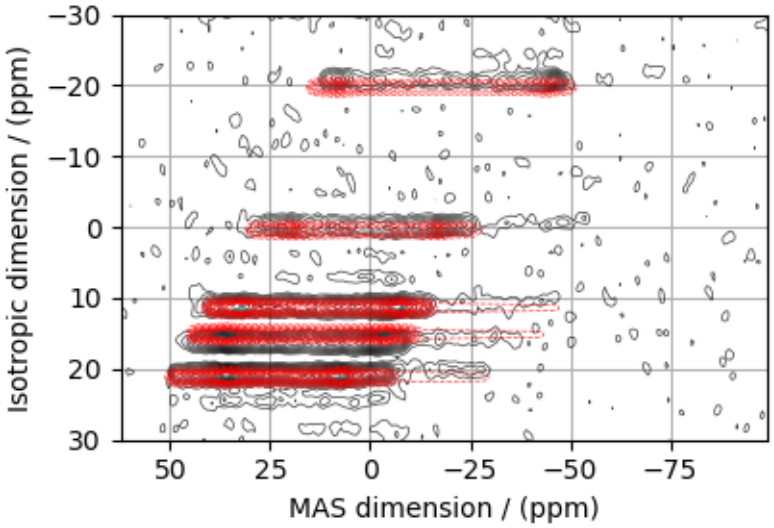
Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[DAS])
sim.config.number_of_sidebands = 1  # no sidebands are required for this dataset.
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.15 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.1 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
        sp.Scale(factor=4e7),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
```

(continues on next page)

(continued from previous page)

```
# -------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(processed_data, colors="r", linestyles="--", **options)
ax.invert_xaxis()
ax.set_ylim(30, -30)
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM           0.15     -inf      inf     True     None
SP_0_operation_2_Gaussian_FWHM            0.1     -inf      inf     True     None
SP_0_operation_4_Scale_factor            4e+07     -inf      inf     True     None
sys_0_abundance                           12.5        0      100     True     None
sys_0_site_0_isotropic_chemical_shift       29     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq            6.1e+06     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta              0.1        0        1     True     None
sys_1_abundance                           12.5        0      100     True     None
sys_1_site_0_isotropic_chemical_shift       39     -inf      inf     True     None
sys_1_site_0_quadrupolar_Cq            5.4e+06     -inf      inf     True     None
sys_1_site_0_quadrupolar_eta              0.2        0        1     True     None
```

(continues on next page)

(continued from previous page)

```
sys_2_abundance                              25        0      100     True     None
sys_2_site_0_isotropic_chemical_shift      54.8     -inf      inf     True     None
sys_2_site_0_quadrupolar_Cq             5.5e+06     -inf      inf     True     None
sys_2_site_0_quadrupolar_eta               0.15        0        1     True     None
sys_3_abundance                              25        0      100     True     None
sys_3_site_0_isotropic_chemical_shift        51     -inf      inf     True     None
sys_3_site_0_quadrupolar_Cq             5.5e+06     -inf      inf     True     None
sys_3_site_0_quadrupolar_eta               0.15        0        1     True     None
sys_4_abundance                              25        0      100    False 100-sys_0_abundance-
↪sys_1_abundance-sys_2_abundance-sys_3_abundance
sys_4_site_0_isotropic_chemical_shift        56     -inf      inf     True     None
sys_4_site_0_quadrupolar_Cq             5.1e+06     -inf      inf     True     None
sys_4_site_0_quadrupolar_eta                0.3        0        1     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize(method="powell")
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = Powell
    # function evals   = 2194
    # data points      = 131072
    # variables        = 22
    chi-square         = 186495.645
    reduced chi-square = 1.42308771
    Akaike info crit   = 46267.9953
    Bayesian info crit = 46483.2324
##  Warning: uncertainties could not be estimated:
    this fitting method does not natively calculate uncertainties
    and numdifftools is not installed for lmfit to do this. Use
    `pip install numdifftools` for lmfit to estimate uncertainties
    with this fitting method.
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  27.8024407 (init = 29)
    sys_0_site_0_quadrupolar_Cq:            6099675.46 (init = 6100000)
    sys_0_site_0_quadrupolar_eta:           0.01078074 (init = 0.1)
    sys_0_abundance:                        11.1052088 (init = 12.5)
    sys_1_site_0_isotropic_chemical_shift:  38.5408329 (init = 39)
    sys_1_site_0_quadrupolar_Cq:            5399536.41 (init = 5400000)
    sys_1_site_0_quadrupolar_eta:           0.20001097 (init = 0.2)
    sys_1_abundance:                        13.8849100 (init = 12.5)
    sys_2_site_0_isotropic_chemical_shift:  55.9052118 (init = 54.8)
    sys_2_site_0_quadrupolar_Cq:            5493792.38 (init = 5500000)
    sys_2_site_0_quadrupolar_eta:           0.17362890 (init = 0.15)
    sys_2_abundance:                        24.8249011 (init = 25)
    sys_3_site_0_isotropic_chemical_shift:  51.2660231 (init = 51)
    sys_3_site_0_quadrupolar_Cq:            5500002.73 (init = 5500000)
```

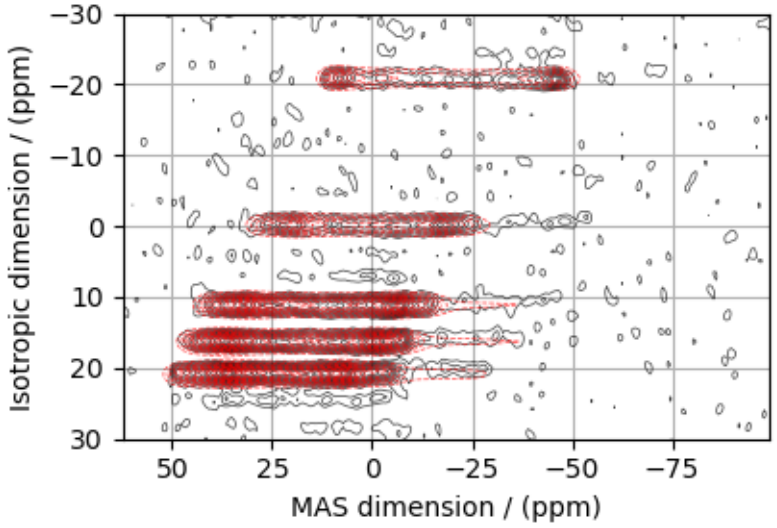
(continues on next page)

(continued from previous page)

```
    sys_3_site_0_quadrupolar_eta:            0.20827675 (init = 0.15)
    sys_3_abundance:                         23.6462498 (init = 25)
    sys_4_site_0_isotropic_chemical_shift:   55.6908554 (init = 56)
    sys_4_site_0_quadrupolar_Cq:             5099895.27 (init = 5100000)
    sys_4_site_0_quadrupolar_eta:            0.29999967 (init = 0.3)
    sys_4_abundance:                         26.5387303 == '100-sys_0_abundance-sys_1_abundance-
↪sys_2_abundance-sys_3_abundance'
    SP_0_operation_1_Gaussian_FWHM:          -0.34003342 (init = 0.15)
    SP_0_operation_2_Gaussian_FWHM:          0.13846642 (init = 0.1)
    SP_0_operation_4_Scale_factor:           42620049.0 (init = 4e+07)
```

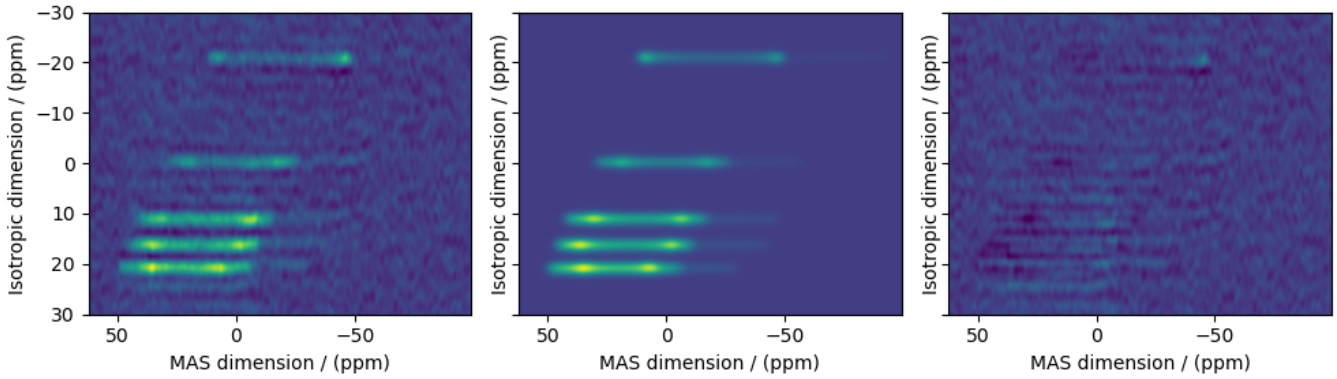
The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(best_fit, colors="r", linestyles="--", **options)
ax.invert_xaxis()
ax.set_ylim(30, -30)
plt.grid()
plt.tight_layout()
plt.show()
```



The best fit solution

```
residuals = sf.residuals(sim, processor)[0]

fig, ax = plt.subplots(
    1, 3, sharey=True, figsize=(10, 3.0), subplot_kw={"projection": "csdm"}
)
vmax, vmin = experiment.max(), experiment.min()
for i, dat in enumerate([experiment, best_fit, residuals]):
    ax[i].imshow(dat, aspect="auto", vmax=vmax, vmin=vmin)
    ax[i].invert_xaxis()
ax[0].set_ylim(30, -30)
plt.tight_layout()
plt.show()
```



Total running time of the script: (1 minutes 23.021 seconds)

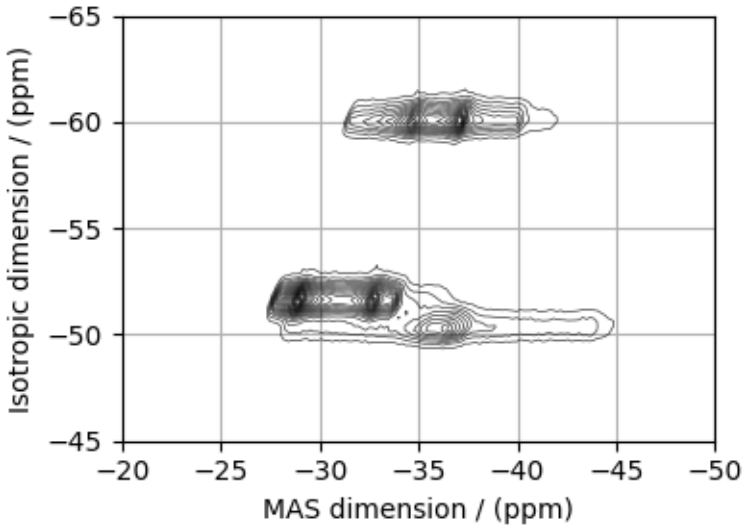
12.2.4 ^{87}Rb 2D 3QMAS NMR of $RbNO_3$

The following is a 3QMAS fitting example for $RbNO_3$. The dataset was acquired and shared by Brendan Wilson.

```
import numpy as np
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator
from mrsimulator.methods import ThreeQ_VAS
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
from mrsimulator.utils.collection import single_site_system_generator
```

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/814455/files/RbNO3_MQMAS.csdf"
experiment = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 175.5476

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
max_amp = experiment.max()
levels = (np.arange(24) + 1) * max_amp / 25  # contours are drawn at these levels.
options = dict(levels=levels, alpha=0.75, linewidths=0.5)  # plot options

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.set_xlim(-20, -50)
ax.set_ylim(-45, -65)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Guess model

Create a guess list of spin systems.

```
shifts = [-26.4, -28.5, -31.3]  # in ppm
Cq = [1.7e6, 2.0e6, 1.7e6]  # in  Hz
eta = [0.2, 1.0, 0.6]
abundance = [33.33, 33.33, 33.33]  # in %

spin_systems = single_site_system_generator(
    isotopes="87Rb",
    isotropic_chemical_shifts=shifts,
    quadrupolar={"Cq": Cq, "eta": eta},
    abundance=abundance,
)
```

Method

Create the 3QMAS method.

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

MQMAS = ThreeQ_VAS(
    channels=["87Rb"],
    magnetic_flux_density=9.395,  # in T
    spectral_dimensions=spectral_dims,
    experiment=experiment,  # add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the das method.
for sys in spin_systems:
    sys.transition_pathways = MQMAS.get_transition_pathways(sys)
```

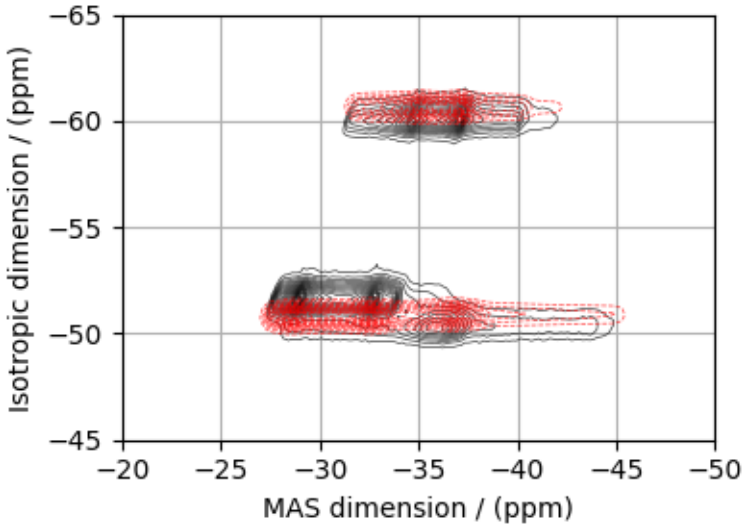
Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[MQMAS])
sim.config.number_of_sidebands = 1
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="0.08 kHz", dim_index=0),
        sp.apodization.Gaussian(FWHM="0.1 kHz", dim_index=1),
        sp.FFT(dim_index=(0, 1)),
        sp.Scale(factor=1e7),
```

(continues on next page)

(continued from previous page)

```
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(processed_data, colors="r", linestyles="--", **options)
ax.set_xlim(-20, -50)
ax.set_ylim(-45, -65)
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM           0.08     -inf      inf     True     None
SP_0_operation_2_Gaussian_FWHM            0.1     -inf      inf     True     None
SP_0_operation_4_Scale_factor            1e+07     -inf      inf     True     None
sys_0_abundance                          33.33        0      100     True     None
sys_0_site_0_isotropic_chemical_shift    -26.4     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq            1.7e+06     -inf      inf     True     None
```

(continues on next page)

(continued from previous page)

```
sys_0_site_0_quadrupolar_eta                0.2        0        1     True     None
sys_1_abundance                           33.33        0      100     True     None
sys_1_site_0_isotropic_chemical_shift     -28.5     -inf      inf     True     None
sys_1_site_0_quadrupolar_Cq               2e+06     -inf      inf     True     None
sys_1_site_0_quadrupolar_eta                  1        0        1     True     None
sys_2_abundance                           33.33        0      100    False 100-sys_0_abundance-
↪sys_1_abundance
sys_2_site_0_isotropic_chemical_shift     -31.3     -inf      inf     True     None
sys_2_site_0_quadrupolar_Cq             1.7e+06     -inf      inf     True     None
sys_2_site_0_quadrupolar_eta                0.6        0        1     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 409
    # data points      = 262144
    # variables        = 14
    chi-square         = 514378.952
    reduced chi-square = 1.96230478
    Akaike info crit   = 176730.432
    Bayesian info crit = 176877.105
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift: -26.7600498 +/- 3.3639e-04 (0.00%) (init = -26.4)
    sys_0_site_0_quadrupolar_Cq:            1693623.74 +/- 115.012581 (0.01%) (init = 1700000)
    sys_0_site_0_quadrupolar_eta:           0.21817910 +/- 3.1396e-04 (0.14%) (init = 0.2)
    sys_0_abundance:                        41.9395333 +/- 0.02257962 (0.05%) (init = 33.33333)
    sys_1_site_0_isotropic_chemical_shift: -28.1157726 +/- 9.7428e-04 (0.00%) (init = -28.5)
    sys_1_site_0_quadrupolar_Cq:            2027908.41 +/- 348.654106 (0.02%) (init = 2000000)
    sys_1_site_0_quadrupolar_eta:           0.90003177 +/- 7.1639e-04 (0.08%) (init = 1)
    sys_1_abundance:                        24.2721929 +/- 0.02577385 (0.11%) (init = 33.33333)
    sys_2_site_0_isotropic_chemical_shift: -31.0314607 +/- 4.1538e-04 (0.00%) (init = -31.3)
    sys_2_site_0_quadrupolar_Cq:            1721336.41 +/- 151.036797 (0.01%) (init = 1700000)
    sys_2_site_0_quadrupolar_eta:           0.56032125 +/- 4.5876e-04 (0.08%) (init = 0.6)
    sys_2_abundance:                        33.7882738 +/- 0.02119368 (0.06%) == '100-sys_0_
↪abundance-sys_1_abundance'
    SP_0_operation_1_Gaussian_FWHM:         0.06201469 +/- 2.8251e-04 (0.46%) (init = 0.08)
    SP_0_operation_2_Gaussian_FWHM:         0.15110281 +/- 9.3281e-05 (0.06%) (init = 0.1)
    SP_0_operation_4_Scale_factor:          13385739.8 +/- 6957.71769 (0.05%) (init = 1e+07)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_1_site_0_quadrupolar_Cq, sys_1_site_0_quadrupolar_eta)              = -0.805
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_Cq)     = -0.795
    C(sys_2_site_0_quadrupolar_Cq, sys_2_site_0_quadrupolar_eta)              = -0.646
    C(sys_0_abundance, sys_1_abundance)                                 = -0.623
    C(SP_0_operation_2_Gaussian_FWHM, SP_0_operation_4_Scale_factor)          =  0.452
```

(continues on next page)

(continued from previous page)

```
    C(sys_2_site_0_isotropic_chemical_shift, sys_2_site_0_quadrupolar_Cq)    = -0.415
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_eta)   = -0.293
    C(sys_0_site_0_quadrupolar_Cq, sys_0_site_0_quadrupolar_eta)             = -0.284
    C(sys_1_site_0_isotropic_chemical_shift, sys_1_site_0_quadrupolar_Cq)    = -0.276
    C(sys_0_abundance, SP_0_operation_4_Scale_factor)                        = -0.274
    C(sys_2_site_0_isotropic_chemical_shift, sys_2_site_0_quadrupolar_eta)   = -0.272
    C(sys_1_abundance, SP_0_operation_4_Scale_factor)                        =  0.245
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_1_Gaussian_FWHM)          = -0.238
    C(sys_1_site_0_quadrupolar_eta, sys_1_abundance)                         = -0.228
    C(sys_1_site_0_quadrupolar_Cq, sys_1_abundance)                          =  0.228
    C(sys_0_abundance, sys_1_site_0_quadrupolar_eta)                         =  0.176
    C(sys_0_abundance, sys_1_site_0_quadrupolar_Cq)                          = -0.160
    C(sys_1_abundance, SP_0_operation_2_Gaussian_FWHM)                       = -0.145
    C(sys_2_site_0_quadrupolar_Cq, SP_0_operation_1_Gaussian_FWHM)           = -0.130
    C(sys_0_site_0_quadrupolar_eta, sys_0_abundance)                         =  0.117
    C(sys_1_site_0_quadrupolar_eta, SP_0_operation_2_Gaussian_FWHM)          =  0.114
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_4_Scale_factor)         =  0.107
    C(sys_2_site_0_isotropic_chemical_shift, SP_0_operation_1_Gaussian_FWHM) =  0.106
    C(sys_1_site_0_quadrupolar_Cq, SP_0_operation_4_Scale_factor)            =  0.102
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(best_fit, colors="r", linestyles="--", **options)
ax.set_xlim(-20, -50)
ax.set_ylim(-45, -65)
plt.grid()
plt.tight_layout()
plt.show()
```

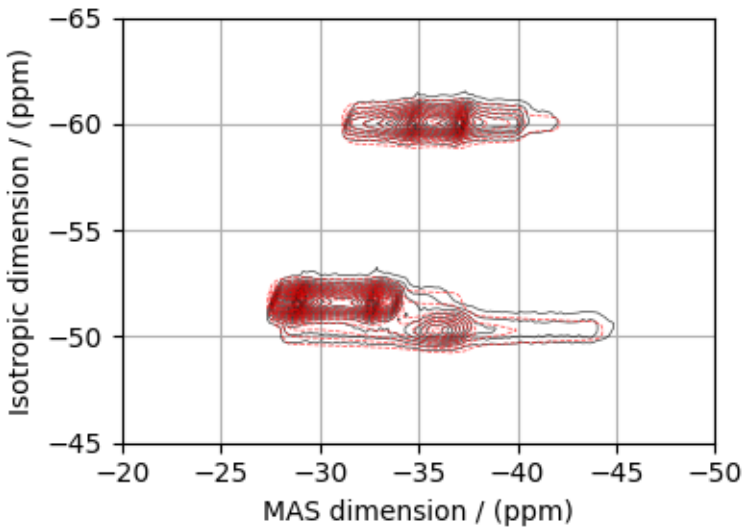
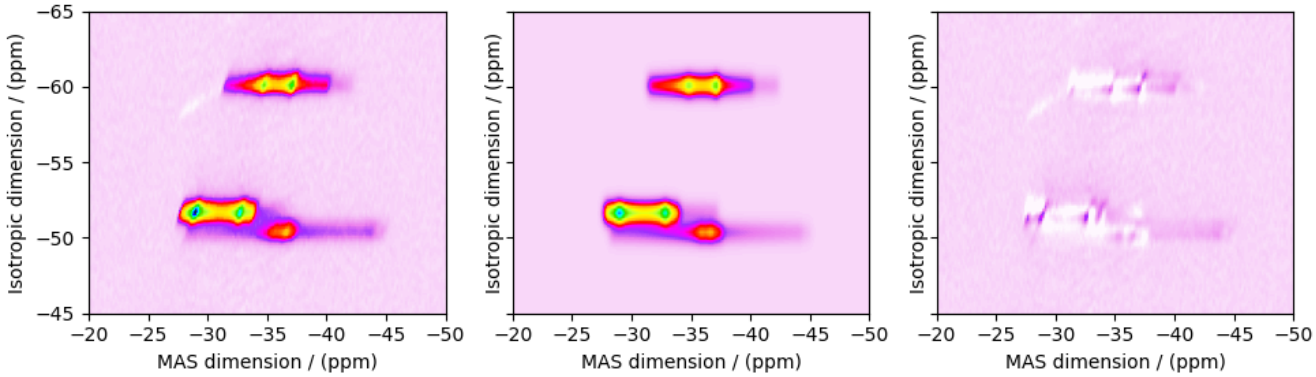


Image plots with residuals

```
residuals = sf.residuals(sim, processor)[0]

fig, ax = plt.subplots(
    1, 3, sharey=True, figsize=(10, 3.0), subplot_kw={"projection": "csdm"}
)
vmax, vmin = experiment.max(), experiment.min()
for i, dat in enumerate([experiment, best_fit, residuals]):
    ax[i].imshow(dat, aspect="auto", cmap="gist_ncar_r", vmax=vmax, vmin=vmin)
    ax[i].set_xlim(-20, -50)
ax[0].set_ylim(-45, -65)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 23.686 seconds)

12.2.5 $NiCl_2.2D_2O$, ^2H (I=1) Shifting-d echo

^2H (I=1) 2D NMR CSA-Quad 1st order correlation spectrum.

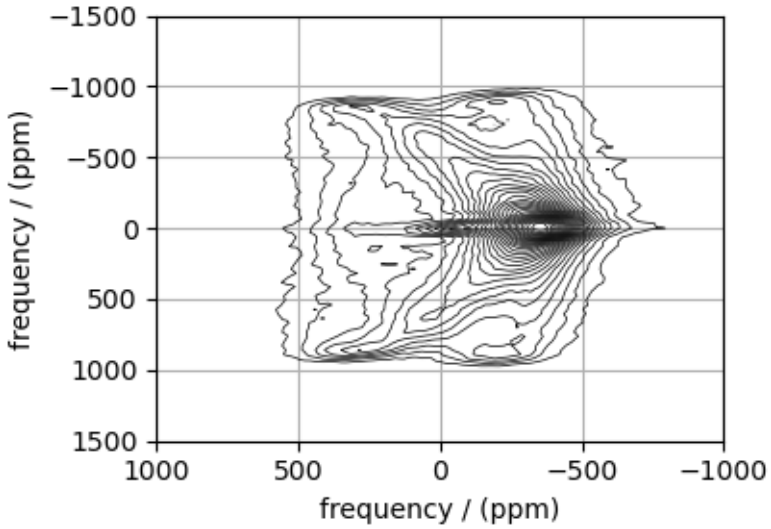
The following is an example of fitting static shifting-d echo NMR correlation spectrum of $NiCl_2 \cdot 2D_2O$ crystalline solid. The spectrum used here is from Walder *et al.*[1].

```
import numpy as np
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, Site, SpinSystem
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/830903/files/NiCl2.2D2O.csdf"
experiment = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 7.500

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
max_amp = experiment.max()
levels = (np.arange(29) + 1) * max_amp / 30  # contours are drawn at these levels.
options = dict(levels=levels, linewidths=0.5)  # plot options

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.set_xlim(1000, -1000)
ax.set_ylim(1500, -1500)
plt.grid()
plt.tight_layout()
plt.show()
```

[1] Walder B.J, Patterson A.M., Baltisberger J.H, and Grandinetti P.J Hydrogen motional disorder in crystalline iron group chloride dihydrates spectroscopy, J. Chem. Phys. (2018) **149**, 084503. DOI: 10.1063/1.5037151



Create a fitting model

Guess model

Create a guess list of spin systems.

```
site = Site(
    isotope="2H",
    isotropic_chemical_shift=-90,  # in ppm
    shielding_symmetric={
        "zeta": -610,  # in ppm
        "eta": 0.15,
        "alpha": 0.7,  # in rads
        "beta": 2.0,  # in rads
        "gamma": 3.0,  # in rads
    },
    quadrupolar={"Cq": 75.2e3, "eta": 0.9},  # Cq in Hz
)

spin_systems = [SpinSystem(sites=[site])]
```

Method

Use the generic 2D method, *Method2D*, to generate a shifting-d echo method. The reported shifting-d 2D sequence is a correlation of the shielding frequencies to the first-order quadrupolar frequencies. Here, we create a correlation method using the `freq_contrib` attribute, which acts as a switch for including the frequency contributions from interaction during the event.

In the following method, we assign the `["Quad1_2"]` and `["Shielding1_0", "Shielding1_2"]` as the value to the `freq_contrib` key. The *Quad1_2* is an enumeration for selecting the first-order second-rank quadrupolar frequency contributions. *Shielding1_0* and *Shielding1_2* are enumerations for the first-order shielding with zeroth and second-rank tensor contributions, respectively. See *FrequencyEnum* (page 316) for details.

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)
```

(continues on next page)

(continued from previous page)

```
shifting_d = Method2D(
    channels=["2H"],
    magnetic_flux_density=9.395,  # in T
    spectral_dimensions=[
        {
            **spectral_dims[0],
            "label": "Quadrupolar frequency",
            "events": [
                {
                    "rotor_frequency": 0,
                    "transition_query": {"P": [-1]},
                    "freq_contrib": ["Quad1_2"],
                }
            ],
        },
        {
            **spectral_dims[1],
            "label": "Paramagnetic shift",
            "events": [
                {
                    "rotor_frequency": 0,
                    "transition_query": {"P": [-1]},
                    "freq_contrib": ["Shielding1_0", "Shielding1_2"],
                }
            ],
        },
    ],
    experiment=experiment,  # also add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = shifting_d.get_transition_pathways(sys)
```

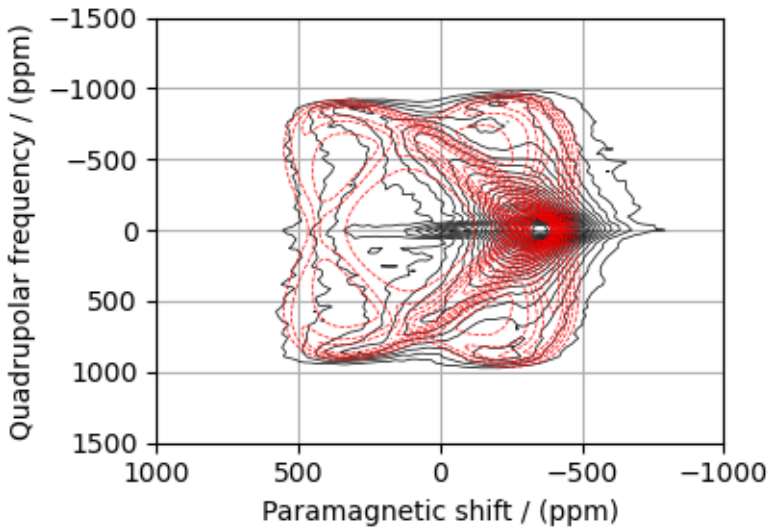
Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[shifting_d])
sim.config.integration_volume = "hemisphere"
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=(0, 1)),
        sp.apodization.Gaussian(FWHM="5 kHz", dim_index=0),  # along dimension 0
        sp.apodization.Gaussian(FWHM="5 kHz", dim_index=1),  # along dimension 1
```

(continues on next page)

(continued from previous page)

```
        sp.FFT(dim_index=(0, 1)),
        sp.Scale(factor=5e8),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(processed_data, colors="r", linestyles="--", **options)
ax.set_xlim(1000, -1000)
ax.set_ylim(1500, -1500)
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM          5     -inf      inf     True     None
SP_0_operation_2_Gaussian_FWHM          5     -inf      inf     True     None
SP_0_operation_4_Scale_factor       5e+08     -inf      inf     True     None
sys_0_abundance                       100        0      100    False      100
```

(continues on next page)

(continued from previous page)

```
sys_0_site_0_isotropic_chemical_shift        -90     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq             7.52e+04     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta                 0.9        0        1     True     None
sys_0_site_0_shielding_symmetric_alpha       0.7     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_beta          2     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_eta        0.15        0        1     True     None
sys_0_site_0_shielding_symmetric_gamma         3     -inf      inf     True     None
sys_0_site_0_shielding_symmetric_zeta       -610     -inf      inf     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 359
    # data points      = 65536
    # variables        = 11
    chi-square         = 212297.385
    reduced chi-square = 3.23994483
    Akaike info crit   = 77052.2569
    Bayesian info crit = 77152.2508
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:  -97.1846806 +/- 0.19427758 (0.20%) (init = -90)
    sys_0_site_0_shielding_symmetric_zeta:  -565.058769 +/- 0.40287314 (0.07%) (init = -610)
    sys_0_site_0_shielding_symmetric_eta:    0.05364167 +/- 0.00616187 (11.49%) (init = 0.15)
    sys_0_site_0_shielding_symmetric_alpha:  0.97731504 +/- 0.13101180 (13.41%) (init = 0.7)
    sys_0_site_0_shielding_symmetric_beta:   2.01773319 +/- 0.00473113 (0.23%) (init = 2)
    sys_0_site_0_shielding_symmetric_gamma:  3.12058212 +/- 0.21619387 (6.93%) (init = 3)
    sys_0_site_0_quadrupolar_Cq:             75656.7971 +/- 22.5870158 (0.03%) (init = 75200)
    sys_0_site_0_quadrupolar_eta:            0.95111166 +/- 7.3610e-04 (0.08%) (init = 0.9)
    sys_0_abundance:                         100.000000 +/- 0.00000000 (0.00%) == '100'
    SP_0_operation_1_Gaussian_FWHM:          16.5165225 +/- 0.02199109 (0.13%) (init = 5)
    SP_0_operation_2_Gaussian_FWHM:          4.11079567 +/- 0.04248529 (1.03%) (init = 5)
    SP_0_operation_4_Scale_factor:           7.3499e+08 +/- 437188.510 (0.06%) (init = 5e+08)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_shielding_symmetric_beta, sys_0_site_0_shielding_symmetric_gamma)  =  0.995
    C(sys_0_site_0_shielding_symmetric_alpha, sys_0_site_0_shielding_symmetric_beta)  = -0.987
    C(sys_0_site_0_shielding_symmetric_alpha, sys_0_site_0_shielding_symmetric_gamma) = -0.986
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_shielding_symmetric_zeta)   = -0.638
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_1_Gaussian_FWHM)           = -0.602
    C(sys_0_site_0_quadrupolar_Cq, sys_0_site_0_quadrupolar_eta)                      = -0.543
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_2_Gaussian_FWHM)                   =  0.521
    C(sys_0_site_0_shielding_symmetric_eta, sys_0_site_0_shielding_symmetric_gamma)   = -0.469
    C(sys_0_site_0_shielding_symmetric_eta, sys_0_site_0_shielding_symmetric_beta)    = -0.455
    C(sys_0_site_0_shielding_symmetric_beta, SP_0_operation_1_Gaussian_FWHM)          =  0.382
    C(sys_0_site_0_shielding_symmetric_gamma, SP_0_operation_1_Gaussian_FWHM)         =  0.382
```

(continues on next page)

(continued from previous page)

```
    C(sys_0_site_0_shielding_symmetric_eta, sys_0_site_0_shielding_symmetric_alpha)   =  0.344
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_4_Scale_factor)                  =  0.331
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_4_Scale_factor)           = -0.326
    C(sys_0_site_0_shielding_symmetric_alpha, SP_0_operation_1_Gaussian_FWHM)         = -0.321
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_2_Gaussian_FWHM)                    = -0.220
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_4_Scale_factor)           =  0.217
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_shielding_symmetric_eta)    =  0.215
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_2_Gaussian_FWHM)                 = -0.210
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_4_Scale_factor)                     =  0.162
    C(SP_0_operation_2_Gaussian_FWHM, SP_0_operation_4_Scale_factor)                  =  0.126
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_eta)            =  0.123
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_1_Gaussian_FWHM)                   = -0.123
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_2_Gaussian_FWHM)           =  0.117
    C(sys_0_site_0_isotropic_chemical_shift, SP_0_operation_2_Gaussian_FWHM)          =  0.114
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_shielding_symmetric_beta)   =  0.107
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(best_fit, colors="r", linestyles="--", **options)
ax.set_xlim(1000, -1000)
ax.set_ylim(1500, -1500)
plt.grid()
plt.tight_layout()
plt.show()
```

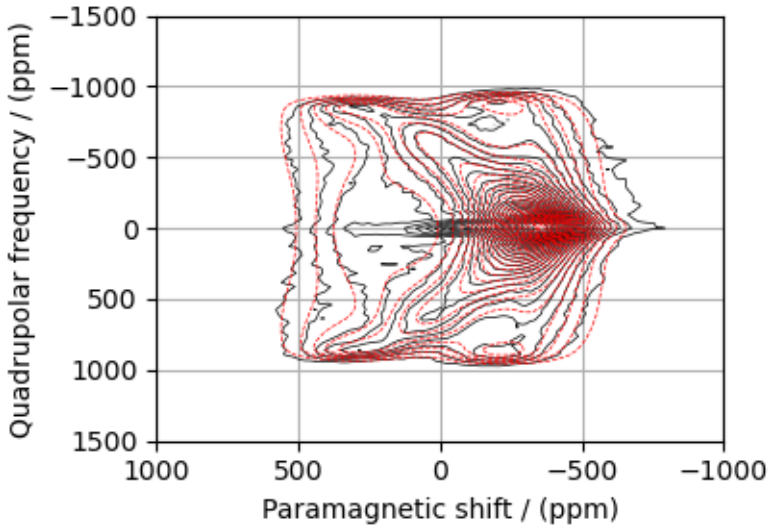
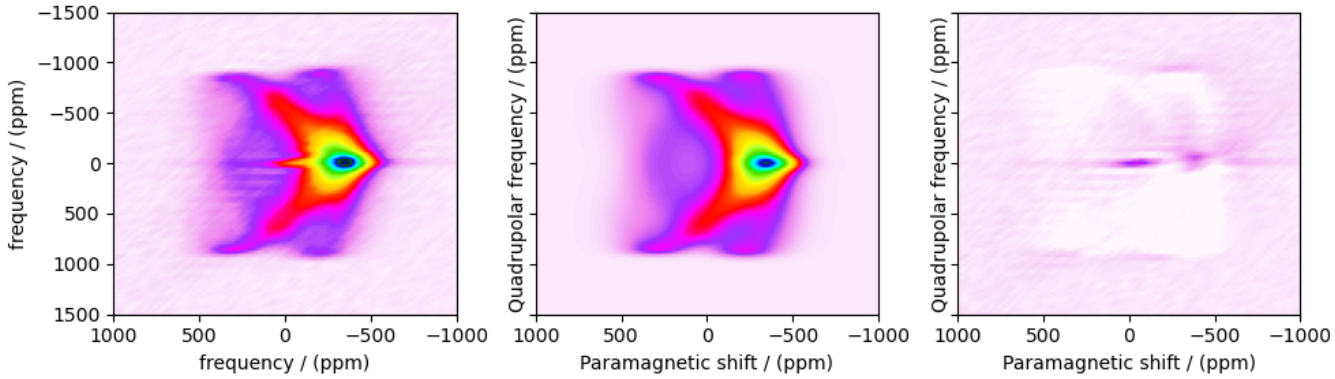


Image plots with residuals

```
residuals = sf.residuals(sim, processor)[0]

fig, ax = plt.subplots(
    1, 3, sharey=True, figsize=(10, 3.0), subplot_kw={"projection": "csdm"}
)
vmax, vmin = experiment.max(), experiment.min()
for i, dat in enumerate([experiment, best_fit, residuals]):
    ax[i].imshow(dat, aspect="auto", cmap="gist_ncar_r", vmax=vmax, vmin=vmin)
    ax[i].set_xlim(1000, -1000)
ax[0].set_ylim(1500, -1500)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 30.805 seconds)

12.2.6 $CoCl_2.2D_2O$, ^2H (I=1) Shifting-d echo

^2H (I=1) 2D NMR CSA-Quad 1st order correlation spectrum.

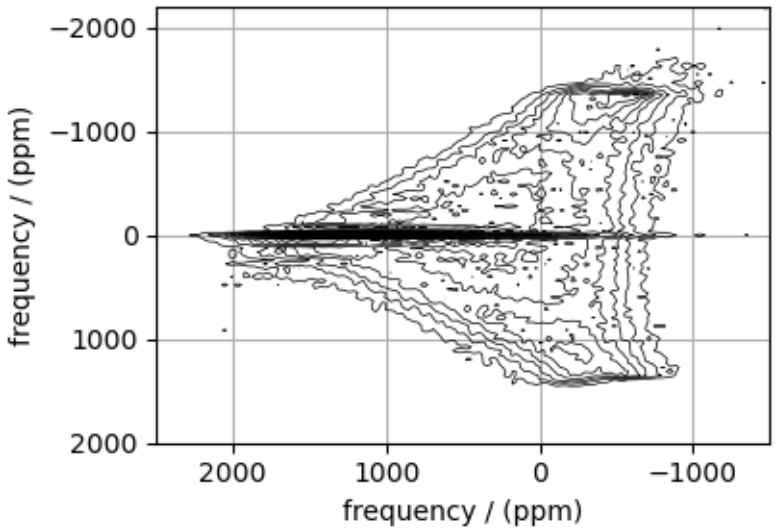
The following is an example of fitting static shifting-d echo NMR correlation spectrum of $NiCl_2 \cdot 2D_2O$ crystalline solid. The spectrum used here is from Walder *et al.*[1].

```
import numpy as np
import csdmpy as cp
import matplotlib.pyplot as plt
from lmfit import Minimizer, report_fit

from mrsimulator import Simulator, Site, SpinSystem
from mrsimulator.methods import Method2D
from mrsimulator import signal_processing as sp
from mrsimulator.utils import spectral_fitting as sf
from mrsimulator.utils import get_spectral_dimensions
```

[1] Walder B.J, Patterson A.M., Baltisberger J.H, and Grandinetti P.J Hydrogen motional disorder in crystalline iron group chloride dihydrates spectroscopy, J. Chem. Phys. (2018) **149**, 084503. DOI: 10.1063/1.5037151

Import the dataset

```
filename = "https://sandbox.zenodo.org/record/830903/files/CoCl2.2D2O.csdf"
experiment = cp.load(filename)

# standard deviation of noise from the dataset
sigma = 11.11578

# For spectral fitting, we only focus on the real part of the complex dataset
experiment = experiment.real

# Convert the coordinates along each dimension from Hz to ppm.
_ = [item.to("ppm", "nmr_frequency_ratio") for item in experiment.dimensions]

# plot of the dataset.
max_amp = experiment.max()
levels = (np.arange(24) + 1) * max_amp / 25  # contours are drawn at these levels.
options = dict(levels=levels, linewidths=0.5)  # plot options

plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.set_xlim(2500, -1500)
ax.set_ylim(2000, -2200)
plt.grid()
plt.tight_layout()
plt.show()
```



Create a fitting model

Guess model

Create a guess list of spin systems.

```
site = Site(
    isotope="2H",
    isotropic_chemical_shift=200,  # in ppm
    shielding_symmetric={
        "zeta": -1300,  # in ppm
        "eta": 0.2,
        "alpha": np.pi,  # in rads
        "beta": np.pi / 2,  # in rads
        "gamma": np.pi / 2,  # in rads
    },
    quadrupolar={"Cq": 110e3, "eta": 0.83},  # Cq in Hz
)

spin_systems = [SpinSystem(sites=[site])]
```

Method

Use the generic 2D method, *Method2D*, to generate a shifting-d echo method.

```
# Get the spectral dimension parameters from the experiment.
spectral_dims = get_spectral_dimensions(experiment)

shifting_d = Method2D(
    channels=["2H"],
    magnetic_flux_density=9.395,  # in T
    spectral_dimensions=[
        {
            **spectral_dims[0],
            "label": "Quadrupolar frequency",
            "events": [
                {
                    "rotor_frequency": 0,
                    "transition_query": {"P": [-1]},
                    "freq_contrib": ["Quad1_2"],
                }
            ],
        },
        {
            **spectral_dims[1],
            "label": "Paramagnetic shift",
            "events": [
                {
                    "rotor_frequency": 0,
                    "transition_query": {"P": [-1]},
                    "freq_contrib": ["Shielding1_0", "Shielding1_2"],
                }
            ],
        },
```

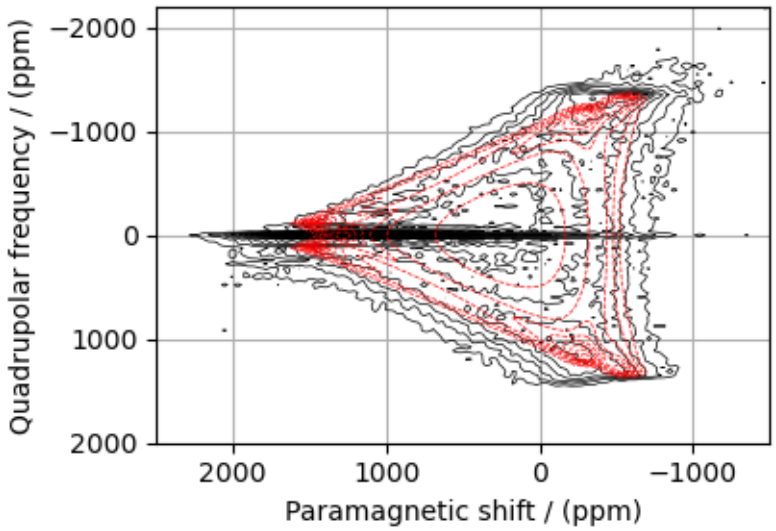
(continues on next page)

(continued from previous page)

```
    ],
    experiment=experiment,  # also add the measurement to the method.
)

# Optimize the script by pre-setting the transition pathways for each spin system from
# the method.
for sys in spin_systems:
    sys.transition_pathways = shifting_d.get_transition_pathways(sys)
```

Guess Spectrum

```
# Simulation
# ----------
sim = Simulator(spin_systems=spin_systems, methods=[shifting_d])
sim.config.integration_volume = "hemisphere"
sim.run()

# Post Simulation Processing
# --------------------------
processor = sp.SignalProcessor(
    operations=[
        # Gaussian convolution along both dimensions.
        sp.IFFT(dim_index=0),
        sp.apodization.Gaussian(FWHM="10 kHz", dim_index=0),  # along dimension 0
        sp.FFT(dim_index=0),
        sp.Scale(factor=5e8),
    ]
)
processed_data = processor.apply_operations(data=sim.methods[0].simulation).real

# Plot of the guess Spectrum
# --------------------------
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(processed_data, colors="r", linestyles="--", **options)
ax.set_xlim(2500, -1500)
ax.set_ylim(2000, -2200)
plt.grid()
plt.tight_layout()
plt.show()
```



Least-squares minimization with LMFIT

Use the *make_LMFIT_params()* (page 351) for a quick setup of the fitting parameters.

```
params = sf.make_LMFIT_params(sim, processor)
params["sys_0_site_0_shielding_symmetric_alpha"].vary = False
params["sys_0_site_0_shielding_symmetric_beta"].vary = False
params["sys_0_site_0_shielding_symmetric_gamma"].vary = False
print(params.pretty_print(columns=["value", "min", "max", "vary", "expr"]))
```

Out:

```
Name                                    Value      Min      Max     Vary     Expr
SP_0_operation_1_Gaussian_FWHM             10     -inf      inf     True     None
SP_0_operation_3_Scale_factor           5e+08     -inf      inf     True     None
sys_0_abundance                           100        0      100    False      100
sys_0_site_0_isotropic_chemical_shift     200     -inf      inf     True     None
sys_0_site_0_quadrupolar_Cq            1.1e+05     -inf      inf     True     None
sys_0_site_0_quadrupolar_eta              0.83        0        1     True     None
sys_0_site_0_shielding_symmetric_alpha   3.142     -inf      inf    False     None
sys_0_site_0_shielding_symmetric_beta    1.571     -inf      inf    False     None
sys_0_site_0_shielding_symmetric_eta       0.2        0        1     True     None
sys_0_site_0_shielding_symmetric_gamma   1.571     -inf      inf    False     None
sys_0_site_0_shielding_symmetric_zeta    -1300     -inf      inf     True     None
None
```

Solve the minimizer using LMFIT

```
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
report_fit(result)
```

Out:

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 442
    # data points      = 65536
    # variables        = 7
    chi-square         = 309489.005
    reduced chi-square = 4.72293191
    Akaike info crit   = 101747.037
    Bayesian info crit = 101810.670
[[Variables]]
    sys_0_site_0_isotropic_chemical_shift:   202.512829 +/- 0.89419451 (0.44%) (init = 200)
    sys_0_site_0_shielding_symmetric_zeta:  -1355.90491 +/- 1.48289502 (0.11%) (init = -1300)
    sys_0_site_0_shielding_symmetric_eta:    0.22077495 +/- 0.00550582 (2.49%) (init = 0.2)
    sys_0_site_0_shielding_symmetric_alpha:  3.141593 (fixed)
    sys_0_site_0_shielding_symmetric_beta:   1.570796 (fixed)
    sys_0_site_0_shielding_symmetric_gamma:  1.570796 (fixed)
    sys_0_site_0_quadrupolar_Cq:             114245.459 +/- 94.9713991 (0.08%) (init = 110000)
    sys_0_site_0_quadrupolar_eta:            0.99999667 +/- 0.00311737 (0.31%) (init = 0.83)
    sys_0_abundance:                         100.000000 +/- 0.00000000 (0.00%) == '100'
    SP_0_operation_1_Gaussian_FWHM:          37.8199216 +/- 0.18725437 (0.50%) (init = 10)
    SP_0_operation_3_Scale_factor:           9.3760e+08 +/- 1661029.24 (0.18%) (init = 5e+08)
[[Correlations]] (unreported correlations are < 0.100)
    C(sys_0_site_0_quadrupolar_Cq, sys_0_site_0_quadrupolar_eta)                    = -0.935
    C(sys_0_site_0_shielding_symmetric_eta, SP_0_operation_1_Gaussian_FWHM)         = -0.636
    C(SP_0_operation_1_Gaussian_FWHM, SP_0_operation_3_Scale_factor)                =  0.355
    C(sys_0_site_0_shielding_symmetric_eta, sys_0_site_0_quadrupolar_eta)           = -0.354
    C(sys_0_site_0_shielding_symmetric_eta, sys_0_site_0_quadrupolar_Cq)            =  0.347
    C(sys_0_site_0_quadrupolar_eta, SP_0_operation_1_Gaussian_FWHM)                 =  0.202
    C(sys_0_site_0_quadrupolar_Cq, SP_0_operation_1_Gaussian_FWHM)                  = -0.197
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_shielding_symmetric_zeta) =  0.153
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_3_Scale_factor)         = -0.132
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_quadrupolar_Cq)           = -0.117
    C(sys_0_site_0_shielding_symmetric_zeta, sys_0_site_0_quadrupolar_eta)          =  0.115
    C(sys_0_site_0_isotropic_chemical_shift, sys_0_site_0_quadrupolar_Cq)           = -0.112
    C(sys_0_site_0_shielding_symmetric_zeta, SP_0_operation_1_Gaussian_FWHM)        =  0.106
```

The best fit solution

```
best_fit = sf.bestfit(sim, processor)[0]

# Plot the spectrum
plt.figure(figsize=(4.25, 3.0))
ax = plt.subplot(projection="csdm")
ax.contour(experiment, colors="k", **options)
ax.contour(best_fit, colors="r", linestyles="--", **options)
ax.set_xlim(2500, -1500)
ax.set_ylim(2000, -2200)
plt.grid()
plt.tight_layout()
plt.show()
```

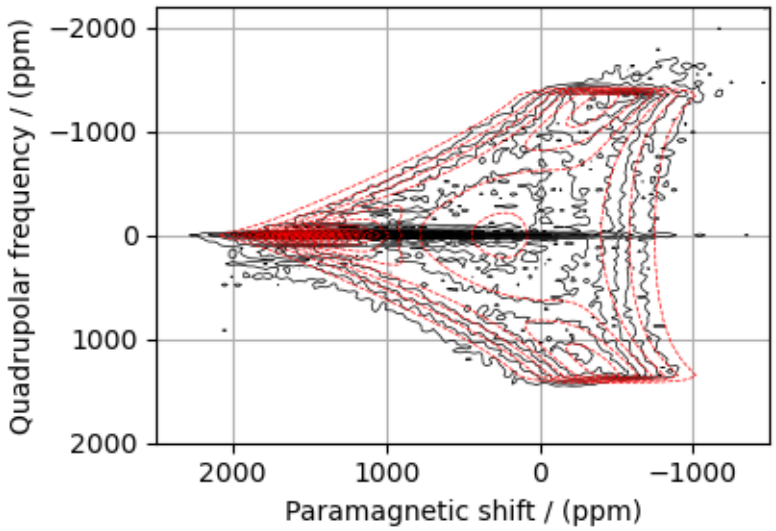
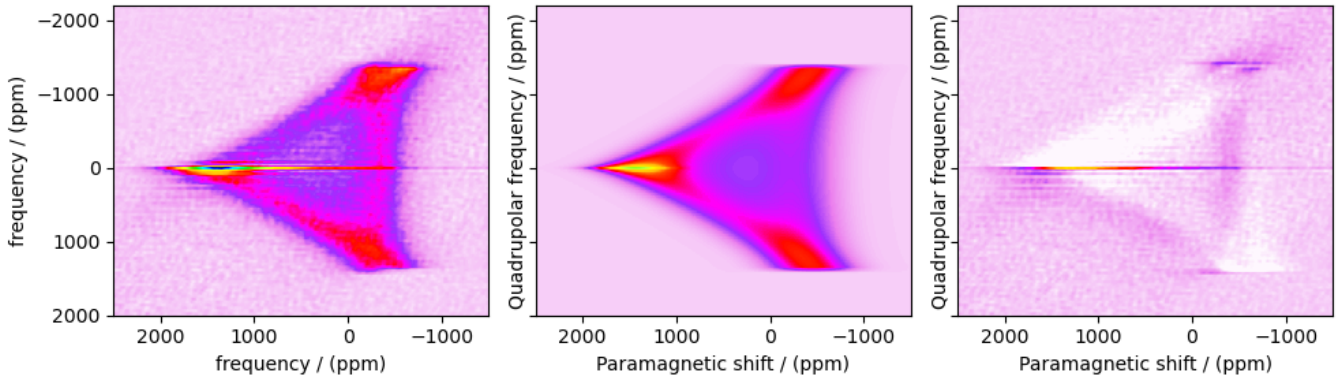


Image plots with residuals

```
residuals = sf.residuals(sim, processor)[0]

fig, ax = plt.subplots(
    1, 3, sharey=True, figsize=(10, 3.0), subplot_kw={"projection": "csdm"}
)
vmax, vmin = experiment.max(), experiment.min()
for i, dat in enumerate([experiment, best_fit, residuals]):
    ax[i].imshow(dat, aspect="auto", cmap="gist_ncar_r", vmax=vmax, vmin=vmin)
    ax[i].set_xlim(2500, -1500)
ax[0].set_ylim(2000, -2200)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 35.582 seconds)

CHAPTER

THIRTEEN

PERFORMANCE BENCHMARK

One of the objectives in the design of the `mrsimulator` library is to enable fast NMR spectrum simulation. For this, we have put a considerable effort into the optimization of the library. The following benchmark shows the performance of the library in computing the solid-state NMR spectra from single-site spin systems for the shift and quadrupolar tensor interactions at static and MAS conditions.

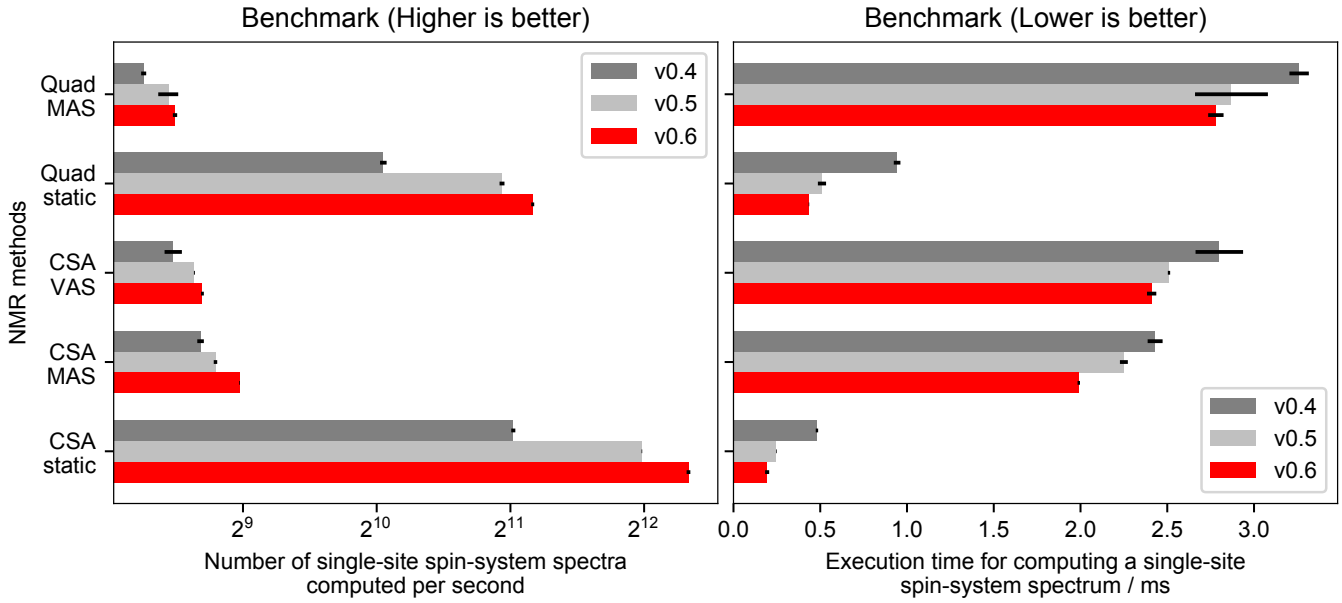


Figure 13.1: (Left) The number of single-site spin systems computer per seconds. (Right) The execution time (in ms) in computing spectrum from a single-site spin system.

Benchmark specs

The benchmarks were performed on a 2.3 GHz Quad-Core Intel Core i5 Laptop using 8 GB 2133 MHz LPDDR3 memory. For consistent benchmarking, 1000 single-site spin systems were constructed, where the tensor parameters of the sites (*zeta* and *eta* for the shielding tensor, and *Cq* and *eta* for the quadrupolar tensor) were randomly populated. The execution time for this setup was recorded. and the process repeated 70 times. The reported value is the mean and the standard deviation.

All calculations were performed using the default Simulator *config* (page 290) attribute values.

13.1 Benchmark for the previous versions

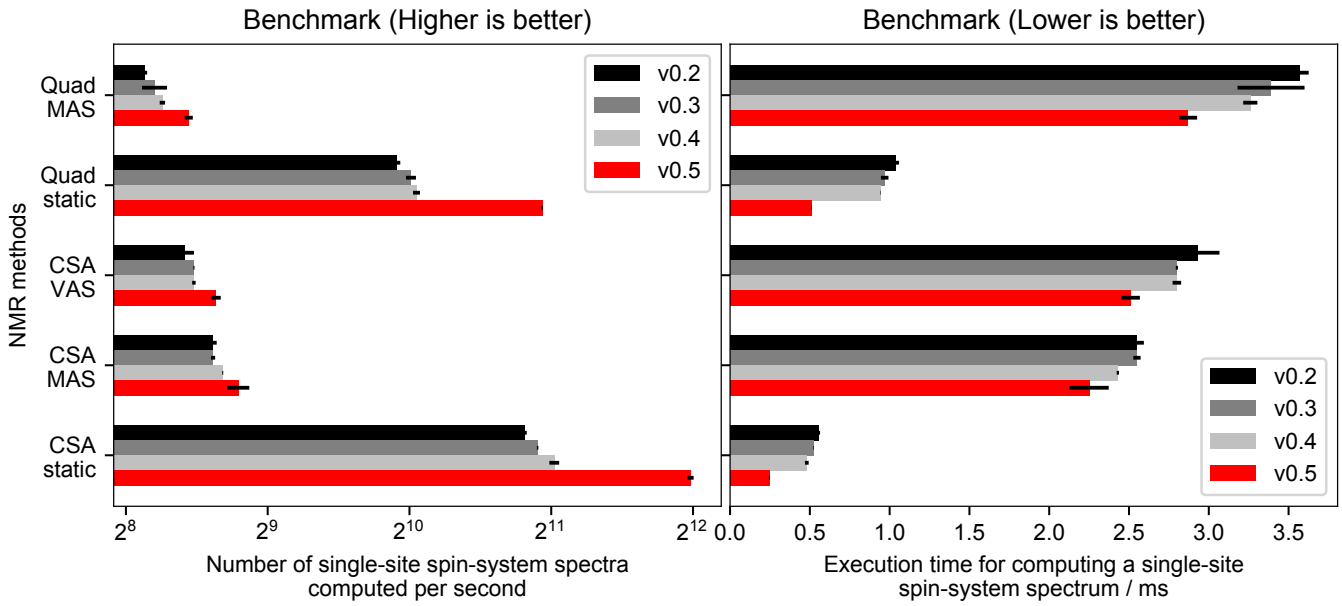


Figure 13.2: (Left) The number of single-site spin systems computer per seconds. (Right) The execution time (in ms) in computing spectrum from a single-site spin system.

Part V

Theory

CHAPTER

FOURTEEN

HOW DOES MRSIMULATOR WORK?

The NMR spectral simulation in `mrsimulator` is based on Symmetry Pathways in Solid-State NMR by Grandinetti *et al.*[1]

14.1 Introduction to NMR frequency components

The nuclear magnetic resonance (NMR) frequency, $\Omega(\Theta, i, j)$, for the $|i\rangle \rightarrow |j\rangle$ transition, where $|i\rangle$ and $|j\rangle$ are the eigenstates of the stationary-state semi-classical Hamiltonian, can be written as a sum of frequency components,

$$\Omega(\Theta, i, j) = \sum_k \Omega_k(\Theta, i, j), \tag{14.1}$$

where Θ is the sample's lattice spatial orientation described with the Euler angles $\Theta = (\alpha, \beta, \gamma)$, and Ω_k is the frequency component from the k^{th} interaction of the stationary-state semi-classical Hamiltonian.

Each frequency component, $\Omega_k(\Theta, i, j)$, is separated into three parts,

$$\Omega_k(\Theta, i, j) = \omega_k \; \Xi_L^{(k)}(\Theta) \; \xi_L^{(k)}(i, j), \tag{14.2}$$

where ω_k is the size of the k^{th} frequency component, and $\Xi_L^{(k)}(\Theta)$ and $\xi_L^{(k)}(i, j)$ are the sample's spatial orientation and quantized NMR transition functions corresponding to the L^{th} rank spatial and spin irreducible spherical tensors, respectively.

The spatial orientation function, $\Xi_L^{(k)}(\Theta)$, in Eq. (14.2), is defined in the laboratory frame, where the z-axis is the direction of the external magnetic field. This function is the spatial contribution to the observed frequency component arising from the rotation of the L^{th}-rank irreducible tensor, $\varrho_{L,n}^{(k)}$, from the principal axis system, to the lab frame via Wigner rotation which follows,

$$\Xi_L^{(k)}(\Theta) = \sum_{n_0=-L}^{L} D_{n_0,0}^L(\Theta_0) \sum_{n_1=-L}^{L} D_{n_1,n_0}^L(\Theta_1) \; ... \; \sum_{n_i=-L}^{L} D_{n_i,n}^L(\Theta_i) \;\; \varrho_{L,n}^{(k)}. \tag{14.3}$$

Here, the term $D_{n_i,n_j}^L(\Theta)$ is the Wigner rotation matrix element, generically denoted as,

$$D_{n_i,n_j}^L(\Theta) = e^{-in_i\alpha} d_{n_i,n_j}^L(\beta) e^{-in_j\gamma}, \tag{14.4}$$

where $d_{n_i,n_j}^L(\beta)$ is Wigner small d element.

---

[1] Grandinetti, P. J., Ash, J. T., Trease, N. M. Symmetry pathways in solid-state NMR, PNMRS 2011 **59**, *2*, 121-196. DOI: 10.1016/j.pnmrs.2010.11.003

In the case of the single interaction Hamiltonian, that is, in the absence of cross-terms, `mrsimulator` further defines the product of the size of the k^{th} frequency component, ω_k, and the L^{th}-rank irreducible tensor components, $\varrho^{(k)}_{L,n}$, in the principal axis system of the interaction tensor, $\boldsymbol{\rho}^{(\lambda)}$, as the scaled spatial orientation tensor (sSOT) component,

$$\varsigma^{(k)}_{L,n} = \omega_k \varrho^{(k)}_{L,n}, \tag{14.5}$$

of rank L, also defined in the principal axis system of the interaction tensor. Using Eqs. (14.3) and (14.5), we re-express Eq. (14.2) as

$$\Omega_k(\Theta, i, j) = \sum_{n_0=-L}^{L} D^L_{n_0,0}(\Theta_0) \sum_{n_1=-L}^{L} D^L_{n_1,n_0}(\Theta_1) \ ... \ \sum_{n_i=-L}^{L} D^L_{n_i,n}(\Theta_i) \ \ \varpi^{(k)}_{L,n}, \tag{14.6}$$

where

$$\varpi^{(k)}_{L,n} = \varsigma^{(k)}_{L,n} \ \ \xi^{(k)}_L(i,j) \tag{14.7}$$

is the frequency tensor component (FT) of rank L, defined in the principal axis system of the interaction tensor and corresponds to the $|i\rangle \rightarrow |j\rangle$ spin transition.

14.2 Scaled spatial orientation tensor (sSOT) components in PAS, $\varsigma^{(k)}_{L,n}$

14.2.1 Single nucleus scaled spatial orientation tensor components

Nuclear shielding interaction

The nuclear shielding tensor, $\boldsymbol{\rho}^{(\sigma)}$, is a second rank reducible tensor which can be decomposed into a sum of the zeroth-rank isotropic, first-rank anti-symmetric, and second-rank traceless symmetric irreducible spherical tensors. In the principal axis system, the zeroth-rank, $\rho^{(\sigma)}_{0,0}$ and the second-rank, $\rho^{(\sigma)}_{2,n}$, irreducible tensors follow,

$$\rho^{(\sigma)}_{0,0} = -\sqrt{3}\sigma_{\text{iso}}, \quad \rho^{(\sigma)}_{2,0} = \sqrt{\tfrac{3}{2}}\zeta_\sigma, \quad \rho^{(\sigma)}_{2,\pm1} = 0, \quad \rho^{(\sigma)}_{2,\pm2} = -\tfrac{1}{2}\eta_\sigma\zeta_\sigma, \tag{14.8}$$

where $\sigma_{\text{iso}}, \zeta_\sigma$, and η_σ are the isotropic nuclear shielding, shielding anisotropy, and shielding asymmetry of the site, respectively. The shielding anisotropy, and asymmetry are defined using Haeberlen notation.

First-order perturbation

The size of the frequency component, ω_k, from the first-order perturbation expansion of Nuclear shielding Hamiltonian is $-\omega_0 = \gamma B_0$, where ω_0 is the Larmor angular frequency of the nucleus, and γ, B_0 are the gyromagnetic ratio of the nucleus and the macroscopic magnetic flux density of the applied external magnetic field, respectively. The relation between $\varrho^{(\sigma)}_{L,n}$ and $\rho^{(\sigma)}_{L,n}$ follows,

$$\begin{aligned} \varrho^{(\sigma)}_{0,0} &= -\frac{1}{\sqrt{3}}\rho^{(\sigma)}_{0,0} \\ \varrho^{(\sigma)}_{2,n} &= \sqrt{\frac{2}{3}}\rho^{(\sigma)}_{2,n} \end{aligned} \tag{14.9}$$

Table 14.1: A list of scaled spatial orientation tensors in the principal axis system of the nuclear shielding tensor, $\varsigma^{(k)}_{L,n}$ from Eq. (14.5), of rank L resulting from the Mth order perturbation expansion of the Nuclear shielding Hamiltonian is presented.

Order, M	Rank, L	$\varsigma^{(k)}_{L,n} = \omega_k \varrho^{(k)}_{L,n}$
1	0	$\varsigma^{(\sigma)}_{0,0} = -\omega_0 \sigma_{\text{iso}}$
1	2	$\varsigma^{(\sigma)}_{2,0} = -\omega_0 \zeta_\sigma$, $\varsigma^{(\sigma)}_{2,\pm1} = 0$, $\varsigma^{(\sigma)}_{2,\pm2} = \frac{1}{\sqrt{6}} \omega_0 \eta_\sigma \zeta_\sigma$

Electric quadrupole interaction

The electric field gradient (efg) tensor, $\boldsymbol{\rho}^{(q)}$, is also a second-rank tensor, however, unlike the nuclear shielding tensor, the efg tensor is always a symmetric second-rank irreducible tensor. In the principal axis system, this tensor is given as,

$$\rho^{(q)}_{2,0} = \sqrt{\tfrac{3}{2}} \zeta_q, \quad \rho^{(q)}_{2,\pm1} = 0, \quad \rho^{(q)}_{2,\pm2} = -\tfrac{1}{2} \eta_q \zeta_q, \tag{14.10}$$

where ζ_q, and η_q are the efg tensor anisotropy, and asymmetry of the site, respectively. The efg anisotropy, and asymmetry are defined using Haeberlen convention.

First-order perturbation

The size of the frequency component from the first-order perturbation expansion of Electric quadrupole Hamiltonian is $\omega_k = \omega_q$, where $\omega_q = \frac{6\pi C_q}{2I(2I-1)}$ is the quadrupole splitting angular frequency. Here, C_q is the quadrupole coupling constant, and I is the spin quantum number of the quadrupole nucleus. The relation between $\varrho^{(q)}_{L,n}$ and $\rho^{(q)}_{L,n}$ follows,

$$\varrho^{(q)}_{2,n} = \frac{1}{3\zeta_q} \rho^{(q)}_{2,n}. \tag{14.11}$$

Second-order perturbation

The size of the frequency component from the second-order perturbation expansion of Electric quadrupole Hamiltonian is $\omega_k = \frac{\omega_q^2}{\omega_0}$, where ω_0 is the Larmor angular frequency of the quadrupole nucleus. The relation between $\varrho^{(qq)}_{L,n}$ and $\rho^{(q)}_{L,n}$ follows,

$$\varrho^{(qq)}_{L,n} = \frac{1}{9\zeta_q^2} \sum_{m=-2}^{2} \langle L\ n \mid 2\ 2\ m\ n-m \rangle\, \rho^{(q)}_{2,m}\ \rho^{(q)}_{2,n-m}, \tag{14.12}$$

where $\langle L\ M \mid l_1\ l_2\ m_1\ m_2 \rangle$ is the Clebsch Gordan coefficient.

Table 14.2: A list of scaled spatial orientation tensors in the principal axis system of the efg tensor, $\varsigma^{(k)}_{L,n}$ from Eq. (14.5), of rank L resulting from the Mth order perturbation expansion of the Electric Quadrupole Hamiltonian is presented.

Order, M	Rank, L	$\varsigma^{(k)}_{L,n} = \omega_k \varrho^{(k)}_{L,n}$
1	2	$\varsigma^{(q)}_{2,0} = \frac{1}{\sqrt{6}}\omega_q$, $\varsigma^{(q)}_{2,\pm1} = 0$, $\varsigma^{(q)}_{2,\pm2} = -\frac{1}{6}\eta_q\omega_q$
2	0	$\varsigma^{(qq)}_{0,0} = \frac{\omega_q^2}{\omega_0}\frac{1}{6\sqrt{5}}\left(\frac{\eta_q^2}{3}+1\right)$
2	2	$\varsigma^{(qq)}_{2,0} = \frac{\omega_q^2}{\omega_0}\frac{\sqrt{2}}{6\sqrt{7}}\left(\frac{\eta_q^2}{3}-1\right)$, $\varsigma^{(qq)}_{2,\pm1} = 0$, $\varsigma^{(qq)}_{2,\pm2} = -\frac{\omega_q^2}{\omega_0}\frac{1}{3\sqrt{21}}\eta_q$
2	4	$\varsigma^{(qq)}_{4,0} = \frac{\omega_q^2}{\omega_0}\frac{1}{\sqrt{70}}\left(\frac{\eta_q^2}{18}+1\right)$, $\varsigma^{(qq)}_{4,\pm1} = 0$, $\varsigma^{(qq)}_{4,\pm2} = -\frac{\omega_q^2}{\omega_0}\frac{1}{6\sqrt{7}}\eta_q$, $\varsigma^{(qq)}_{4,\pm3} = 0$, $\varsigma^{(qq)}_{4,\pm4} = \frac{\omega_q^2}{\omega_0}\frac{1}{36}\eta_q^2$

14.2.2 Coupled nucleus scaled spatial orientation tensor components

Weak J-coupling interaction

The J-coupling tensor, $\boldsymbol{\rho}^{(J)}$, is a second rank reducible tensor which can be decomposed into a sum of the zeroth-rank isotropic, first-rank anti-symmetric, and second-rank traceless symmetric irreducible spherical tensors. In the principal axis system, the zeroth-rank, $\rho^{(J)}_{0,0}$ and the second-rank, $\rho^{(J)}_{2,n}$, irreducible tensors follow,

$$\rho^{(J)}_{0,0} = -\sqrt{3}J_{\text{iso}}, \quad \rho^{(J)}_{2,0} = \sqrt{\tfrac{3}{2}}\zeta_J, \quad \rho^{(J)}_{2,\pm1} = 0, \quad \rho^{(J)}_{2,\pm2} = -\tfrac{1}{2}\eta_J\zeta_J, \tag{14.13}$$

where J_{iso}, ζ_J, and η_J are the isotropic J-coupling, coupling anisotropy and asymmetry parameters, respectively. The J anisotropy and asymmetry are defined using Haeberlen notation.

First-order perturbation

The size of the frequency component from the first-order perturbation expansion of weak J-coupling Hamiltonian is $\omega_k = 2\pi$. The relation between $\varrho^{(J)}_{L,n}$ and $\rho^{(J)}_{L,n}$ follows,

$$\begin{aligned} \varrho^{(J)}_{0,0} &= -\frac{1}{\sqrt{3}}\rho^{(J)}_{0,0} \\ \varrho^{(J)}_{2,n} &= \sqrt{\frac{2}{3}}\rho^{(J)}_{2,n} \end{aligned} \tag{14.14}$$

Table 14.3: A list of scaled spatial orientation tensors in the principal axis system of the J-coupling tensor, $\varsigma^{(k)}_{L,n}$ from Eq. (14.5), of rank L resulting from the Mth order perturbation expansion of the J-coupling Hamiltonian is presented.

Order, M	Rank, L	$\varsigma^{(k)}_{L,n} = \omega_k \varrho^{(k)}_{L,n}$
1	0	$\varsigma^{(J)}_{0,0} = 2\pi J_{\text{iso}}$
1	2	$\varsigma^{(J)}_{2,0} = 2\pi\zeta_J$, $\varsigma^{(J)}_{2,\pm1} = 0$, $\varsigma^{(J)}_{2,\pm2} = -\frac{1}{\sqrt{6}} 2\pi\eta_J\zeta_J$

Weak dipolar-coupling interaction

The dipolar-coupling tensor, $\boldsymbol{\rho}^{(d)}$, is a second rank reducible tensor which can be decomposed as a second-rank traceless symmetric irreducible spherical tensors. In the principal axis system, the second-rank, $\rho^{(d)}_{2,n}$, irreducible tensors follow,

$$\rho^{(d)}_{2,0} = \sqrt{\tfrac{3}{2}}\zeta_d, \quad \rho^{(d)}_{2,\pm1} = 0, \quad \rho^{(d)}_{2,\pm2} = 0, \tag{14.15}$$

where ζ_d is second-rank symmetric dipolar coupling tensor anisotropy given as

$$\zeta_d = \frac{2}{r^3} \tag{14.16}$$

where r is the distance between two coupled magnetic dipoles. The dipolar splitting is given as

$$\omega_d = -\frac{\mu_0}{4\pi}\frac{\gamma_1\gamma_2\hbar}{r^3} = -\frac{\mu_0}{8\pi}\zeta_d\gamma_1\gamma_2\hbar \tag{14.17}$$

and the dipolar coupling constant, $D = \frac{\omega_d}{2\pi}$.

First-order perturbation

The size of the frequency component from the first-order perturbation expansion of weak J-coupling Hamiltonian is $\omega_k = \frac{2\omega_d}{\zeta_d}$. The relation between $\varrho^{(d)}_{L,n}$ and $\rho^{(d)}_{L,n}$ follows,

$$\varrho^{(d)}_{2,n} = \sqrt{\frac{2}{3}}\rho^{(d)}_{2,n} \tag{14.18}$$

Table 14.4: A list of scaled spatial orientation tensors in the principal axis system of the dipolar-coupling tensor, $\varsigma^{(k)}_{L,n}$ from Eq. (14.5), of rank L resulting from the Mth order perturbation expansion of the dipolar-coupling Hamiltonian is presented.

Order, M	Rank, L	$\varsigma^{(k)}_{L,n} = \omega_k \varrho^{(k)}_{L,n}$
1	2	$\varsigma^{(d)}_{2,0} = 2\omega_d$, $\varsigma^{(d)}_{2,\pm1} = 0$, $\varsigma^{(d)}_{2,\pm2} = 0$

14.3 Spin transition functions, $\xi_L^{(k)}(i,j)$

The spin transition function is typically manipulated via the coupling of the nuclear magnetic dipole moment with the oscillating external magnetic field from the applied radio-frequency pulse. Considering the strength of the external magnetic rf field is orders of magnitude larger than the internal spin-couplings, the manipulation of spin transition functions are described using the orthogonal rotation subgroups.

14.3.1 Single nucleus spin transition functions

Table 14.5: A list of single nucleus spin transition functions, $\xi_L^{(k)}(i,j)$.

$\xi_L^{(k)}(i,j)$	Rank, L	Value	Description
$\mathbb{s}(i,j)$	0	0	$\left\langle j\lvert\hat{T}_{00}\rvert j\right\rangle - \left\langle i\lvert\hat{T}_{00}\rvert i\right\rangle$
$\mathbb{p}(i,j)$	1	$j-i$	$\left\langle j\lvert\hat{T}_{10}\rvert j\right\rangle - \left\langle i\lvert\hat{T}_{10}\rvert i\right\rangle$
$\mathbb{d}(i,j)$	2	$\sqrt{\frac{3}{2}}\left(j^2-i^2\right)$	$\left\langle j\lvert\hat{T}_{20}\rvert j\right\rangle - \left\langle i\lvert\hat{T}_{20}\rvert i\right\rangle$
$\mathbb{f}(i,j)$	3	$\frac{1}{\sqrt{10}}[5(j^3-i^3)+(1-3I(I+1))(j-i)]$	$\left\langle j\lvert\hat{T}_{30}\rvert j\right\rangle - \left\langle i\lvert\hat{T}_{30}\rvert i\right\rangle$

Here, $\hat{T}_{L,k}(\mathbf{I})$ are the irreducible spherical tensor operators of rank L, $k \in [-L, L]$, for transition $|i\rangle \rightarrow |j\rangle$. In terms of the tensor product of the Cartesian operators, the above spherical tensors are expressed as follows,

Spherical tensor operator	Representation in Cartesian operators
$\hat{T}_{0,0}(\mathbf{I})$	$\hat{1}$
$\hat{T}_{1,0}(\mathbf{I})$	$\hat{I}_z$
$\hat{T}_{2,0}(\mathbf{I})$	$\frac{1}{\sqrt{6}}\left[3\hat{I}_z^2 - I(I+1)\hat{1}\right]$
$\hat{T}_{3,0}(\mathbf{I})$	$\frac{1}{\sqrt{10}}\left[5\hat{I}_z^3 + (1-3I(I+1))\,\hat{I}_z\right]$

where I is the spin quantum number of the nucleus and $\hat{\mathbf{1}}$ is the identity operator.

Table 14.6: A list of composite single nucleus spin transition functions, $\xi_L^{(k)}(i,j)$. Here, I is the spin quantum number of the nucleus.

$\xi_L^{(k)}(i,j)$	Value
$\mathbb{c}_0(i,j)$	$\frac{4}{\sqrt{125}}\left[I(I+1)-\frac{3}{4}\right]\mathbb{p}(i,j) + \sqrt{\frac{18}{25}}\mathbb{f}(i,j)$
$\mathbb{c}_2(i,j)$	$\sqrt{\frac{2}{175}}\left[I(I+1)-\frac{3}{4}\right]\mathbb{p}(i,j) - \frac{6}{\sqrt{35}}\mathbb{f}(i,j)$
$\mathbb{c}_4(i,j)$	$-\sqrt{\frac{18}{875}}\left[I(I+1)-\frac{3}{4}\right]\mathbb{p}(i,j) - \frac{17}{\sqrt{175}}\mathbb{f}(i,j)$

14.3.2 Weakly coupled nucleus spin transition functions

Table 14.7: A list of weakly coupled nucleus spin transition functions, $\xi_L^{(k)}(m_{f_I}, m_{f_S}, m_{i_I}, m_{i_S})$.

$\xi_L^{(k)}(m_{f_I}, m_{f_S}, m_{i_I}, m_{i_S})$	Value	Description
$\mathbb{d}_{IS}(m_{f_I}, m_{f_S}, m_{i_I}, m_{i_S})$	$m_{f_I} m_{f_S} - m_{i_I} m_{i_S}$	$\left\langle m_{f_I} m_{f_S} \middle\vert \hat{T}_{10}(I)\hat{T}_{10}(S) \middle\vert m_{f_I} m_{f_S} \right\rangle$ – $\left\langle m_{i_I} m_{i_S} \middle\vert \hat{T}_{10}(I)\hat{T}_{10}(S) \middle\vert m_{i_I} m_{i_S} \right\rangle$

Here, $\hat{T}_{L,k}(\mathbf{I})$ are the irreducible spherical tensor operators of rank L, $k \in [-L, L]$, for transition $|m_{i_I} m_{i_S}\rangle \rightarrow |m_{f_I} m_{f_S}\rangle$ in weakly coupled basis.

14.4 Frequency tensor components (FT) in PAS, $\varpi_{L,n}^{(k)}$

Table 14.8: The table presents a list of frequency tensors defined in the principal axis system of the respective interaction tensor from Eq. (14.7), $\varpi_{L,n}^{(k)}$, of rank L resulting from the Mth order perturbation expansion of the interaction Hamiltonians supported in `mrsimulator`.

Interaction	Order, M	Rank, L	$\varpi_{L,n}^{(k)}$
Nuclear shielding	1	0	$\varpi_{0,0}^{(\sigma)} = \varsigma_{0,0}^{(\sigma)} \quad \mathbb{p}(i,j)$
Nuclear shielding	1	2	$\varpi_{2,n}^{(\sigma)} = \varsigma_{2,n}^{(\sigma)} \quad \mathbb{p}(i,j)$
Electric Quadrupole	1	2	$\varpi_{2,n}^{(q)} = \varsigma_{2,n}^{(q)} \quad \mathbb{d}(i,j)$
Electric Quadrupole	2	0	$\varpi_{0,0}^{(qq)} = \varsigma_{0,0}^{(qq)} \quad \mathbb{c}_0(i,j)$
Electric Quadrupole	2	2	$\varpi_{2,n}^{(qq)} = \varsigma_{2,n}^{(qq)} \quad \mathbb{c}_2(i,j)$
Electric Quadrupole	2	4	$\varpi_{4,n}^{(qq)} = \varsigma_{4,n}^{(qq)} \quad \mathbb{c}_4(i,j)$
Weak J-coupling	1	0	$\varpi_{0,0}^{(J)} = \varsigma_{0,0}^{(J)} \quad \mathbb{d}_{IS}(m_{f_I}, m_{f_S}, m_{i_I}, m_{i_S})$
Weak J-coupling	1	2	$\varpi_{2,n}^{(J)} = \varsigma_{2,n}^{(J)} \quad \mathbb{d}_{IS}(m_{f_I}, m_{f_S}, m_{i_I}, m_{i_S})$
Weak dipolar-coupling	1	2	$\varpi_{2,n}^{(d)} = \varsigma_{2,n}^{(d)} \quad \mathbb{d}_{IS}(m_{f_I}, m_{f_S}, m_{i_I}, m_{i_S})$

References

CHAPTER

FIFTEEN

MODELS

15.1 Czjzek distribution

A Czjzek distribution model[1] is a random distribution of the second-rank traceless symmetric tensors about a zero tensor. An explicit form of a traceless symmetric second-rank tensor, $\mathbf{S}$, in Cartesian basis, follows,

$$\mathbf{S} = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{xy} & S_{yy} & S_{yz} \\ S_{xz} & S_{yz} & S_{zz} \end{bmatrix}, \tag{15.1}$$

where $S_{xx} + S_{yy} + S_{zz} = 0$. The elements of the above Cartesian tensor, S_{ij}, can be decomposed into second-rank irreducible spherical tensor components[3], $R_{2,k}$, following

$$\begin{aligned} S_{xx} &= \frac{1}{2}(R_{2,2} + R_{2,-2}) - \frac{1}{\sqrt{6}}R_{2,0}, \\ S_{xy} &= S_{yx} = -\frac{i}{2}(R_{2,2} - R_{2,-2}), \\ S_{yy} &= -\frac{1}{2}(R_{2,2} + R_{2,-2}) - \frac{1}{\sqrt{6}}R_{2,0}, \\ S_{xz} &= S_{zx} = -\frac{1}{2}(R_{2,1} - R_{2,-1}), \\ S_{zz} &= \sqrt{\frac{2}{3}}R_{20}, \\ S_{yz} &= S_{zy} = \frac{i}{2}(R_{2,1} + R_{2,-1}). \end{aligned} \tag{15.2}$$

In the Czjzek model, the distribution of the second-rank traceless symmetric tensor is based on the assumption of a random distribution of the five irreducible spherical tensor components, $R_{2,k}$, drawn from an uncorrelated five-dimensional multivariate normal distribution. Since $R_{2,k}$ components are complex, random sampling is performed on

[1] Czjzek, G., Fink, J., Götz, F., Schmidt, H., Coey, J. M. D., Atomic coordination and the distribution of electric field gradients in amorphous solids Phys. Rev. B (1981) **23** 2513-30. DOI: 10.1103/PhysRevB.23.2513

[3] Grandinetti, P. J., Ash, J. T., Trease, N. M. Symmetry pathways in solid-state NMR, PNMRS 2011 **59**, *2*, 121-196. DOI: 10.1016/j.pnmrs.2010.11.003

the equivalent real tensor components, which are a linear combination of $R_{2,k}$, and are given as

$$\begin{aligned} U_1 &= \frac{1}{\sqrt{6}} R_{2,0}, \\ U_2 &= -\frac{1}{\sqrt{12}} (R_{2,1} - R_{2,-1}), \\ U_3 &= \frac{i}{\sqrt{12}} (R_{2,1} + R_{2,-1}), \\ U_4 &= -\frac{i}{\sqrt{12}} (R_{2,2} - R_{2,-2}), \\ U_5 &= \frac{1}{\sqrt{12}} (R_{2,2} + R_{2,-2}), \end{aligned} \tag{15.3}$$

where U_i forms an ortho-normal basis. The components, U_i, are drawn from a five-dimensional uncorrelated multivariate normal distribution with zero mean and covariance matrix, $\Lambda = \sigma^2 \mathbf{I}_5$, where $\mathbf{I}_5$ is a 5×5 identity matrix and σ is the standard deviation.

In terms of U_i, the traceless second-rank symmetric Cartesian tensor elements, S_{ij}, follows

$$\begin{aligned} S_{xx} &= \sqrt{3} U_5 - U_1, \\ S_{xy} &= S_{yx} = \sqrt{3} U_4, \\ S_{yy} &= -\sqrt{3} U_5 - U_1, \\ S_{xz} &= S_{zx} = \sqrt{3} U_2, \\ S_{zz} &= 2 U_1, \\ S_{yz} &= S_{zy} = \sqrt{3} U_3, \end{aligned} \tag{15.4}$$

and the explicit matrix form of $\mathbf{S}$ is

$$\mathbf{S} = \begin{bmatrix} \sqrt{3} U_5 - U_1 & \sqrt{3} U_4 & \sqrt{3} U_2 \\ \sqrt{3} U_4 & -\sqrt{3} U_5 - U_1 & \sqrt{3} U_3 \\ \sqrt{3} U_2 & \sqrt{3} U_3 & 2 U_1 \end{bmatrix}. \tag{15.5}$$

In a shorthand notation, we denote a Czjzek distribution of second-rank traceless symmetric tensor as $S_C(\sigma)$.

15.2 Extended Czjzek distribution

An Extended Czjzek distribution model[2] is a random perturbation of the second-rank traceless symmetric tensors about a non-zero tensor, which is given as

$$S_T = S(0) + \rho S_C(\sigma = 1), \tag{15.6}$$

where S_T is the total tensor, $S(0)$ is the non-zero dominant second-rank tensor, $S_C(\sigma = 1)$ is the Czjzek random model attributing to the random perturbation of the tensor about the dominant tensor, $S(0)$, and ρ is the size of the perturbation. Note, in the above equation, the σ parameter from the Czjzek random model, S_C, has no meaning and

[2] Caër, G.L., Bureau, B., Massiot, D., An extension of the Czjzek model for the distributions of electric field gradients in disordered solids and an application to NMR spectra of 71Ga in chalcogenide glasses. Journal of Physics: Condensed Matter, (2010), **22**. DOI: 10.1088/0953-8984/22/6/065402

is set to one. The factor, ρ, is defined as

$$\rho = \frac{||S(0)||\epsilon}{\sqrt{30}}, \tag{15.7}$$

where $\|S(0)\|$ is the 2-norm of the dominant tensor, and ϵ is a fraction.

Part VI

API and references

CHAPTER

SIXTEEN

SIMULATION API

16.1 Simulator

class mrsimulator.Simulator(*, *name: str = None*, *label: str = None*, *description: str = None*, *spin_systems: List[mrsimulator.spin_system.SpinSystem (page 297)] = []*, *methods: List[mrsimulator.method.Method (page 310)] = []*, *config: mrsimulator.simulator.config.ConfigSimulator (page 296) = ConfigSimulator(number_of_sidebands=64, integration_volume='octant', integration_density=70, decompose_spectrum='none')*, *indexes: list = []*)

Bases: pydantic.main.BaseModel

The simulator class.

spin_systems
: A list of *SpinSystem* (page 297) or equivalent dict objects representing a collection of isolated NMR spin systems present within the sample. The default value is an empty list.

Example

```
>>> sim = Simulator()
>>> sim.spin_systems = [
...     SpinSystem(sites=[Site(isotope='17O')], abundance=0.015),
...     SpinSystem(sites=[Site(isotope='1H')], abundance=1),
... ]
```

```
>>> # or equivalently
>>> sim.spin_systems = [
...     {'sites': [{'isotope': '17O'}], 'abundance': 0.015},
...     {'sites': [{'isotope': '1H'}], 'abundance': 1},
... ]
```

Type A list of *SpinSystem* (page 297) or equivalent dict objects (optional).

methods
: A list of *Method* (page 310) or equivalent dict objects representing an NMR methods. The default value is an empty list.

Example

```
>>> from mrsimulator.methods import BlochDecaySpectrum
>>> from mrsimulator.methods import BlochDecayCTSpectrum
>>> sim.methods = [
...     BlochDecaySpectrum(channels=['17O'], spectral_width=50000),
...     BlochDecayCTSpectrum(channels=['17O'])
... ]
```

Type A list of *Method* (page 310) or equivalent dict objects (optional).

`config`
The *ConfigSimulator* (page 296) object is used to configure the simulation. The valid attributes of the ConfigSimulator object are

- `number_of_sidebands`,
- `integration_density`,
- `integration_volume`, and
- `decompose_spectrum`

Example

```
>>> from mrsimulator.simulator.config import ConfigSimulator
>>> sim.config = ConfigSimulator(
...     number_of_sidebands=32,
...     integration_density=64,
...     integration_volume='hemisphere',
...     decompose_spectrum='spin_system',
... )
```

```
>>> # or equivalently
>>> sim.config = {
...     'number_of_sidebands': 32,
...     'integration_density': 64,
...     'integration_volume': 'hemisphere',
...     'decompose_spectrum': 'spin_system',
... }
```

See *Configuring Simulator object* (page 43) for details.

Type *ConfigSimulator* (page 296) object or equivalent dict object (optional).

`name`
The name or id of the simulation or sample. The default value is None.

Example

```
>>> sim.name = '1H-17O'
>>> sim.name
'1H-17O'
```

Type str (optional)

`label`
The label for the simulation or sample. The default value is None.

Example

```
>>> sim.label = 'Test simulator'
>>> sim.label
'Test simulator'
```

Type str (optional)

`description`
A description of the simulation or sample. The default value is None.

Example

```
>>> sim.description = 'Simulation for sample 1'
>>> sim.description
'Simulation for sample 1'
```

Type str (optional)

Method Documentation

`classmethod parse_dict_with_units(`*py_dict:* dict *(page 340)*`)`
Parse the physical quantity from a dictionary representation of the Simulator object, where the physical quantity is expressed as a string with a number and a unit.

Parameters `py_dict` (*dict*) – A required python dict object.
Returns A *Simulator* (page 289) object.

Example

```
>>> sim_py_dict = {
...     'config': {
...         'decompose_spectrum': 'none',
...         'integration_density': 70,
...         'integration_volume': 'octant',
...         'number_of_sidebands': 64
...     },
```

(continues on next page)

(continued from previous page)

```
...     'spin_systems': [
...         {
...             'abundance': '100 %',
...             'sites': [{
...                 'isotope': '13C',
...                 'isotropic_chemical_shift': '20.0 ppm',
...                 'shielding_symmetric': {'eta': 0.5, 'zeta': '10.0 ppm'}
...             }]
...         },
...         {
...             'abundance': '100 %',
...             'sites': [{
...                 'isotope': '1H',
...                     'isotropic_chemical_shift': '-4.0 ppm',
...                     'shielding_symmetric': {'eta': 0.1, 'zeta': '2.1 ppm'}
...             }]
...         },
...         {
...             'abundance': '100 %',
...             'sites': [{
...                 'isotope': '27Al',
...                 'isotropic_chemical_shift': '120.0 ppm',
...                 'shielding_symmetric': {'eta': 0.1, 'zeta': '2.1 ppm'}
...             }]
...         }
...     ]
... }
>>> sim = Simulator.parse_dict_with_units(sim_py_dict)
>>> len(sim.spin_systems)
3
```

json(*include_methods: bool = False*, *include_version: bool = False*)

Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a value and a unit.

Parameters

- include_methods (*bool*) – If True, the output dictionary will include the serialized method objects. The default value is False.
- include_version (*bool*) – If True, add a version key-value pair to the serialized output dictionary. The default is False.

Returns A Dict object.

Example

```
>>> pprint(sim.json())
{'config': {'decompose_spectrum': 'none',
            'integration_density': 70,
            'integration_volume': 'octant',
            'number_of_sidebands': 64},
 'spin_systems': [{'abundance': '100.0 %',
                   'sites': [{'isotope': '13C',
```

(continues on next page)

(continued from previous page)

```
                            'isotropic_chemical_shift': '20.0 ppm',
                            'shielding_symmetric': {'eta': 0.5,
                                                    'zeta': '10.0 ppm'}}]},
                  {'abundance': '100.0 %',
                   'sites': [{'isotope': '1H',
                            'isotropic_chemical_shift': '-4.0 ppm',
                            'shielding_symmetric': {'eta': 0.1,
                                                    'zeta': '2.1 ppm'}}]},
                  {'abundance': '100.0 %',
                   'sites': [{'isotope': '27Al',
                            'isotropic_chemical_shift': '120.0 ppm',
                            'shielding_symmetric': {'eta': 0.1,
                                                    'zeta': '2.1 ppm'}}]}]}
```

`reduced_dict`(*exclude=['property_units', 'indexes']*) → *dict* (page 340)

Returns a reduced dictionary representation of the class object by removing all key-value pair corresponding to keys listed in the *exclude* argument, and keys with value as None.

Parameters `exclude` (*list*) – A list of keys to exclude from the dictionary.

Return: A dict.

`get_isotopes`(*spin_I: Optional[float] = None*, *symbol: bool = False*) → list

List of unique isotopes from the sites within the list of the spin systems corresponding to spin quantum number *I*. If *I* is None, a list of all unique isotopes is returned instead.

Parameters

- `spin_I` (*float*) – An optional spin quantum number. The valid input are the multiples of 0.5.
- `symbol` (*bool*) – If true, return a list of str with isotope symbols.

Returns A list of *Isotope* (page 336) objects.

Example

```
>>> sim.get_isotopes()
[Isotope(symbol='13C'), Isotope(symbol='1H'), Isotope(symbol='27Al')]
>>> sim.get_isotopes(symbol=True)
['13C', '1H', '27Al']
```

```
>>> sim.get_isotopes(spin_I=0.5)
[Isotope(symbol='13C'), Isotope(symbol='1H')]
>>> sim.get_isotopes(spin_I=0.5, symbol=True)
['13C', '1H']
```

```
>>> sim.get_isotopes(spin_I=1.5)
[]
```

```
>>> sim.get_isotopes(spin_I=2.5)
[Isotope(symbol='27Al')]
>>> sim.get_isotopes(spin_I=2.5, symbol=True)
['27Al']
```

load_spin_systems(*filename: str*)
Load a list of spin systems from the given JSON serialized file.

See an example of a JSON serialized file. For details, refer to the *mrsimulator I/O* (page 51) section of this documentation.

Parameters filename (*str*) – A local or remote address to a JSON serialized file.

Example

```
>>> sim.load_spin_systems(filename)
```

export_spin_systems(*filename: str*)
Export a list of spin systems to a JSON serialized file.

See an example of a JSON serialized file. For details, refer to the *mrsimulator I/O* (page 51) section.

Parameters filename (*str*) – A filename of the serialized file.

Example

```
>>> sim.export_spin_systems(filename)
```

load_methods(*filename: str*)
Load a list of methods from the given JSON serialized file.

Parameters filename (*str*) – A local or remote address to a JSON serialized file.

Example

```
>>> sim.load_methods(filename)
```

export_methods(*filename: str*)
Export a list of methods to a JSON serialized file.

Parameters filename (*str*) – A filename of the serialized file.

Example

```
>>> sim.export_methods(filename)
```

run(*method_index: Optional[list] = None*, *n_jobs: int = 1*, *pack_as_csdm: bool = True*, ***kwargs*)
Run the simulation and compute spectrum.

Parameters

- method_index – An integer or a list of integers. If provided, only the simulations corresponding to the methods at the given index/indexes will be computed. The default is None, *i.e.*, the simulation for all method will be computed.
- pack_as_csdm (*bool*) – If true, the simulation results are stored as a CSDM object, otherwise, as a ndarray object. The simulations are stored as the value of the *simulation* (page 311) attribute of the corresponding method.

Example

```
>>> sim.run()
```

save(*filename: str*, *with_units: bool = True*)
Serialize the simulator object to a JSON file.

Parameters
- **with_units** (*bool*) – If true, the attribute values are serialized as physical quantities expressed as a string with a value and a unit. If false, the attribute values are serialized as floats.
- **filename** (*str*) – The filename of the serialized file.

Example

```
>>> sim.save('filename')
```

classmethod load(*filename: str*, *parse_units: bool = True*)
Load the *Simulator* (page 289) object from a JSON file by parsing.

Parameters
- **parse_units** (*bool*) – If true, parse the attribute values from the serialized file for physical quantities, expressed as a string with a value and a unit.
- **filename** (*str*) – The filename of a JSON serialized mrsimulator file.

Returns A *Simulator* (page 289) object.

Example

```
>>> sim_1 = sim.load('filename')
```

See also:

mrsimulator I/O (page 51)

sites()
Unique sites within the Simulator object as a list of Site objects.

Returns A *Sites* (page 333) object.

Example

```
>>> sites = sim.sites()
```

16.2 ConfigSimulator

class mrsimulator.simulator.ConfigSimulator(*, *number_of_sidebands: mrsimulator.simulator.config.ConstrainedIntValue = 64*, *integration_volume: Literal['octant', 'hemisphere'] = 'octant'*, *integration_density: mrsimulator.simulator.config.ConstrainedIntValue = 70*, *decompose_spectrum: Literal['none', 'spin_system'] = 'none'*)

Bases: pydantic.main.BaseModel

The configurable attributes for the Simulator class used in simulation.

number_of_sidebands
: The value is the requested number of sidebands that will be computed in the simulation. The value cannot be zero or negative. The default value is 64.

  Type int (optional)

integration_volume
: The value is the volume over which the solid-state spectral frequency integration is performed. The valid literals of this enumeration are

  - octant (default), and
  - hemisphere

  Type enum (optional)

integration_density
: The value represents the integration density or equivalently the number of orientations over which the frequency integration is performed within a given volume. If n is the integration_density, then the total number of orientation is given as

$$n_{\text{octants}} \frac{(n+1)(n+2)}{2}, \tag{16.1}$$

  where n_{octants} is the number of octants in the given volume. The default value is 70.

  Type int (optional)

decompose_spectrum
: The value specifies how a simulation result is decomposed into an array of spectra. The valid literals of this enumeration are

  - none (default): When the value is *none*, the resulting simulation is a single spectrum, which is an integration of the spectra over all spin systems.
  - spin_system: When the value is *spin_system*, the resulting simulation is an array of spectra, where each spectrum arises from a spin system within the Simulator object.

  Type enum (optional)

Example

```
>>> a = Simulator()
>>> a.config.number_of_sidebands = 128
>>> a.config.integration_density = 96
>>> a.config.integration_volume = 'hemisphere'
>>> a.config.decompose_spectrum = 'spin_system'
```

Method Documentation

`get_orientations_count()`
Return the total number of orientations.

Example

```
>>> a = Simulator()
>>> a.config.integration_density = 20
>>> a.config.integration_volume = 'hemisphere'
>>> a.config.get_orientations_count() # (4 * 21 * 22 / 2) = 924
924
```

16.3 SpinSystem

`class mrsimulator.SpinSystem(*, property_units: Dict = {'abundance': 'pct'}, name: str = None, label: str = None, description: str = None, sites: Union[List[mrsimulator.spin_system.site.Site (page 302)], numpy.ndarray] = [], couplings: Union[List[mrsimulator.spin_system.coupling.Coupling (page 306)], numpy.ndarray] = None, abundance: mrsimulator.spin_system.ConstrainedFloatValue = 100.0, transition_pathways: List = None)`
Bases: `mrsimulator.utils.parseable.Parseable`

Base class representing an isolated spin system containing multiple sites and couplings amongst them.

Attribute Documentation

`sites`
A list of *Site* (page 302) or equivalent dict objects within the spin system. Each site object represents single-site nuclear spin interaction (nuclear shielding and EFG) tensor parameters. The default value is an empty list.

Example

```
>>> sys1 = SpinSystem()
>>> sys1.sites = [Site(isotope="17O"), Site(isotope="1H")]
>>> # or equivalently
>>> sys1.sites = [{"isotope": "17O"}, {"isotope": "1H"}]
```

Type list of *Site* (page 302) or equivalent dict objects (optional)

`couplings`
A list of *Coupling* (page 306) or equivalent dict objects within the spin system. Each coupling object represents two-site spin interaction (J-coupling and Dipolar) tensor parameters. The default value is an empty list.

Example

```
>>> sys1 = SpinSystem()
>>> sys1.couplings = [
...     Coupling(site_index=[0, 1], isotropic_j=10.1),
...     Coupling(site_index=[2, 1], dipolar={"D": 1500})
... ]
>>> # or equivalently
>>> sys1.couplings = [
...     {"site_index": [0, 1], "isotropic_j": 10.1},
...     {"site_index": [2, 1], "dipolar": {"D": 1500}}
... ]
```

Type list of *Coupling* (page 306) or equivalent dict objects (optional)

`abundance`
The abundance of the spin system in units of %. The default value is 100. The value of this attribute is useful when multiple spin systems are present.

Example

```
>>> sys1.abundance = 10
```

Type float (optional)

`name`
The value is the name or id of the spin system. The default value is None.

Example

```
>>> sys1.name = "1H-17O-0"
>>> print(sys1.name)
1H-17O-0
```

Type str (optional)

`label`
The value is a label for the spin system. The default value is None.

Example

```
>>> sys1.label = "Heteronuclear spin system"
>>> print(sys1.label)
Heteronuclear spin system
```

Type str (optional)

`description`
The value is a description of the spin system. The default value is None.

Example

```
>>> sys1.description = "A test for the spin system"
>>> print(sys1.description)
A test for the spin system
```

Type str (optional)

`transition_pathways`
A list of *TransitionPathway* (page 339) or equivalent dict objects. Each transition pathway is a list of *Transition* (page 337) objects. The resulting spectrum is a sum of the resonances arising from individual transition pathways. The default value is None.

Example

```
>>> sys1.transition_pathways = [
...     [
...         {"initial": [-2.5, 0.5], "final": [2.5, 0.5]},
...         {"initial": [0.5, 0.5], "final": [-0.5, 0.5]}
...     ]
... ]
>>> print(sys1.transition_pathways)
[|2.5, 0.5⟩⟨-2.5, 0.5| ⟶ |-0.5, 0.5⟩⟨0.5, 0.5|]
```

Note: From any given spin system, the list of relevant transition pathways is determined by the NMR method. For example, consider a single site I=3/2 spin system. For this system, a Bloch decay spectrum

method will select three transition pathways, one corresponding to the central and two to the satellite transitions. On the other hand, a Bloch decay central transition selective method will only select one transition pathway, corresponding to the central transition.

Since the spin system is independent of the NMR method, the value of this attribute is, therefore, transient. You may use this attribute to override the default transition pathway query selection criterion of the NMR method objects.

Only use this attribute if you know what you are doing.

At times, this attribute may provide a significant improvement in the performance, especially in iterative algorithms, such as the least-squares algorithm, where a one-time transition pathway query is sufficient. Repeated queries for the transition pathways will add significant overhead to the computation.

See also:

Fitting example

Type list of *TransitionPathway* (page 339) (optional).

Method Documentation

`get_isotopes`(*spin_I: Optional[float] = None*, *symbol: bool = False*) → list

An ordered list of *Isotope* (page 336) objects from the sites within the spin system corresponding to the given value of spin quantum number *I*. If *I* is None, a list of all Isotope objects is returned instead.

Parameters

- `spin_I` (*float*) – An optional spin quantum number. The valid inputs are the multiples of 0.5.
- `symbol` (*bool*) – If true, return a list of str with isotope symbols.

Returns A list of *Isotope* (page 336) objects.

Example

```
>>> spin_systems.get_isotopes() # three spin systems
[Isotope(symbol='13C'), Isotope(symbol='1H'), Isotope(symbol='27Al')]
>>> spin_systems.get_isotopes(symbol=True) # three spin systems
['13C', '1H', '27Al']
```

```
>>> spin_systems.get_isotopes(spin_I=0.5) # isotopes with I=0.5
[Isotope(symbol='13C'), Isotope(symbol='1H')]
>>> spin_systems.get_isotopes(spin_I=0.5, symbol=True) # isotopes with I=0.5
['13C', '1H']
```

```
>>> spin_systems.get_isotopes(spin_I=1.5) # isotopes with I=1.5
[]
```

```
>>> spin_systems.get_isotopes(spin_I=2.5) # isotopes with I=2.5
[Isotope(symbol='27Al')]
>>> spin_systems.get_isotopes(spin_I=2.5, symbol=True) # isotopes with I=2.5
['27Al']
```

`zeeman_energy_states()` → list

Return a list of all *ZeemanState* (page 333) objects of the spin system, where the energy states are represented by a list of quantum numbers,

$$|\Psi\rangle = [m_1, m_2, ..m_n], \tag{16.2}$$

where m_i is the quantum number associated with the i^{th} site within the spin system, and Ψ is the energy state.

Example

```
>>> spin_system_1H_13C.zeeman_energy_states()  # four energy level system.
[|-0.5, -0.5⟩, |-0.5, 0.5⟩, |0.5, -0.5⟩, |0.5, 0.5⟩]
```

Returns A list of *ZeemanState* (page 333) objects.

`all_transitions()` → mrsimulator.transition.pathway.TransitionList

Returns a list of all possible spin *Transition* (page 337) objects in the given spin system.

Example

```
>>> spin_system_1H_13C.all_transitions()  # 16 two energy level transitions
[|-0.5, -0.5⟩⟨-0.5, -0.5|,
|-0.5, 0.5⟩⟨-0.5, -0.5|,
|0.5, -0.5⟩⟨-0.5, -0.5|,
|0.5, 0.5⟩⟨-0.5, -0.5|,
|-0.5, -0.5⟩⟨-0.5, 0.5|,
|-0.5, 0.5⟩⟨-0.5, 0.5|,
|0.5, -0.5⟩⟨-0.5, 0.5|,
|0.5, 0.5⟩⟨-0.5, 0.5|,
|-0.5, -0.5⟩⟨0.5, -0.5|,
|-0.5, 0.5⟩⟨0.5, -0.5|,
|0.5, -0.5⟩⟨0.5, -0.5|,
|0.5, 0.5⟩⟨0.5, -0.5|,
|-0.5, -0.5⟩⟨0.5, 0.5|,
|-0.5, 0.5⟩⟨0.5, 0.5|,
|0.5, -0.5⟩⟨0.5, 0.5|,
|0.5, 0.5⟩⟨0.5, 0.5|]
```

Returns A list of *Transition* (page 337) objects.

`classmethod parse_dict_with_units(`*py_dict:* dict *(page 340)*`)`

Parse physical quantities from a dictionary representation of the SpinSystem object, where the physical quantity is expressed as a string with a number and a unit.

Parameters `py_dict` (*dict*) – A required python dict object.
Returns *SpinSystem* (page 297) object.

Example

```
>>> spin_system_dict = {
...     "sites": [{
...         "isotope":"13C",
...         "isotropic_chemical_shift": "20 ppm",
...         "shielding_symmetric": {
...             "zeta": "10 ppm",
...             "eta": 0.5
...         }
...     }]
... }
>>> spin_system_1 = SpinSystem.parse_dict_with_units(spin_system_dict)
```

`json()` → *dict* (page 340)
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a number and a unit.

```
>>> pprint(spin_system_1.json())
{'abundance': '100.0 %',
 'sites': [{'isotope': '13C',
            'isotropic_chemical_shift': '20.0 ppm',
            'shielding_symmetric': {'eta': 0.5, 'zeta': '10.0 ppm'}}]}
```

16.4 Site

`class mrsimulator.Site(*, property_units: Dict = {'isotropic_chemical_shift': 'ppm'}, name: str = None, label: str = None, description: str = None, isotope: str = '1H', isotropic_chemical_shift: float = 0.0, shielding_symmetric: mrsimulator.spin_system.tensors.SymmetricTensor (page 333) = None, shielding_antisymmetric: mrsimulator.spin_system.tensors.AntisymmetricTensor (page 335) = None, quadrupolar: mrsimulator.spin_system.tensors.SymmetricTensor (page 333) = None)`

Bases: `mrsimulator.utils.parseable.Parseable`

Base class representing a single-site nuclear spin interaction tensor parameters. The single-site nuclear spin interaction tensors include the nuclear shielding and the electric quadrupolar tensor.

Attribute Documentation

`isotope`
A string expressed as an atomic number followed by an isotope symbol, eg., *'13C'*, *'17O'*. The default value is *'1H'*.

Example

```
>>> site = Site(isotope='2H')
```

Type str (optional)

`isotropic_chemical_shift`
The isotropic chemical shift of the site in ppm. The default value is 0.

Example

```
>>> site.isotropic_chemical_shift = 43.3
```

Type float (optional)

`shielding_symmetric`
The attribute represents the parameters of the irreducible second-rank traceless symmetric part of the nuclear shielding tensor. The default value is None.

The allowed attributes of the *SymmetricTensor* (page 333) class for *shielding_symmetric* are `zeta`, `eta`, `alpha`, `beta`, and `gamma`, where `zeta` is the shielding anisotropy, in ppm, and `eta` is the shielding asymmetry parameter defined using the Haeberlen convention. The Euler angles `alpha`, `beta`, and `gamma` are in radians.

Example

```
>>> site.shielding_symmetric = {'zeta': 10, 'eta': 0.5}
```

```
>>> # or equivalently
>>> site.shielding_symmetric = SymmetricTensor(zeta=10, eta=0.5)
```

Type *SymmetricTensor* (page 333) or equivalent dict object (optional).

`shielding_antisymmetric`
The attribute represents the parameters of the irreducible first-rank antisymmetric part of the nuclear shielding tensor. The default value is None.

The allowed attributes of the *AntisymmetricTensor* (page 335) class for *shielding_antisymmetric* are `zeta`, `alpha`, and `beta`, where `zeta` is the anisotropy parameter, in ppm, of the anti-symmetric first-rank tensor. The angles `alpha` and `beta` are in radians.

Example

```
>>> site.shielding_antisymmetric = {'zeta': 20}
```

```
>>> # or equivalently
>>> site.shielding_antisymmetric = AntisymmetricTensor(zeta=20)
```

Type *AntisymmetricTensor* (page 335) or equivalent dict object (optional).

`quadrupolar`
The attribute represents the parameters of the traceless irreducible second-rank symmetric part of the electric-field gradient tensor. The default value is None.

The allowed attributes of the *SymmetricTensor* (page 333) class for *quadrupolar* are `Cq`, `eta`, `alpha`, `beta`, and `gamma`, where `Cq` is the quadrupolar coupling constant, in Hz, and `eta` is the quadrupolar asymmetry parameter. The Euler angles `alpha`, `beta`, and `gamma` are in radians.

Example

```
>>> site.quadrupolar = {'Cq': 3.2e6, 'eta': 0.52}
```

```
>>> # or equivalently
>>> site.quadrupolar = SymmetricTensor(Cq=3.2e6, eta=0.52)
```

Type *SymmetricTensor* (page 333) or equivalent dict object (optional).

`name`
The name or id of the site. The default value is None.

Example

```
>>> site.name = '2H-0'
>>> site.name
'2H-0'
```

Type str (optional)

`label`
The label for the site. The default value is None.

Example

```
>>> site.label = 'Quad site'
>>> site.label
'Quad site'
```

Type str (optional)

`description`
A description of the site. The default value is None.

Example

```
>>> site.description = 'An example Quadrupolar site.'
>>> site.description
'An example Quadrupolar site.'
```

Type str (optional)

Example

The following are a few examples of the site object.

```
>>> site1 = Site(
...     isotope='33S',
...     isotropic_chemical_shift=20, # in ppm
...     shielding_symmetric={
...         "zeta": 10, # in ppm
...         "eta": 0.5
...     },
...     quadrupolar={
...         "Cq": 5.1e6, # in Hz
...         "eta": 0.5
...     }
... )
```

Using SymmetricTensor objects.

```
>>> site1 = Site(
...     isotope='13C',
...     isotropic_chemical_shift=20, # in ppm
...     shielding_symmetric=SymmetricTensor(zeta=10, eta=0.5),
... )
```

Method Documentation

classmethod parse_dict_with_units(*py_dict:* dict *(page 340)*)
Parse the physical quantity from a dictionary representation of the Site object, where the physical quantity is expressed as a string with a number and a unit.

Parameters py_dict (*dict*) – A required python dict object.
Returns *Site* (page 302) object.

Example

```
>>> site_dict = {
...     "isotope": "13C",
...     "isotropic_chemical_shift": "20 ppm",
...     "shielding_symmetric": {"zeta": "10 ppm", "eta":0.5}
... }
>>> site1 = Site.parse_dict_with_units(site_dict)
```

json() → *dict* (page 340)
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a number and a unit.

```
>>> pprint(site1.json())
{'isotope': '13C',
 'isotropic_chemical_shift': '20.0 ppm',
 'shielding_symmetric': {'eta': 0.5, 'zeta': '10.0 ppm'}}
```

16.5 Coupling

class mrsimulator.Coupling(*, *property_units: Dict = {'isotropic_j': 'Hz'}*, *name: str = None*, *label: str = None*, *description: str = None*, *site_index: List[int]*, *isotropic_j: float = 0.0*, *j_symmetric:* mrsimulator.spin_system.tensors.SymmetricTensor *(page 333) = None*, *j_antisymmetric:* mrsimulator.spin_system.tensors.AntisymmetricTensor *(page 335) = None*, *dipolar:* mrsimulator.spin_system.tensors.SymmetricTensor *(page 333) = None*)
Bases: mrsimulator.utils.parseable.Parseable

Base class representing a two-site coupled nuclear spin interaction tensor parameters, which include the J-coupling and dipolar tensor.

Attribute Documentation

site_index
A list of two integers, each corresponding to the index of the coupled sites.

Example

```
>>> coupling = Coupling(site_index=[0, 1])
```

Type list of int (required)

`isotropic_j`
The isotropic j-coupling, in Hz, between the coupled sites. The default is 0.

Example

```
>>> coupling.isotropic_j = 43.3
```

Type float (optional)

`j_symmetric`
The attribute represents the parameters of the irreducible second-rank traceless symmetric part of the J-coupling tensor. The default value is None.

The allowed attributes of the *SymmetricTensor* (page 333) class for *j_symmetric* are `zeta`, `eta`, `alpha`, `beta`, and `gamma`, where `zeta` is the J anisotropy, in Hz, and `eta` is the J asymmetry parameter defined using the Haeberlen convention. The Euler angles `alpha`, `beta`, and `gamma` are in radians.

Example

```
>>> coupling.j_symmetric = {'zeta': 10, 'eta': 0.5}
```

```
>>> # or equivalently
>>> coupling.j_symmetric = SymmetricTensor(zeta=10, eta=0.5)
```

Type *SymmetricTensor* (page 333) or equivalent dict object (optional).

`j_antisymmetric`
The attribute represents the parameters of the irreducible first-rank antisymmetric part of the J tensor. The default value is None.

The allowed attributes of the *AntisymmetricTensor* (page 335) class for *j_antisymmetric* are `zeta`, `alpha`, and `beta`, where `zeta` is the anisotropy parameter of the anti-symmetric first-rank tensor given in Hz. The angles `alpha` and `beta` are in radians.

Example

```
>>> coupling.j_antisymmetric = {'zeta': 20}
```

```
>>> # or equivalently
>>> coupling.j_antisymmetric = AntisymmetricTensor(zeta=20)
```

Type *AntisymmetricTensor* (page 335) or equivalent dict object (optional).

`dipolar`
The attribute represents the parameters of the irreducible second-rank traceless symmetric part of the direct-dipolar coupling tensor. The default value is None.

The allowed attributes of the *SymmetricTensor* (page 333) class for *dipolar* are `D`, `alpha`, `beta`, and `gamma`, where `D` is the dipolar coupling constant, in Hz. The Euler angles `alpha`, `beta`, and `gamma` are in radians.

Example

```
>>> coupling.dipolar = {'D': 320}
```

```
>>> # or equivalently
>>> coupling.dipolar = SymmetricTensor(D=320)
```

Type *SymmetricTensor* (page 333) or equivalent dict object (optional).

`name`
The name or id of the coupling. The default value is None.

Example

```
>>> coupling.name = '1H-1H'
>>> coupling.name
'1H-1H'
```

Type str (optional)

`label`
The label for the coupling. The default value is None.

Example

```
>>> coupling.label = 'Weak coupling'
>>> coupling.label
'Weak coupling'
```

Type str (optional)

`description`
A description of the coupling. The default value is None.

Example

```
>>> coupling.description = 'An example coupled sites.'
>>> coupling.description
'An example coupled sites.'
```

Type str (optional)

Example

The following are a few examples of setting the site object.

```
>>> coupling1 = Coupling(
...     site_index=[0, 1],
...     isotropic_j=20, # in Hz
...     j_symmetric={
...         "zeta": 10, # in Hz
...         "eta": 0.5
...     },
...     dipolar={"D": 5.1e3}, # in Hz
... )
```

Using SymmetricTensor objects.

```
>>> coupling1 = Coupling(
...     site_index=[0, 1],
...     isotropic_j=20, # in Hz
...     j_symmetric=SymmetricTensor(zeta=10, eta=0.5),
...     dipolar=SymmetricTensor(D=5.1e3), # in Hz
... )
```

Method Documentation

`classmethod parse_dict_with_units(`*py_dict:* dict *(page 340)*`)`

Parse the physical quantity from a dictionary representation of the Coupling object, where the physical quantity is expressed as a string with a number and a unit.

Parameters `py_dict` (*dict*) – A required python dict object.
Returns *Site* (page 302) object.

Example

```
>>> coupling_dict = {
...     "site_index": [1, 2],
...     "isotropic_j": "20 Hz",
...     "j_symmetric": {"zeta": "10 Hz", "eta":0.5}
... }
>>> coupling1 = Coupling.parse_dict_with_units(coupling_dict)
```

`json()` → *dict* (page 340)
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a number and a unit.

```
>>> pprint(coupling1.json())
{'isotropic_j': '20.0 Hz',
 'j_symmetric': {'eta': 0.5, 'zeta': '10.0 Hz'},
 'site_index': [0, 1]}
```

16.6 Method

16.6.1 Method

`class mrsimulator.Method(`*, property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}, name: str = None, label: str = None, description: str = None, channels: List[str] = [], spectral_dimensions: List[mrsimulator.method.spectral_dimension.SpectralDimension (page 313)] = [SpectralDimension(property_units={'spectral_width': 'Hz', 'reference_offset': 'Hz', 'origin_offset': 'Hz'}, count=1024, spectral_width=25000.0, reference_offset=0.0, origin_offset=None, label=None, description=None, events=[])], affine_matrix: Union[numpy.ndarray, List] = None, simulation: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None, experiment: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None`)`

Bases: `mrsimulator.utils.parseable.Parseable`

Base Method class. A method class represents the NMR method.

`channels`
The value is a list of isotope symbols over which the given method applies. An isotope symbol is given as a string with the atomic number followed by its atomic symbol, for example, '1H', '13C', and '33S'. The default is an empty list. The number of isotopes in a *channel* depends on the method. For example, a *BlochDecaySpectrum* method is a single channel method, in which case, the value of this attribute is a list with a single isotope symbol, ['13C'].

Example

```
>>> bloch = Method()
>>> bloch.channels = ['1H']
```

Type List[str] (optional)

`spectral_dimensions`
The number of spectral dimensions depends on the given method. For example, a *BlochDecaySpectrum* method is a one-dimensional method and thus requires a single spectral dimension. The default is a single default *SpectralDimension* (page 313) object.

Example

```
>>> bloch = Method()
>>> bloch.spectral_dimensions = [SpectralDimension(count=8, spectral_width=50)]
>>> # or equivalently
>>> bloch.spectral_dimensions = [{'count': 8, 'spectral_width': 50}]
```

Type List[*SpectralDimension* (page 313)] or List[dict] (optional).

`simulation`
An object holding the result of the simulation. The initial value of this attribute is None. A value is assigned to this attribute when you run the simulation using the *run()* (page 294) method.

Type CSDM or ndarray (N/A)

`experiment`
An object holding the experimental measurement for the given method, if available. The default value is None.

Example

```
>>> bloch.experiment = my_data
```

Type CSDM or ndarray (optional)

`name`
Name or id of the method. The default value is None.

Example

```
>>> bloch.name = 'BlochDecaySpectrum'
>>> bloch.name
'BlochDecaySpectrum'
```

Type str (optional)

`label`
Label for the method. The default value is None.

Example

```
>>> bloch.label = 'One pulse acquired spectrum'
>>> bloch.label
'One pulse acquired spectrum'
```

Type str (optional)

`description`
A description of the method. The default value is None.

Example

```
>>> bloch.description = 'Huh!'
>>> bloch.description
'Huh!'
```

Type str (optional)

`affine_matrix`
A (n x n) affine transformation matrix, where n is the number of spectral_dimensions. If provided, the corresponding affine transformation is applied to the computed frequencies. The default is None, i.e., no transformation is applied.

Example

```
>>> method = Method2D()
>>> method.affine_matrix = [[1, -1], [0, 1]]
>>> print(method.affine_matrix)
[[ 1 -1]
 [ 0  1]]
```

Type np.ndarray or 2D list (optional)

Method Documentation

`classmethod parse_dict_with_units(py_dict)`
Parse the physical quantity from a dictionary representation of the Method object, where the physical quantity is expressed as a string with a number and a unit.

Parameters `py_dict` (*dict*) – A python dict representation of the Method object.
Returns A *Method* (page 310) object.

`json()` → mrsimulator.method.Method.dict
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a value and a unit.

Returns A python dict object.

`update_spectral_dimension_attributes_from_experiment()`
Update the spectral dimension attributes of the method to match the attributes of the experiment from the *experiment* (page 311) attribute.

`get_transition_pathways(spin_system)` → List[*mrsimulator.transition.pathway.TransitionPathway* (page 339)]
Return a list of transition pathways from the given spin system that satisfy the query selection criterion of the method.

Parameters `spin_system` (SpinSystem (page 297)) – A SpinSystem object.
Returns An array of *TransitionPathway* (page 339) objects. Each TransitionPathway object is an ordered collection of Transition objects.

Example

```
>>> from mrsimulator import SpinSystem
>>> from mrsimulator.methods import ThreeQ_VAS
>>> sys = SpinSystem(sites=[{'isotope': '27Al'}, {'isotope': '29Si'}])
>>> method = ThreeQ_VAS(channels=['27Al'])
>>> pprint(method.get_transition_pathways(sys))
[|1.5, -0.5⟩⟨-1.5, -0.5| ⟶ |-0.5, -0.5⟩⟨0.5, -0.5|,
 |1.5, -0.5⟩⟨-1.5, -0.5| ⟶ |-0.5, 0.5⟩⟨0.5, 0.5|,
 |1.5, 0.5⟩⟨-1.5, 0.5| ⟶ |-0.5, -0.5⟩⟨0.5, -0.5|,
 |1.5, 0.5⟩⟨-1.5, 0.5| ⟶ |-0.5, 0.5⟩⟨0.5, 0.5|]
```

`shape()` → tuple

The shape of the method's spectral dimension array.

Returns tuple

Example

```
>>> from mrsimulator.methods import Method2D
>>> method = Method2D(spectral_dimensions=[{'count': 40}, {'count': 10}])
>>> method.shape()
(40, 10)
```

16.6.2 SpectralDimension

`class mrsimulator.SpectralDimension(*, property_units: Dict = {'origin_offset': 'Hz', 'reference_offset': 'Hz', 'spectral_width': 'Hz'}, count: mrsimulator.method.spectral_dimension.ConstrainedIntValue = 1024, spectral_width: mrsimulator.method.spectral_dimension.ConstrainedFloatValue = 25000.0, reference_offset: float = 0.0, origin_offset: float = None, label: str = None, description: str = None, events: List[mrsimulator.method.event.Event (page 315)] = [])`

Bases: `mrsimulator.utils.parseable.Parseable`

Base SpectralDimension class defines a spectroscopic dimension of the method.

`count`

The number of points, N, along the spectroscopic dimension. The default value is 1024.

Type int (optional)

`spectral_width`

The spectral width, Δx, of the spectroscopic dimension in units of Hz. The default value is 25000.

Type float (optional)

`reference_offset`

The reference offset, x_0, of the spectroscopic dimension in units of Hz. The default value is 0.

Type float (optional)

`origin_offset`

The origin offset (Larmor frequency) along the spectroscopic dimension in units of Hz. The default value

is None. When the value is None, the origin offset is set to the Larmor frequency of the isotope from the *channels* (page 310) attribute of the method.

Type float (optional)

`label`
The value is a label of the spectroscopic dimension. The default value is None.

Type str (optional)

`description`
The value is a description of the spectroscopic dimension. The default value is None.

Type str (optional)

`events`
The value describes a series of events along the spectroscopic dimension.

Type A list of *Event* (page 315) or equivalent dict objects (optional).

Method Documentation

`classmethod parse_dict_with_units(`*py_dict:* dict *(page 340)*`)`
Parse the physical quantities of a SpectralDimension object from a python dictionary object.

Parameters `py_dict` (*dict*) – Dict object

`coordinates_Hz()` → numpy.ndarray
The grid coordinates along the dimension in units of Hz, evaluated as

$$x_{\text{Hz}} = ([0, 1, ...N-1] - T)\,\frac{\Delta x}{N} + x_0 \tag{16.3}$$

where $T = N/2$ and $T = (N-1)/2$ for even and odd values of N, respectively.

`coordinates_ppm()` → numpy.ndarray
The grid coordinates along the dimension as dimension frequency ratio in units of ppm. The coordinates are evaluated as

$$x_{\text{ppm}} = \frac{x_{\text{Hz}}}{x_0 + \omega_0} \tag{16.4}$$

where ω_0 is the Larmor frequency.

`json()` → *dict* (page 340)
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a number and a unit.

`to_csdm_dimension()` → csdmpy.dimension.Dimension
Return the spectral dimension as a CSDM dimension object.

16.6.3 Event

class mrsimulator.Event(*, property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}, fraction: float = 1.0, magnetic_flux_density: mrsimulator.method.event.ConstrainedFloatValue = 9.4, rotor_frequency: mrsimulator.method.event.ConstrainedFloatValue = 0.0, rotor_angle: mrsimulator.method.event.ConstrainedFloatValue = 0.955316618, freq_contrib: List[mrsimulator.method.frequency_contrib.FrequencyEnum] = [<FrequencyEnum.Shielding1_0: 'Shielding1_0'>, <FrequencyEnum.Shielding1_2: 'Shielding1_2'>, <FrequencyEnum.Quad1_2: 'Quad1_2'>, <FrequencyEnum.Quad2_0: 'Quad2_0'>, <FrequencyEnum.Quad2_2: 'Quad2_2'>, <FrequencyEnum.Quad2_4: 'Quad2_4'>], transition_query: mrsimulator.method.transition_query.TransitionQuery = TransitionQuery(P={'channel-1': [[-1.0]]}, D=None, F=None, transitions=None))

Bases: mrsimulator.utils.parseable.Parseable

Base Event class defines the spin environment and the transition query for a segment of the transition pathway.

fraction
: The weight of the frequency contribution from the event. The default is 1.

  Type float

magnetic_flux_density
: The macroscopic magnetic flux density, H_0, of the applied external magnetic field during the event in units of T. The default value is 9.4.

  Type float

rotor_frequency
: The sample spinning frequency ν_r, during the event in units of Hz. The default value is 0.

  Type float

rotor_angle
: The angle between the sample rotation axis and the applied external magnetic field vector, θ, during the event in units of rad. The default value is 0.9553166, i.e. the magic angle.

  Type float

freq_contrib
: A list of FrequencyEnum enumeration. The default is all frequency enumerations.

  Type List[*mrsimulator.method.frequency_contrib.FrequencyEnum* (page 316)]

transition_query
: A TransitionQuery or an equivalent dict object listing the queries used in selecting the active transitions during the event. Only the active transitions from this query will contribute to the net frequency.

  Type *mrsimulator.method.transition_query.TransitionQuery* (page 317)

Method Documentation

classmethod parse_dict_with_units(*py_dict:* dict *(page 340)*)
Parse the physical quantities of an Event object from a python dictionary object.

Parameters py_dict (*dict*) – Dict object

json() → *dict* (page 340)
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a value and a unit.

16.6.4 FrequencyEnum

class mrsimulator.method.frequency_contrib.FrequencyEnum(*value*)
Bases: str, enum.Enum

Enumeration for selecting specific frequency contributions. The enumerations are:

Shielding1_0
Selects first-order and zeroth-rank nuclear shielding frequency contributions.

Type str

Shielding1_2
Selects first-order and second-rank nuclear shielding frequency contributions.

Type str

Quad1_2
Selects first-order and second-rank quadrupolar frequency contributions.

Type str

Quad2_0
Selects second-order and zeroth-rank quadrupolar frequency contributions.

Type str

Quad2_2
Selects second-order and second-rank quadrupolar frequency contributions.

Type str

Quad2_4
Selects second-order and fourth-rank quadrupolar frequency contributions.

Type str

Method Documentation

json() → dict
Parse the class object to a JSON compliant python dictionary object.

index() → int
Get the index of enumeration relative to freq_list_all.

16.6.5 TransitionQuery

class mrsimulator.method.transition_query.TransitionQuery(*, *P: Dict = {'channel-1': [[- 1.0]]}*, *D: Dict = None*, *F: Dict = None*, *transitions: List[*mrsimulator.transition.base.Transition *(page 337)] = None*)

Bases: pydantic.main.BaseModel

Base TransitionQuery class.

P
: A dict of channels, with each channel as a list of p symmetry functions per site. Here p = Δm is the difference between spin quantum numbers of the final and initial states.

Example

```
>>> method = Method2D()
>>> method.spectral_dimensions[0].events[0].transition_query.P = {
...     'channel-1': [[-1]]
... }
```

Type Optional[Dict]

D
: A dict of channels, with each channel as a list of d symmetry functions per site. Here p = Δm is the difference between spin quantum numbers of the final and initial states.

Example

```
>>> method.spectral_dimensions[0].events[0].transition_query.D = {
...     'channel-1': [[0]]
... }
```

Type Optional[Dict]

Method Documentation

json() → dict
: Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a value and a unit.

16.7 Methods

The following are the list of methods currently supported by `mrsimulator` as a part of the `mrsimulator.methods` module. To import a method, for example the *BlochDecaySpectrum*, used

```
>>> from mrsimulator.methods import BlochDecaySpectrum
```

All methods categorize into two groups, generic and specialized methods. A generic method is general and is based on the number of spectral dimensions. At present, there are two generic methods, `Method1D` and `Method2D`. All specialized methods are derived from their respective generic method objects. The purpose of the specialized methods is to facilitate user ease when setting up some commonly used methods, such as the MQMAS, STMAS, PASS, MAT, etc.

16.7.1 Summary

Generic methods

Method1D (page 319)([spectral_dimensions])	A generic one-dimensional spectrum method.
Method2D (page 322)([spectral_dimensions])	A generic two-dimensional correlation spectrum method.

Specialized 1D methods

BlochDecaySpectrum (page 320)([spectral_dimensions])	A one-dimensional Bloch decay spectrum method.
BlochDecayCTSpectrum (page 321)([spectral_dimensions])	A one-dimensional central transition selective Bloch decay spectrum method.

Specialized 2D methods

ThreeQ_VAS (page 325)(*[, property_units, name, label, ...])	Simulate a sheared and scaled 3Q 2D variable-angle spinning spectrum.
FiveQ_VAS (page 326)(*[, property_units, name, label, ...])	Simulate a sheared and scaled 5Q variable-angle spinning spectrum.
SevenQ_VAS (page 327)(*[, property_units, name, label, ...])	Simulate a sheared and scaled 7Q variable-angle spinning spectrum.
ST1_VAS (page 329)(*[, property_units, name, label, ...])	Simulate a sheared and scaled inner satellite and central transition correlation spectrum.
ST2_VAS (page 330)(*[, property_units, name, label, ...])	Simulate a sheared and scaled second to inner satellite and central transition correlation spectrum.
SSB2D (page 332)(*[, property_units, name, label, ...])	Simulating a sheared 2D finite to infinite speed MAS correlation spectum.

16.7.2 Table of contents

Generic one-dimensional method

mrsimulator.methods.Method1D(*spectral_dimensions=[{}], **kwargs*)

A generic one-dimensional spectrum method.

Parameters

- `name` (*str (optional)*) – The value is the name or id of the method. The default value is None.
- `label` (*str (optional)*) – The value is a label for the method. The default value is None.
- `description` (*str (optional)*) – The value is a description of the method. The default value is None.
- `experiment` (*CSDM or ndarray (optional)*) – An object holding the experimental measurement for the given method, if available. The default value is None.
- `channels` (*list (optional)*) – The value is a list of isotope symbols over which the given method applies. An isotope symbol is given as a string with the atomic number followed by its atomic symbol, for example, '1H', '13C', and '33S'. The default is an empty list. The number of isotopes in a *channel* depends on the method. For example, a *BlochDecaySpectrum* method is a single channel method, in which case, the value of this attribute is a list with a single isotope symbol, ['13C'].
- `spectral_dimensions` (List of *SpectralDimension* (page 313) or dict objects (optional).) – The number of spectral dimensions depends on the given method. For example, a *BlochDecaySpectrum* method is a one-dimensional method and thus requires a single spectral dimension. The default is a single default *SpectralDimension* (page 313) object.
- `magetic_flux_density` (*float (optional)*) – A global value for the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default is `9.4`.
- `rotor_angle` (*float (optional)*) – A global value for the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is `0.9553166`, i.e. the magic angle.
- `rotor_frequency` (*float (optional)*) – A global value for the sample spinning frequency, ν_r, in units of Hz. The default value is `0`.

Returns

Return type A *Method* (page 310) instance.

Note: If any parameter is defined outside of the *spectral_dimensions* list, the value of those parameters is considered global. In a multi-event method, you may also assign parameter values to individual events.

Example

Example method for simulating triple-quantum 1D spectrum.

```
>>> from mrsimulator.methods import Method1D
>>> method1 = Method1D(
...     channels=["87Rb"],
...     magnetic_flux_density=7,  # in T
...     rotor_angle=54.735 * np.pi / 180,
...     rotor_frequency=1e9,
...     spectral_dimensions=[
...         {
...             "count": 1024,
...             "spectral_width": 1e4,  # in Hz
...             "reference_offset": -4e3,  # in Hz
```

(continues on next page)

(continued from previous page)

```
...             "label": "quad only",
...             "events": [{"transition_query": {"P": [-3], "D": [0]}}],
...         }
...     ],
... )
```

Bloch Decay Spectrum method

mrsimulator.methods.BlochDecaySpectrum(*spectral_ dimensions=[{}], **kwargs*)

A one-dimensional Bloch decay spectrum method.

Parameters

- `name` (*str (optional)*) – The value is the name or id of the method. The default value is None.
- `label` (*str (optional)*) – The value is a label for the method. The default value is None.
- `description` (*str (optional)*) – The value is a description of the method. The default value is None.
- `experiment` (*CSDM or ndarray (optional)*) – An object holding the experimental measurement for the given method, if available. The default value is None.
- `channels` (*list (optional)*) – The value is a list of isotope symbols over which the given method applies. An isotope symbol is given as a string with the atomic number followed by its atomic symbol, for example, '1H', '13C', and '33S'. The default is an empty list. The number of isotopes in a *channel* depends on the method. For example, a *BlochDecaySpectrum* method is a single channel method, in which case, the value of this attribute is a list with a single isotope symbol, ['13C'].
- `spectral_dimensions` (List of *SpectralDimension* (page 313) or dict objects (optional).) – The number of spectral dimensions depends on the given method. For example, a *BlochDecaySpectrum* method is a one-dimensional method and thus requires a single spectral dimension. The default is a single default *SpectralDimension* (page 313) object.
- `magetic_flux_density` (*float (optional)*) – A global value for the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default is `9.4`.
- `rotor_angle` (*float (optional)*) – A global value for the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is `0.9553166`, i.e. the magic angle.
- `rotor_frequency` (*float (optional)*) – A global value for the sample spinning frequency, ν_r, in units of Hz. The default value is `0`.

Returns

Return type A *Method* (page 310) instance.

Example

```
>>> from mrsimulator.methods import BlochDecaySpectrum
>>> Bloch_method = BlochDecaySpectrum(
...     channels=["1H"],
...     rotor_frequency=5000,  # in Hz
...     rotor_angle=0.95531,  # in rad
...     magnetic_flux_density=9.4,  # in T
...     spectral_dimensions=[
...         dict(count=1024, spectral_width=50000, reference_offset=-8000)
...     ],
... )
```

Bloch decay method is a special case of *Method1D* (page 319), given as

```
>>> from mrsimulator.methods import Method1D
>>> Blochdecay = Method1D(
...     channels=["1H"],
...     rotor_frequency=5000,  # in Hz
...     rotor_angle=0.95531,  # in rad
...     magnetic_flux_density=9.4,  # in T
...     spectral_dimensions=[
...         {
...             "count": 1024,
...             "spectral_width": 50000,  # in Hz
...             "reference_offset": -8000,  # in Hz
...             "events": [{"transition_query": {"P": [-1]}}],
...         }
...     ],
... )
```

Bloch Decay Central Transition Spectrum method

mrsimulator.methods.BlochDecayCTSpectrum(*spectral_dimensions=[{}], **kwargs*)

A one-dimensional central transition selective Bloch decay spectrum method.

Parameters

- `name` (*str (optional)*) – The value is the name or id of the method. The default value is None.
- `label` (*str (optional)*) – The value is a label for the method. The default value is None.
- `description` (*str (optional)*) – The value is a description of the method. The default value is None.
- `experiment` (*CSDM or ndarray (optional)*) – An object holding the experimental measurement for the given method, if available. The default value is None.
- `channels` (*list (optional)*) – The value is a list of isotope symbols over which the given method applies. An isotope symbol is given as a string with the atomic number followed by its atomic symbol, for example, '1H', '13C', and '33S'. The default is an empty list. The number of isotopes in a *channel* depends on the method. For example, a *BlochDecaySpectrum* method is a single channel method, in which case, the value of this attribute is a list with a single isotope symbol, ['13C'].
- `spectral_dimensions` (List of *SpectralDimension* (page 313) or dict objects (optional).) – The number of spectral dimensions depends on the given method. For example, a *BlochDecaySpectrum* method is a one-dimensional method and thus requires a single spectral dimension. The default is a single default *SpectralDimension* (page 313) object.
- `magetic_flux_density` (*float (optional)*) – A global value for the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default is `9.4`.
- `rotor_angle` (*float (optional)*) – A global value for the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is `0.9553166`, i.e. the magic angle.
- `rotor_frequency` (*float (optional)*) – A global value for the sample spinning frequency, ν_r, in units of Hz. The default value is `0`.

Returns

Return type A *Method* (page 310) instance.

Example

```
>>> from mrsimulator.methods import BlochDecayCTSpectrum
>>> Bloch_CT_method = BlochDecayCTSpectrum(
```

(continues on next page)

(continued from previous page)

```
...     channels=["1H"],
...     rotor_frequency=5000,  # in Hz
...     rotor_angle=0.95531,  # in rad
...     magnetic_flux_density=9.4,  # in T
...     spectral_dimensions=[
...         dict(count=1024, spectral_width=50000, reference_offset=-8000)
...     ],
... )
```

Bloch decay central transition selective method is a special case of *Method1D* (page 319), given as

```
>>> from mrsimulator.methods import Method1D
>>> BlochdecayCT = BlochDecayCTSpectrum(
...     channels=["1H"],
...     rotor_frequency=5000,  # in Hz
...     rotor_angle=0.95531,  # in rad
...     magnetic_flux_density=9.4,  # in T
...     spectral_dimensions=[
...         {
...             "count": 1024,
...             "spectral_width": 50000,  # in Hz
...             "reference_offset": -8000,  # in Hz
...             "events": [{"transition_query": {"P": [-1], "D": [0]}}],
...         }
...     ],
... )
```

Generic two-dimensional correlation method

`mrsimulator.methods.Method2D`(*spectral_ dimensions=[{}], **kwargs*)

A generic two-dimensional correlation spectrum method.

Parameters

- `name` (*str (optional)*) – The value is the name or id of the method. The default value is None.
- `label` (*str (optional)*) – The value is a label for the method. The default value is None.
- `description` (*str (optional)*) – The value is a description of the method. The default value is None.
- `experiment` (*CSDM or ndarray (optional)*) – An object holding the experimental measurement for the given method, if available. The default value is None.
- `channels` (*list (optional)*) – The value is a list of isotope symbols over which the given method applies. An isotope symbol is given as a string with the atomic number followed by its atomic symbol, for example, '1H', '13C', and '33S'. The default is an empty list. The number of isotopes in a *channel* depends on the method. For example, a *BlochDecaySpectrum* method is a single channel method, in which case, the value of this attribute is a list with a single isotope symbol, ['13C'].
- `spectral_dimensions` (List of *SpectralDimension* (page 313) or dict objects (optional).) – The number of spectral dimensions depends on the given method. For example, a *BlochDecaySpectrum* method is a one-dimensional method and thus requires a single spectral dimension. The default is a single default *SpectralDimension* (page 313) object.
- `magetic_flux_density` (*float (optional)*) – A global value for the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default is `9.4`.

- `rotor_angle` (*float (optional)*) – A global value for the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is `0.9553166`, i.e. the magic angle.
- `affine_matrix` (*np.ndarray or list (optional)*) – An affine transformation square matrix, $\mathbf{A} \in \mathbb{R}^{n \times n}$, where n is the number of spectral dimensions. The affine operation follows $\mathbf{V}' = \mathbf{A} \cdot \mathbf{V}$, where $\mathbf{V} \in \mathbb{R}^n$ and $\mathbf{V}' \in \mathbb{R}^n$ are the initial and transformed frequency coordinates.

Returns

Return type A *Method* (page 310) instance.

Note: If any parameter is defined outside of the *spectral_dimensions* list, the value of those parameters is considered global. In a multi-event method, you may also assign parameter values to individual events.

Example

```
>>> from mrsimulator.methods import Method2D
>>> method = Method2D(
...     channels=["87Rb"],
...     magnetic_flux_density=7,  # in T. Global value for `magnetic_flux_density`.
...     rotor_angle=0.95531,  # in rads. Global value for the `rotor_angle`.
...     spectral_dimensions=[
...         {
...             "count": 256,
...             "spectral_width": 4e3,  # in Hz
...             "reference_offset": -5e3,  # in Hz
...             "event": [
...                 {  # Global value for the `magnetic_flux_density` and `rotor_angle`
...                     # is used during this event.
...                     "transition_query": {"P": [-3], "D": [0]}
...                 }
...             ],
...         },
...         {
...             "count": 512,
...             "spectral_width": 1e4,  # in Hz
...             "reference_offset": -4e3,  # in Hz
...             "event": [
...                 {  # Global value for the `magnetic_flux_density` is used during this
...                     # event. User defined local value for `rotor_angle` is used here.
...                     "rotor_angle": 1.2238,  # in rads
...                     "transition_query": {"P": [-1], "D": [0]},
...                 }
...             ],
...         },
...     ],
...     affine_matrix=[[1, -1], [0, 1]],
... )
```

Multi-quantum variable-angle spinning

The following classes are used in simulating multi-quantum variable-angle spinning spectrum correlating the frequencies from the symmetric multiple-quantum transition to the central transition frequencies. The p and d pathways for the MQVAS methods are

$$\begin{aligned} p &: 0 \rightarrow M \rightarrow -1 \\ d &: 0 \rightarrow 0 \rightarrow 0 \end{aligned} \tag{16.5}$$

where M is the multiple-quantum number. The value of M depends on the spin quantum number, I, and is listed in Table 16.4.

Affine mapping

The resulting spectrum is sheared and scaled, such that the frequencies along the indirect dimension are given as

$$\langle \Omega \rangle_{\text{MQ-VAS}} = \frac{1}{1+\kappa}\Omega_{m,-m} + \frac{\kappa}{1+\kappa}\Omega_{1/2,-1/2}. \tag{16.6}$$

Here, $\langle \Omega \rangle_{\text{MQ-VAS}}$ is the average frequency along the indirect dimension, $\Omega_{m,-m}$ and $\Omega_{1/2,-1/2}$ are the frequency contributions from the $|m\rangle \rightarrow |-m\rangle$ symmetric multiple-quantum transition and the central transition, respectively, and κ is the shear factor. The values of the shear factor for various transitions are listed in Table 16.4.

Table 16.4: The table lists the multi-quantum transition associated with the spin I, and the corresponding shear factor, κ, used in affine mapping of the MQ-VAS methods.

Spin	Symmetric multi-quantum transition	M	κ
3/2	$\left(\frac{3}{2} \rightarrow -\frac{3}{2}\right)$	-3	21/27
5/2	$\left(-\frac{3}{2} \rightarrow \frac{3}{2}\right)$	3	114/72
5/2	$\left(\frac{5}{2} \rightarrow -\frac{5}{2}\right)$	-5	150/72
7/2	$\left(-\frac{3}{2} \rightarrow \frac{3}{2}\right)$	3	303/135
7/2	$\left(-\frac{5}{2} \rightarrow \frac{5}{2}\right)$	5	165/135
7/2	$\left(\frac{7}{2} \rightarrow -\frac{7}{2}\right)$	-7	483/135
9/2	$\left(-\frac{3}{2} \rightarrow \frac{3}{2}\right)$	3	546/216
9/2	$\left(-\frac{5}{2} \rightarrow \frac{5}{2}\right)$	5	570/216
9/2	$\left(-\frac{7}{2} \rightarrow \frac{7}{2}\right)$	5	84/216

Triple-quantum variable-angle spinning method

class mrsimulator.methods.ThreeQ_VAS(*, property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}, name: str = None, label: str = None, description: str = None, channels: List[str] = [], spectral_dimensions: List[mrsimulator.method.spectral_dimension.SpectralDimension (page 313)] = [SpectralDimension(property_units={'spectral_width': 'Hz', 'reference_offset': 'Hz', 'origin_offset': 'Hz'}, count=1024, spectral_width=25000.0, reference_offset=0.0, origin_offset=None, label=None, description=None, events=[])], affine_matrix: Union[numpy.ndarray, List] = None, simulation: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None, experiment: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None)

Simulate a sheared and scaled 3Q 2D variable-angle spinning spectrum.

Parameters

- `channels` – A list of isotope symbols over which the method will be applied.
- `spectral_dimensions` – A list of python dict. Each dict is contains keywords that describe the coordinates along a spectral dimension. The keywords along with its definition are:
  - **count:** An optional integer with the number of points, N, along the dimension. The default value is 1024.
  - **spectral_width:** An *optional* float with the spectral width, Δx, along the dimension in units of Hz. The default is 25 kHz.
  - **reference_offset:** An *optional* float with the reference offset, x_0 along the dimension in units of Hz. The default value is 0 Hz.
  - **origin_offset:** An *optional* float with the origin offset (Larmor frequency) along the dimension in units of Hz. The default value is None.
- `magetic_flux_density` – An *optional* float containing the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default value is 9.4.
- `rotor_angle` – An *optional* float containing the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is 0.9553166, i.e. the magic angle.

Note: The attribute *rotor_frequency* cannot be modified for this method and is set to simulate an infinite speed spectrum.

Returns A *Method* (page 310) instance.

Example

```
>>> method = ThreeQ_VAS(
...     channels=["87Rb"],
...     magnetic_flux_density=7,  # in T
...     spectral_dimensions=[
...         {
...             "count": 128,
...             "spectral_width": 3e3,  # in Hz
...             "reference_offset": -2e3,  # in Hz
...             "label": "Isotropic dimension",
...         },
...         {
```

(continues on next page)

(continued from previous page)

```
...             "count": 512,
...             "spectral_width": 1e4,  # in Hz
...             "reference_offset": -5e3,  # in Hz
...             "label": "MAS dimension",
...         },
...     ],
... )
>>> sys = SpinSystem(sites=[Site(isotope='87Rb')])
>>> method.get_transition_pathways(sys)
[|-1.5⟩⟨1.5| ⟶ |-0.5⟩⟨0.5|]
```

Five-quantum variable-angle spinning method

class mrsimulator.methods.FiveQ_VAS(*, *property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}*, *name: str = None*, *label: str = None*, *description: str = None*, *channels: List[str] = []*, *spectral_dimensions: List[*mrsimulator.method.spectral_dimension.SpectralDimension *(page 313)] = [SpectralDimension(property_units={'spectral_width': 'Hz', 'reference_offset': 'Hz', 'origin_offset': 'Hz'}, count=1024, spectral_width=25000.0, reference_offset=0.0, origin_offset=None, label=None, description=None, events=[])]*, *affine_matrix: Union[numpy.ndarray, List] = None*, *simulation: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None*, *experiment: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None*)

Simulate a sheared and scaled 5Q variable-angle spinning spectrum.

Parameters

- `channels` – A list of isotope symbols over which the method will be applied.
- `spectral_dimensions` – A list of python dict. Each dict is contains keywords that describe the coordinates along a spectral dimension. The keywords along with its definition are:
  - **count:** An optional integer with the number of points, N, along the dimension. The default value is 1024.
  - **spectral_width:** An *optional* float with the spectral width, Δx, along the dimension in units of Hz. The default is 25 kHz.
  - **reference_offset:** An *optional* float with the reference offset, x_0 along the dimension in units of Hz. The default value is 0 Hz.
  - **origin_offset:** An *optional* float with the origin offset (Larmor frequency) along the dimension in units of Hz. The default value is None.
- `magetic_flux_density` – An *optional* float containing the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default value is 9.4.
- `rotor_angle` – An *optional* float containing the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is 0.9553166, i.e. the magic angle.

Note: The attribute *rotor_frequency* cannot be modified for this method and is set to simulate an infinite speed spectrum.

Returns A *Method* (page 310) instance.

Example

```
>>> method = FiveQ_VAS(
...     channels=["17O"],
...     magnetic_flux_density=11.7,  # in T
...     spectral_dimensions=[
...         {
...             "count": 512,
...             "spectral_width": 5e3,  # in Hz
...             "reference_offset": -3e3,  # in Hz
...             "label": "Isotropic dimension",
...         },
...         {
...             "count": 512,
...             "spectral_width": 2e4,  # in Hz
...             "reference_offset": -2e3,  # in Hz
...             "label": "MAS dimension",
...         },
...     ],
... )
>>> sys = SpinSystem(sites=[Site(isotope='17O')])
>>> method.get_transition_pathways(sys)
[|-2.5⟩⟨2.5| ⟶ |-0.5⟩⟨0.5|]
```

Seven-quantum variable-angle spinning method

class mrsimulator.methods.SevenQ_VAS(*, property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}, name: str = None, label: str = None, description: str = None, channels: List[str] = [], spectral_dimensions: List[mrsimulator.method.spectral_dimension.SpectralDimension (page 313)] = [SpectralDimension(property_units={'spectral_width': 'Hz', 'reference_offset': 'Hz', 'origin_offset': 'Hz'}, count=1024, spectral_width=25000.0, reference_offset=0.0, origin_offset=None, label=None, description=None, events=[])], affine_matrix: Union[numpy.ndarray, List] = None, simulation: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None, experiment: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None)

Simulate a sheared and scaled 7Q variable-angle spinning spectrum.

Parameters

- `channels` – A list of isotope symbols over which the method will be applied.
- `spectral_dimensions` – A list of python dict. Each dict is contains keywords that describe the coordinates along a spectral dimension. The keywords along with its definition are:
  - **count:** An optional integer with the number of points, N, along the dimension. The default value is 1024.
  - **spectral_width:** An *optional* float with the spectral width, Δx, along the dimension in units of Hz. The default is 25 kHz.
  - **reference_offset:** An *optional* float with the reference offset, x_0 along the dimension in units of Hz. The default value is 0 Hz.
  - **origin_offset:** An *optional* float with the origin offset (Larmor frequency) along the dimension in units of Hz. The default value is None.
- `magetic_flux_density` – An *optional* float containing the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default value is 9.4.

- `rotor_angle` – An *optional* float containing the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is 0.9553166, i.e. the magic angle.

Note: The attribute *rotor_frequency* cannot be modified for this method and is set to simulate an infinite speed spectrum.

Returns A *Method* (page 310) instance.

Example

```
>>> method = SevenQ_VAS(
...     channels=["51V"],
...     magnetic_flux_density=4.7,  # in T
...     spectral_dimensions=[
...         {
...             "count": 256,
...             "spectral_width": 1e3,  # in Hz
...             "reference_offset": 1e3,  # in Hz
...             "label": "Isotropic dimension",
...         },
...         {
...             "count": 1024,
...             "spectral_width": 1e4,  # in Hz
...             "reference_offset": -2e3,  # in Hz
...             "label": "MAS dimension",
...         },
...     ],
... )
>>> sys = SpinSystem(sites=[Site(isotope='51V')])
>>> method.get_transition_pathways(sys)
[|-3.5⟩⟨3.5| ⟶ |-0.5⟩⟨0.5|]
```

Satellite-transition variable-angle spinning (ST-VAS)

The following classes are used in simulating satellite-transition variable-angle spinning spectrum correlating the frequencies from the satellite transitions to the central transition frequencies. The p and d pathways for the ST-VAS methods are

$$\begin{aligned} p &: 0 \rightarrow -1 \rightarrow -1 \\ d &: 0 \rightarrow \pm d_0 \rightarrow 0 \end{aligned} \tag{16.7}$$

where $d_0 = m_f^2 - m_i^2$ for transition $|m_i\rangle \rightarrow |m_f\rangle$. The value of n depends on the spin quantum number, I, and is listed in Table 16.5.

Affine mapping

The resulting spectrum is sheared and scaled, such that the frequencies along indirect dimension are given as

$$\langle\Omega\rangle_{\text{ST-VAS}} = \frac{1}{1+\kappa}\Omega_{m,m-1} + \frac{\kappa}{1+\kappa}\Omega_{1/2,-1/2}. \tag{16.8}$$

Here, $\langle\Omega\rangle_{\text{ST-VAS}}$ is the average frequency along the indirect dimension, $\Omega_{m,m-1}$ and $\Omega_{1/2,-1/2}$ are the frequency contributions from the $|m\rangle \rightarrow |m-1\rangle$ satellite transition and the central transition, respectively, and κ is the shear factor. The values of the shear factor for various satellite transitions are listed in Table 16.5.

Table 16.5: The table lists the satellite transitions associated with the spin I, and the corresponding shear factor, κ, used in affine mapping of the ST-VAS methods.

Spin	Satellite transitions	d_0	κ
3/2	$\left(\frac{3}{2} \rightarrow \frac{1}{2}\right)$, $\left(-\frac{1}{2} \rightarrow -\frac{3}{2}\right)$	2	24/27
5/2	$\left(-\frac{3}{2} \rightarrow -\frac{1}{2}\right)$, $\left(\frac{1}{2} \rightarrow \frac{3}{2}\right)$	2	21/72
5/2	$\left(\frac{5}{2} \rightarrow \frac{3}{2}\right)$, $\left(-\frac{3}{2} \rightarrow -\frac{5}{2}\right)$	4	132/72
7/2	$\left(-\frac{3}{2} \rightarrow -\frac{1}{2}\right)$, $\left(\frac{1}{2} \rightarrow \frac{3}{2}\right)$	2	84/135
7/2	$\left(-\frac{5}{2} \rightarrow -\frac{3}{2}\right)$, $\left(\frac{3}{2} \rightarrow \frac{5}{2}\right)$	4	69/135
9/2	$\left(-\frac{3}{2} \rightarrow -\frac{1}{2}\right)$, $\left(\frac{1}{2} \rightarrow \frac{3}{2}\right)$	2	165/216
9/2	$\left(-\frac{5}{2} \rightarrow -\frac{3}{2}\right)$, $\left(\frac{3}{2} \rightarrow \frac{5}{2}\right)$	4	12/216

Inner satellite variable-angle spinning method

`class mrsimulator.methods.ST1_VAS(`**, property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}, name: str = None, label: str = None, description: str = None, channels: List[str] = [], spectral_dimensions: List[*mrsimulator.method.spectral_dimension.SpectralDimension *(page 313)] = [SpectralDimension(property_units={'spectral_width': 'Hz', 'reference_offset': 'Hz', 'origin_offset': 'Hz'}, count=1024, spectral_width=25000.0, reference_offset=0.0, origin_offset=None, label=None, description=None, events=[])], affine_matrix: Union[numpy.ndarray, List] = None, simulation: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None, experiment: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None*`)`

Simulate a sheared and scaled inner satellite and central transition correlation spectrum.

Parameters

- `channels` – A list of isotope symbols over which the method will be applied.
- `spectral_dimensions` – A list of python dict. Each dict is contains keywords that describe the coordinates along a spectral dimension. The keywords along with its definition are:
  - **count:** An optional integer with the number of points, N, along the dimension. The default value is 1024.
  - **spectral_width:** An *optional* float with the spectral width, Δx, along the dimension in units of Hz. The default is 25 kHz.
  - **reference_offset:** An *optional* float with the reference offset, x_0 along the dimension in units of Hz. The default value is 0 Hz.
  - **origin_offset:** An *optional* float with the origin offset (Larmor frequency) along the dimension in units of Hz. The default value is None.
- `magetic_flux_density` – An *optional* float containing the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default value is 9.4.
- `rotor_angle` – An *optional* float containing the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is 0.9553166, i.e. the

magic angle.

Note: The attribute *rotor_frequency* cannot be modified for this method and is set to simulate an infinite speed spectrum.

Returns A *Method* (page 310) instance.

Example

```
>>> method = ST1_VAS(
...     channels=["87Rb"],
...     magnetic_flux_density=9.4,  # in T
...     spectral_dimensions=[
...         {
...             "count": 128,
...             "spectral_width": 1e3,  # in Hz
...             "reference_offset": -5e3,  # in Hz
...             "label": "Isotropic dimension",
...         },
...         {
...             "count": 256,
...             "spectral_width": 1e4,  # in Hz
...             "reference_offset": -3e3,  # in Hz
...             "label": "MAS dimension",
...         },
...     ],
... )
>>> sys = SpinSystem(sites=[Site(isotope='87Rb')])
>>> pprint(method.get_transition_pathways(sys))
[|-1.5⟩⟨-0.5| ⟶ |-0.5⟩⟨0.5|, |0.5⟩⟨1.5| ⟶ |-0.5⟩⟨0.5|]
```

Second to inner satellite variable-angle spinning method

class mrsimulator.methods.ST2_VAS(*, *property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}*, *name: str = None*, *label: str = None*, *description: str = None*, *channels: List[str] = []*, *spectral_dimensions: List[*mrsimulator.method.spectral_dimension.SpectralDimension *(page 313)] = [SpectralDimension(property_units={'spectral_width': 'Hz', 'reference_offset': 'Hz', 'origin_offset': 'Hz'}, count=1024, spectral_width=25000.0, reference_offset=0.0, origin_offset=None, label=None, description=None, events=[])]*, *affine_matrix: Union[numpy.ndarray, List] = None*, *simulation: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None*, *experiment: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None*)

Simulate a sheared and scaled second to inner satellite and central transition correlation spectrum.

Parameters

- `channels` – A list of isotope symbols over which the method will be applied.
- `spectral_dimensions` – A list of python dict. Each dict is contains keywords that describe the coordinates along a spectral dimension. The keywords along with its definition are:

- **count:** An optional integer with the number of points, N, along the dimension. The default value is 1024.
- **spectral_width:** An *optional* float with the spectral width, Δx, along the dimension in units of Hz. The default is 25 kHz.
- **reference_offset:** An *optional* float with the reference offset, x_0 along the dimension in units of Hz. The default value is 0 Hz.
- **origin_offset:** An *optional* float with the origin offset (Larmor frequency) along the dimension in units of Hz. The default value is None.

- `magetic_flux_density` – An *optional* float containing the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default value is 9.4.
- `rotor_angle` – An *optional* float containing the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is 0.9553166, i.e. the magic angle.

Note: The attribute *rotor_frequency* cannot be modified for this method and is set to simulate an infinite speed spectrum.

Returns A *Method* (page 310) instance.

Example

```
>>> method = ST2_VAS(
...     channels=["17O"],
...     magnetic_flux_density=9.4,  # in T
...     spectral_dimensions=[
...         {
...             "count": 256,
...             "spectral_width": 4e3,  # in Hz
...             "reference_offset": -5e3,  # in Hz
...             "label": "Isotropic dimension",
...         },
...         {
...             "count": 512,
...             "spectral_width": 1e4,  # in Hz
...             "reference_offset": -4e3,  # in Hz
...             "label": "MAS dimension",
...         },
...     ],
... )
>>> sys = SpinSystem(sites=[Site(isotope='17O')])
>>> pprint(method.get_transition_pathways(sys))
[|-2.5⟩⟨-1.5| ⟶ |-0.5⟩⟨0.5|, |1.5⟩⟨2.5| ⟶ |-0.5⟩⟨0.5|]
```

Spinning sideband correlation method

class mrsimulator.methods.SSB2D(*, property_units: Dict = {'magnetic_flux_density': 'T', 'rotor_angle': 'rad', 'rotor_frequency': 'Hz'}, name: str = None, label: str = None, description: str = None, channels: List[str] = [], spectral_dimensions: List[mrsimulator.method.spectral_dimension.SpectralDimension (page 313)] = [SpectralDimension(property_units={'spectral_width': 'Hz', 'reference_offset': 'Hz', 'origin_offset': 'Hz'}, count=1024, spectral_width=25000.0, reference_offset=0.0, origin_offset=None, label=None, description=None, events=[])], affine_matrix: Union[numpy.ndarray, List] = None, simulation: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None, experiment: Union[csdmpy.csdm.CSDM, numpy.ndarray] = None)

Simulating a sheared 2D finite to infinite speed MAS correlation spectum.

For spin I=1/2, the infinite speed MAS is the isotropic dimension.

Parameters

- `channels` – A list of isotope symbols over which the method will be applied.
- `spectral_dimensions` – A list of python dict. Each dict is contains keywords that describe the coordinates along a spectral dimension. The keywords along with its definition are:
  - **count:** An optional integer with the number of points, N, along the dimension. The default value is 1024.
  - **spectral_width:** An *optional* float with the spectral width, Δx, along the dimension in units of Hz. The default is 25 kHz.
  - **reference_offset:** An *optional* float with the reference offset, x_0 along the dimension in units of Hz. The default value is 0 Hz.
  - **origin_offset:** An *optional* float with the origin offset (Larmor frequency) along the dimension in units of Hz. The default value is None.
- `rotor_frequency` – An *optional* float containing the sample spinning frequency ν_r, in units of Hz. The default value is `0`.
- `magetic_flux_density` – An *optional* float containing the macroscopic magnetic flux density, H_0, of the applied external magnetic field in units of T. The default value is 9.4.
- `rotor_angle` – An *optional* float containing the angle between the sample rotation axis and the applied external magnetic field, θ, in units of rad. The default value is 0.9553166, i.e. the magic angle.

Returns A *Method* (page 310) instance.

Example

```
>>> method = SSB2D(
...     channels=["13C"],
...     magnetic_flux_density=7,  # in T
...     rotor_frequency=1500, # in Hz
...     spectral_dimensions=[
...         {
...             "count": 16,
...             "spectral_width": 16*1500,  # in Hz (= count * rotor_frequency)
...             "reference_offset": -5e3,  # in Hz
...             "label": "Sideband dimension",
...         },
...         {
...             "count": 512,
...             "spectral_width": 1e4,  # in Hz
```

(continues on next page)

(continued from previous page)

```
...             "reference_offset": -4e3,  # in Hz
...             "label": "Isotropic dimension",
...         },
...     ],
... )
>>> sys = SpinSystem(sites=[Site(isotope='13C')])
>>> method.get_transition_pathways(sys)
[|-0.5⟩⟨0.5| ⟶ |-0.5⟩⟨0.5|]
```

16.8 Other Objects

16.8.1 Sites

class mrsimulator.simulator.Sites(*data=[]*)

Bases: mrsimulator.utils.abstract_list.AbstractList

A list of unique *Site* (page 302) objects within a simulator object.

Method Documentation

to_pd()

Return sites as a pandas dataframe.

16.8.2 ZeemanState

class mrsimulator.spin_system.zeemanstate.ZeemanState(*n_sites*, **args*)

Bases: object

Zeeman energy state class.

Method Documentation

tolist()

16.8.3 SymmetricTensor

class mrsimulator.spin_system.tensors.SymmetricTensor(***, *property_units: Dict = {'Cq': 'Hz', 'D': 'Hz', 'alpha': 'rad', 'beta': 'rad', 'gamma': 'rad', 'zeta': 'ppm'}*, *zeta: float = None*, *Cq: float = None*, *D: float = None*, *eta: mrsimulator.spin_system.tensors.ConstrainedFloatValue = None*, *alpha: float = None*, *beta: float = None*, *gamma: float = None*)

Bases: mrsimulator.utils.parseable.Parseable

Base SymmetricTensor class representing the traceless symmetric part of an irreducible second-rank tensor.

`zeta`
The anisotropy parameter of the nuclear shielding tensor, in ppm, expressed using the Haeberlen convention. The default value is None.

Example

```
>>> shielding = SymmetricTensor()
>>> shielding.zeta = 10
```

Type float (optional)

`Cq`
The quadrupolar coupling constant, in Hz, derived from the electric field gradient tensor. The default value is None.

Example

```
>>> efg = SymmetricTensor()
>>> efg.Cq = 10e6
```

Type float (optional)

`eta`
The asymmetry parameter of the SymmetricTensor expressed using the Haeberlen convention. The default value is None.

Example

```
>>> shielding.eta = 0.1
>>> efg.eta = 0.5
```

Type float (optional)

`alpha`
Euler angle, α, in radians. The default value is None.

Example

```
>>> shielding.alpha = 0.15
>>> efg.alpha = 1.5
```

Type float (optional)

`beta`
Euler angle, β, in radians. The default value is None.

Example

```
>>> shielding.beta = 3.1415
>>> efg.beta = 1.1451
```

Type float (optional)

gamma
: Euler angle, γ, in radians. The default value is None.

Example

```
>>> shielding.gamma = 2.1
>>> efg.gamma = 0
```

Type float (optional)

Example

```
>>> shielding = SymmetricTensor(zeta=10, eta=0.1, alpha=0.15, beta=3.14, gamma=2.1)
>>> efg = SymmetricTensor(Cq=10e6, eta=0.5, alpha=1.5, beta=1.1451, gamma=0)
```

Method Documentation

json() → dict
: Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a number and a unit.

16.8.4 AntisymmetricTensor

class mrsimulator.spin_system.tensors.AntisymmetricTensor(*, *property_units: Dict = {'alpha': 'rad', 'beta': 'rad', 'zeta': 'ppm'}*, *zeta: float = None*, *alpha: float = None*, *beta: float = None*)

Bases: mrsimulator.utils.parseable.Parseable

Base SymmetricTensor class representing the traceless symmetric part of an irreducible second-rank tensor.

zeta
: The anisotropy parameter of the AntiSymmetricTensor expressed using the Haeberlen convention. The default value is None.

Type Optional[float]

alpha
: Euler angle, alpha, given in radian. The default value is None.

Type Optional[float]

beta
: Euler angle, beta, given in radian. The default value is None.

Type Optional[float]

Method Documentation

`json()` → dict
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a number and a unit.

16.8.5 Isotope

`class mrsimulator.spin_system.isotope.Isotope(*, symbol: str)`
Bases: `pydantic.main.BaseModel`

The Isotope class.

`symbol`
The isotope symbol given as the atomic number followed by the atomic symbol.

Type str (required)

Example

```
>>> # 13C isotope information
>>> carbon = Isotope(symbol='13C')
>>> carbon.spin
0.5
>>> carbon.natural_abundance # in %
1.11
>>> carbon.gyromagnetic_ratio # in MHz/T
10.7084
>>> carbon.atomic_number
6
>>> carbon.quadrupole_moment # in eB
0.0
```

Attribute Description

`spin`	Spin quantum number, I, of the isotope.
`natural_abundance`	Natural abundance of the isotope in units of %.
`gyromagnetic_ratio`	Reduced gyromagnetic ratio of the nucleus given in units of MHz/T.
`atomic_number`	Atomic number of the isotope.
`quadrupole_moment`	Quadrupole moment of the nucleus given in units of eB (electron-barn).

Method Documentation

`json()` → dict
Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a value and a unit.

16.8.6 Transition

`class mrsimulator.transition.Transition(*, initial: List[float] = [], final: List[float] = [])`
Bases: `pydantic.main.BaseModel`

Base Transition class describes a spin transition between two energy states, where the energy states are described using the weakly coupled basis.

$$|m_{i,0}, m_{i,1}, ...m_{i,N}\rangle \rightarrow |m_{f,0}, m_{f,1}, ...m_{f,N}\rangle \tag{16.9}$$

Parameters
- `initial` (*list*) – The initial Zeeman energy state represented as a list of quantum numbers $m_{i,n}$.
- `final` (*list*) – The final Zeeman energy state represented as a list of quantum numbers $m_{f,n}$.

Example

```
>>> from mrsimulator.transition import Transition
>>> t1 = Transition(initial = [0.5, 0.5], final = [0.5, -0.5])
>>> t1
|0.5, -0.5⟩⟨0.5, 0.5|
```

p (page 337)	Return the total Δm (m_final-m_initial) value of the spin transition.
delta_m (page 338)	An alias for p
P (page 338)	Return a list of Δm values of the spin transition for each site in a weakly coupled basis.
D (page 338)	Return a list of Δm**2 values of the spin transition for each site in a weakly coupled basis.

Attribute Documentation

p
Return the total Δm (m_final-m_initial) value of the spin transition.

Example

```
>>> t1.p
-1.0
```

`delta_m`
An alias for p

`P`
Return a list of Δm values of the spin transition for each site in a weakly coupled basis.

Example

```
>>> t1.P
array([ 0., -1.])
```

`D`
Return a list of Δm**2 values of the spin transition for each site in a weakly coupled basis.

Example

```
>>> t1.D
array([0., 0.])
```

Method Documentation

`tolist()` → list
Convert the transition to a list of quantum numbers where the first N quantum numbers corresponds to the initial energy state, while the last N corresponds to the final energy state, where N is the number of sites.

Example

```
>>> t1.tolist()
[0.5, 0.5, 0.5, -0.5]
```

`json()` → dict
Parse the class object to a JSON compliant python dictionary object.

Example

```
>>> t1.json()
{'initial': [0.5, 0.5], 'final': [0.5, -0.5]}
```

16.8.7 TransitionPathway

class mrsimulator.transition.TransitionPathway(*data: list = []*)

Bases: mrsimulator.transition.pathway.TransitionList

Base TransitionPathway class is a list of connected Transitions.

Example

```
>>> from mrsimulator.transition import TransitionPathway, Transition
>>> t1 = Transition(initial = [0.5, 0.5], final = [0.5, -0.5])
>>> t2 = Transition(initial=[0.5, 0.5], final=[-0.5, 0.5])
>>> path = TransitionPathway([t1, t2])
>>> path
|0.5, -0.5⟩⟨0.5, 0.5| ⟶ |-0.5, 0.5⟩⟨0.5, 0.5|
```

Method Documentation

tolist()

Expand TransitionPathway to a Python list.

Example

```
>>> path.tolist()
[0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5]
```

json() → dict

Parse the class object to a JSON compliant python dictionary object.

Example

```
>>> pprint(path.json())
[{'final': [0.5, -0.5], 'initial': [0.5, 0.5]},
 {'final': [-0.5, 0.5], 'initial': [0.5, 0.5]}]
```

16.9 Utility functions

mrsimulator.utils.collection.single_site_system_generator(*isotopes*, *isotropic_chemical_shifts=0*, *shielding_symmetric=None*, *quadrupolar=None*, *abundance=None*, *site_labels=None*, *site_names=None*, *site_descriptions=None*, *rtol=0.001*)

Generate and return a list of single-site spin systems from the input parameters.

Parameters

- isotopes – A string or a list of site isotopes.
- isotropic_chemical_shifts – A float or a list/ndarray of values. The default value is 0.

- `shielding_symmetric` – A shielding symmetric like dict object, where the keyword value can either be a float or a list/ndarray of floats. The default value is None. The allowed keywords are `zeta`, `eta`, `alpha`, `beta`, and `gamma`.
- `quadrupolar` – A quadrupolar like dict object, where the keyword value can either be a float or a list/ndarray of floats. The default value is None. The allowed keywords are `Cq`, `eta`, `alpha`, `beta`, and `gamma`.
- `abundance` – A float or a list/ndarray of floats describing the abundance of each spin system.
- `site_labels` – A string or a list of strings each with a site label. The default is None.
- `site_names` – A string or a list of strings each with a site name. The default is None.
- `site_descriptions` – A string or a list of strings each with a site description. Default is None.
- `rtol` – The relative tolerance. This value is used in determining the cutoff abundance given as $\texttt{abundance}_{\texttt{cutoff}} = \texttt{rtol} * \max(\texttt{abundance})$. The spin systems with abundance below this threshold are ignored.

Note: The parameter value can either be a float or a list/ndarray. If the parameter value is a float, the given value is assigned to the respective parameter in all the spin systems. If the parameter value is a list or ndarray, its ith value is assigned to the respective parameter of the ith spin system. When multiple parameter values are given as lists/ndarrays, the length of all the lists must be the same.

`mrsimulator.utils.get_spectral_dimensions(`*csdm_object*, *units=False*`)`

Extract the count, spectral_width, and reference_offset parameters, associated with the spectral dimensions of the method, from the CSDM dimension objects.

Parameters `csdm_object` – A CSDM object holding the measurement dataset.

Returns A list of dict objects, where each dict containts the count, spectral_width, and reference_offset.

16.10 Mrsimulator IO

`mrsimulator.save(`*filename: str*, *simulator:* mrsimulator.simulator.Simulator *(page 289)*, *signal_processors: Optional[list] = None*, *params: Optional[lmfit.parameter.Parameters] = None*, *with_units: bool = True*`)`

Serialize the Simulator, list of SignalProcessor, and lmfit Parameters objects to a .mrsim file.

Parameters

- `filename` (*str*) – The data is serialized to this file.
- `sim` – Simulator object.
- `signal_processors` – A list of PostSimulator objects corresponding to the methods in the Simulator object. Default is None.
- `params` – lmfit Parameters object. Default is None.
- `with_units` (*bool*) – If true, physical quantities are represented as string with units. The default is True.

`mrsimulator.dict(`*simulator:* mrsimulator.simulator.Simulator *(page 289)*, *signal_processors: Optional[list] = None*, *params: Optional[lmfit.parameter.Parameters] = None*, *with_units: bool = True*`)`

Export the Simulator, list of SignalProcessor, and lmfit Parameters objects to a python dictionary.

Parameters

- `sim` – Simulator object.
- `signal_processors` – A list of PostSimulator objects corresponding to the methods in the Simulator object. Default is None.
- `params` – lmfit Parameters object. Default is None.

- `with_units` (*bool*) – If true, physical quantities are represented as string with units. The default is True.

Returns Python dictionary

`mrsimulator.load`(*filename: str*, *parse_units: bool = True*)

Load Simulator, list of SignalProcessor and optionally lmfit Parameters objects from the .mrsim file.

Parameters

- `filename` (*str*) – The location to the .mrsim file.
- `parse_units` (*bool*) – If true, parse the dictionary for units. The default is True.

Returns Simulator, List[SignalProcessor], Parameters.

Return type Ordered List

`mrsimulator.parse`(*py_dict*, *parse_units: bool = True*)

Parse the dictionary object to respective Simulator, SignalProcessor and optionally lmfit Parameters object.

Parameters

- `py_dict` (*dict*) – Python dictionary representation of mrsimulator.
- `parse_units` (*bool*) – If true, parse the dictionary for units. Default is True.

Returns Simulator, List[SignalProcessor], Parameters.

Return type Ordered List

CHAPTER

SEVENTEEN

SIGNAL-PROCESSING API

17.1 Signal Processing

class mrsimulator.signal_processing.SignalProcessor(*, *processed_data: csdmpy.csdm.CSDM = None*, *operations: List[mrsimulator.signal_processing._base.Operation] = []*)

Bases: pydantic.main.BaseModel

Signal processing class to apply a series of operations to the dependent variables of the simulation dataset.

operations
: A list of operations.

Type List

Examples

```
>>> post_sim = SignalProcessor(operations=[o1, o2])
```

Method Documentation

classmethod parse_dict_with_units(*py_dict: dict*)
: Parse a list of operations dictionary to a SignalProcessor class object.

Parameters pt_dict – A python dict object.

json() → dict
: Parse the class object to a JSON compliant python dictionary object, where the attribute value with physical quantity is expressed as a string with a value and a unit.

Returns A Dict object.

apply_operations(*data*, ***kwargs*)
: Function to apply all the operation functions in the operations member of a SignalProcessor object. Operations applied sequentially over the data member.

Returns A copy of the data member with the operations applied to it.
Return type CSDM object

17.2 Operations

17.2.1 Generic operations

Import the module as

```
>>> from mrsimulator import signal_processing as sp
```

Operation Summary

The following list of operations applies to **all dependent variables** within the CSDM object.

`Scale`	Scale the amplitudes of all dependent variables (y) from a CSDM object.
`Linear`	Apply linear operation across all dependent variables (y) from a CSDM object.
`IFFT`	Apply an inverse Fourier transform on all dependent variables of the CSDM object.
`FFT`	Apply a forward Fourier transform on all dependent variables of the CSDM object.

17.2.2 Baseline

Access the sub-module as `sp.baseline`

Operation Summary

The following list of operations applies to **selected dependent variables** within the CSDM object.

`Polynomial`	Add a baseline polynomial to all dependent variables (y) in the CSDM object.
`ConstantOffset`	Add an offset to the dependent variables (y) of the CSDM object.

See also:

Signal Processing (page 57) for a details.

17.2.3 Apodization

Access the sub-module as `sp.apodization`

Operation Summary

The following list of operations applies to **selected dependent variables** within the CSDM object.

`Gaussian`	Apodize a dependent variable of the CSDM object with a Gaussian function.
`Exponential`	Apodize a dependent variable of the CSDM object by an exponential function.

See also:

Signal Processing (page 57) for a details.

17.2.4 Affine Transformation

Access the sub-module as `sp.affine`

Operation Summary

The following list of operations applies to **selected dependent variables** within the CSDM object.

`Shear`	Apply a shear parallel to dimension at index parallel and normal to dimension at index dim_index.
`Scale`	Scale the dimension along the specified dimension index.

See also:

Signal Processing (page 57) for a details.

CHAPTER

EIGHTEEN

MODELS API

18.1 Czjzek distribution Model

class mrsimulator.models.CzjzekDistribution(*sigma: float*, *polar=False*)

A Czjzek distribution model class.

The Czjzek distribution model is a random sampling of second-rank traceless symmetric tensors whose explicit matrix form follows

$$\mathbf{S} = \begin{bmatrix} \sqrt{3}U_5 - U_1 & \sqrt{3}U_4 & \sqrt{3}U_2 \\ \sqrt{3}U_4 & -\sqrt{3}U_5 - U_1 & \sqrt{3}U_3 \\ \sqrt{3}U_2 & \sqrt{3}U_3 & 2U_1 \end{bmatrix}, \tag{18.1}$$

where the components, U_i, are randomly drawn from a five-dimensional multivariate normal distribution. Each component, U_i, is a dimension of the five-dimensional uncorrelated multivariate normal distribution with the mean of $< U_i >= 0$ and the variance $< U_i U_i >= \sigma^2$.

$$S_T = S_C(\sigma), \tag{18.2}$$

Parameters `sigma` (*float*) – The Gaussian standard deviation.

Note: In the original Czjzek paper, the parameter σ is given as two times the standard deviation of the multi-variate normal distribution used here.

Example

```
>>> from mrsimulator.models import CzjzekDistribution
>>> cz_model = CzjzekDistribution(0.5)
```

rvs(*size: int*)

Draw random variates of length *size* from the distribution.

Parameters `size` – The number of random points to draw.

Returns A list of two NumPy array, where the first and the second array are the anisotropic/quadrupolar coupling constant and asymmetry parameter, respectively.

Example

```
>>> Cq_dist, eta_dist = cz_model.rvs(size=1000000)
```

pdf(*pos*, *size: int = 400000*)

Generates a probability distribution function by binning the random variates of length size onto the given grid system.

Parameters

- pos – A list of coordinates along the two dimensions given as NumPy arrays.
- size – The number of random variates drawn in generating the pdf. The default is 400000.

Returns A list of x and y coordinates and the corresponding amplitudes.

Example

```
>>> import numpy as np
>>> cq = np.arange(50) - 25
>>> eta = np.arange(21)/20
>>> Cq_dist, eta_dist, amp = cz_model.pdf(pos=[cq, eta])
```

18.1.1 Mini-gallery using czjzek distributions

- *Czjzek distribution (Shielding and Quadrupolar)* (page 113)
- *Czjzek distribution, ^{27}Al (I=5/2) 3QMAS* (page 160)

18.2 Extended Czjzek distribution Model

class mrsimulator.models.ExtCzjzekDistribution(*symmetric_tensor:* mrsimulator.spin_system.tensors.SymmetricTensor *(page 333)*, *eps: float*, *polar=False*)

An extended Czjzek distribution distribution model.

The extended Czjzek random distribution[1] model is an extension of the Czjzek model, given as

$$S_T = S(0) + \rho S_C(\sigma = 1), \tag{18.3}$$

where S_T is the total tensor, $S(0)$ is the dominant tensor, $S_C(\sigma = 1)$ is the Czjzek random model attributing to the random perturbation of the tensor about the dominant tensor, $S(0)$, and ρ is the size of the perturbation. Note, in the above equation, the σ parameter from the Czjzek random model, S_C, has no meaning and is set to one. The factor, ρ, is defined as

$$\rho = \frac{||S(0)||\epsilon}{\sqrt{30}}, \tag{18.4}$$

where $\|S(0)\|$ is the 2-norm of the dominant tensor, and ϵ is a fraction.

Parameters

[1] Gérard Le Caër, Bruno Bureau, and Dominique Massiot, An extension of the Czjzek model for the distributions of electric field gradients in disordered solids and an application to NMR spectra of 71Ga in chalcogenide glasses. Journal of Physics: Condensed Matter, 2010, 22, 065402. DOI: 10.1088/0953-8984/22/6/065402

- `symmetric_tensor` (SymmetricTensor (page 333)) – A shielding or quadrupolar symmetric tensor or equivalent dict object.
- `eps` (*float*) – A fraction determining the extent of perturbation.

Example

```
>>> from mrsimulator.models import ExtCzjzekDistribution
>>> S0 = {"Cq": 1e6, "eta": 0.3}
>>> ext_cz_model = ExtCzjzekDistribution(S0, eps=0.35)
```

`rvs`(*size: int*)

Draw random variates of length *size* from the distribution.

Parameters `size` – The number of random points to draw.

Returns A list of two NumPy array, where the first and the second array are the anisotropic/quadrupolar coupling constant and asymmetry parameter, respectively.

Example

```
>>> Cq_dist, eta_dist = ext_cz_model.rvs(size=1000000)
```

`pdf`(*pos*, *size: int = 400000*)

Generates a probability distribution function by binning the random variates of length size onto the given grid system.

Parameters

- `pos` – A list of coordinates along the two dimensions given as NumPy arrays.
- `size` – The number of random variates drawn in generating the pdf. The default is 400000.

Returns A list of x and y coordinates and the corresponding amplitudes.

Example

```
>>> import numpy as np
>>> cq = np.arange(50) - 25
>>> eta = np.arange(21)/20
>>> Cq_dist, eta_dist, amp = cz_model.pdf(pos=[cq, eta])
```

18.2.1 Mini-gallery using extended czjzek distributions

- *Extended Czjzek distribution (Shielding and Quadrupolar)* (page 117)
- *Simulating site disorder (crystalline)* (page 156)

CHAPTER

NINETEEN

FITTING UTILITY API

19.1 LMFIT supplement functions

mrsimulator.utils.spectral_fitting.make_LMFIT_params(*sim:* mrsimulator.simulator.Simulator *(page 289)*, *processors: Optional[list] = None*, *include={}*)

Parse the Simulator and PostSimulator objects for a list of LMFIT parameters.

Parameters

- `sim` (Simulator (page 289)) – Simulator object.
- `processors` (*list*) – List of SignalProcessor objects. The order must match the order of methods within the simulator object.
- `include` (*set*) – set of keywords from the method object to include as a fitting parameter. Default is {}.

The parameter name assocciated with the spin system within Simulator object is generated using the following nomenclature- *sys_i_site_j_attribute1_attribute2* for attribute with signature `sim.spin_systems[i].sites[j].attribute1.attribute2`

Here, *sys_i* refers to the spin system at index *i*, *site_j* refers to the site at index *j* with in the :math:i^text{th} spin system, and *attribute1* and *attribute2* are the site attrbiutes.

For examples:

`sim.spin_systems[1].sites[0].isotropic_chemical_shift` parametrizes to *sys_1_site_0_isotropic_chemical_shift* while `sim.spin_systems[0].sites[1].quadrupolar.Cq` to *sys_0_site_1_quadrupolar_Cq*.

Returns LMFIT Parameters object.

mrsimulator.utils.spectral_fitting.LMFIT_min_function(*params: lmfit.parameter.Parameters*, *sim:* mrsimulator.simulator.Simulator *(page 289)*, *processors: Optional[list] = None*, *sigma: Optional[list] = None*)

The simulation routine to calculate the vector difference between simulation and experiment based on the parameters update.

Parameters

- `params` – Parameters object containing parameters for OLS minimization.
- `sim` – Simulator object.
- `processors` – A list of PostSimulator objects corresponding to the methods in the Simulator object.
- `sigma` – A list of standard deviations corresponding to the experiments in the Simulator.methods attribute

Returns Array of the differences between the simulation and the experimental data.

mrsimulator.utils.spectral_fitting.bestfit(*sim:* mrsimulator.simulator.Simulator *(page 289)*, *processors: Optional[list] = None*)

Return a list of best fit spectrum ordered relative to the methods in the simulator object.

Parameters

- **sim** (Simulator (page 289)) – The simulator object.
- **processors** (*list*) – List of SignalProcessor objects ordered according to the methods in the simulator object.

mrsimulator.utils.spectral_fitting.residuals(*sim:* mrsimulator.simulator.Simulator *(page 289)*, *processors: Optional[list] = None*)

Return a list of residuals corresponsing to the best fit spectrum. The list is based on the order of methods in the simulator object.

Parameters

- **sim** (Simulator (page 289)) – The simulator object.
- **processors** (*list*) – List of SignalProcessor objects ordered according to the methods in the simulator object.

CHAPTER

TWENTY

C-API REFERENCES

20.1 Spin transition functions (STF), $\xi_L^{(k)}(i,j)$

See also:

Spin transition functions (page 280)

20.1.1 Single nucleus spin transition functions

The single spin transition functions for $|m_i\rangle \rightarrow |m_f\rangle$ transition, where m_j is the spin quantum number, and the subscripts i and f refer to the initial and final energy states.

double `STF_p`(const double mf, const double mi)

Single nucleus spin transition function from the irreducible spherical tensor of rank $L = 1$ is given as

$$\begin{aligned} \mathbb{p}(m_f, m_i) &= \left\langle m_f|\hat{T}_{10}|m_f\right\rangle - \left\langle m_i|\hat{T}_{10}|m_i\right\rangle \\ &= m_f - m_i, \end{aligned} \tag{20.1}$$

where $\hat{T}_{10}$ is the irreducible 1st-rank spherical tensor operator in the rotating tilted frame.

Return The spin transition function $\mathbb{p}$.

Parameters

- `mi`: The quantum number associated with the quantized initial energy level.
- `mf`: The quantum number associated with the quantized final energy level.

double `STF_d`(const double mf, const double mi)

Single nucleus spin transition function from the irreducible spherical tensor of rank $L = 2$ is given as

$$\begin{aligned} \mathbb{d}(m_f, m_i) &= \left\langle m_f|\hat{T}_{20}|m_f\right\rangle - \left\langle m_i|\hat{T}_{20}|m_i\right\rangle \\ &= \sqrt{\frac{3}{2}}\left(m_f^2 - m_i^2\right), \end{aligned} \tag{20.2}$$

where $\hat{T}_{20}$ is the irreducible 2nd-rank spherical tensor operator in the rotating tilted frame.

Return The spin transition function $\mathbb{d}$.

Parameters

- `mi`: The quantum number associated with the quantized initial energy level.
- `mf`: The quantum number associated with the quantized final energy level.

double `STF_f(const double mf, const double mi, const double spin)`

Single nucleus spin transition function from the irreducible spherical tensor of rank $L = 3$ is given as

$$\begin{aligned}\mathbb{f}(m_f, m_i) &= \left\langle m_f|\hat{T}_{30}|m_f\right\rangle - \left\langle m_i|\hat{T}_{30}|m_i\right\rangle \\ &= \frac{1}{\sqrt{10}}[5(m_f^3 - m_i^3) + (1 - 3I(I+1))(m_f - m_i)],\end{aligned} \tag{20.3}$$

where $\hat{T}_{30}$ is the irreducible 3rd-rank spherical tensor operator in the rotating tilted frame.

Return The spin transition function $\mathbb{f}$.

Parameters

- `mi`: The quantum number associated with the quantized initial energy level.
- `mf`: The quantum number associated with the quantized final energy level.
- `spin`: The spin quantum angular momentum number.

Composite single nucleus spin transition functions

The composite single spin transition functions are linear combinations of the single spin transition functions.

void `STF_cL(double *restrict cl_value, const double mf, const double mi, const double spin)`

The following single nucleus composite spin transition functions corresponding to rank $L = [0, 2, 4]$ irreducible tensors results from the second-order corrections to the quadrupole frequency. The functions are defined as

$$\begin{aligned}\mathbb{c}_0(m_f, m_i) &= \frac{4}{\sqrt{125}}\left[I(I+1) - \frac{3}{4}\right]\mathbb{p}(m_f, m_i) + \sqrt{\frac{18}{25}}\mathbb{f}(m_f, m_i), \\ \mathbb{c}_2(m_f, m_i) &= \sqrt{\frac{2}{175}}\left[I(I+1) - \frac{3}{4}\right]\mathbb{p}(m_f, m_i) - \frac{6}{\sqrt{35}}\mathbb{f}(m_f, m_i), \\ \mathbb{c}_4(m_f, m_i) &= -\sqrt{\frac{18}{875}}\left[I(I+1) - \frac{3}{4}\right]\mathbb{p}(m_f, m_i) - \frac{17}{\sqrt{175}}\mathbb{f}(m_f, m_i),\end{aligned} \tag{20.4}$$

where $\mathbb{p}(m_f, m_i)$ and $\mathbb{f}(m_f, m_i)$ are single nucleus spin transition functions described before, and I is the spin quantum number.

Parameters

- `mi`: The quantum number associated with the quantized initial energy level.
- `mf`: The quantum number associated with the quantized final energy level.
- `spin`: The spin quantum number, I.
- `cl_value`: A pointer to an array of size 3 where the spin transition functions, $\mathbb{c}_L$, will be stored ordered according to $L = [0, 2, 4]$.

20.1.2 Two weakly coupled nuclei spin transition functions

The weakly coupled spin transition function for $|m_{i_I}, m_{i_S}\rangle \rightarrow |m_{f_I}, m_{f_S}\rangle$ transition. Here, the subscript I and S denotes the two weakly coupled spins.

double `STF_dIS(const double mIf, const double mIi, const double mSf, const double mSi)`

The $\mathbb{d}_{IS}$ spin transition symmetry function.

Two weakly coupled nuclei spin transition function from the irreducible spherical tensors is given as

$$\begin{aligned}\mathbb{d}_{IS}(m_{f_I}, m_{f_S}, m_{i_I}, m_{i_S}) &= \left\langle m_{f_I} m_{f_S} | \hat{T}_{10}(I)\ \hat{T}_{10}(S) | m_{f_I} m_{f_S} \right\rangle - \left\langle m_{i_I} m_{i_S} | \hat{T}_{10}(I)\ \hat{T}_{10}(S) | m_{i_I} m_{i_S} \right\rangle \\ &= m_{f_I} m_{f_S} - m_{i_I} m_{i_S},\end{aligned} \tag{20.5}$$

where $\hat{T}_{10}(I)$ and $\hat{T}_{10}(S)$ are the irreducible first-rank spherical tensor operators in the rotating tilted frame for spin I and S, respectively.

Return The spin transition symmetry function $\mathbb{d}_{IS}$.

Parameters

- `mIf`: The quantum number associated with the quantized final energy state corresponding to spin I.
- `mSf`: The quantum number associated with the quantized final energy state corresponding to spin S.
- `mIi`: The quantum number associated with the quantized initial energy state corresponding to spin I.
- `mSi`: The quantum number associated with the quantized initial energy state corresponding to spin S.

20.2 Scaled spatial orientation tensors (sSOT), $\varsigma^{(k)}_{L,n}$

See also:

Scaled spatial orientation tensor components in PAS (page 276)

20.2.1 Single nucleus spatial orientation tensors

First-order Nuclear shielding

void `sSOT_1st_order_nuclear_shielding_tensor_components(double *restrict R_0, void *restrict R_2, const double omega_0_delta_iso_in_Hz, const double omega_0_zeta_sigma_in_Hz, const double eta, const double *Theta)`

The scaled spatial orientation tensors (sSOT) from the first-order perturbation expansion of the nuclear shielding Hamiltonian, in the principal axis system (PAS), include contributions from the zeroth and second-rank irreducible tensors which follow,

$$\left.\begin{aligned}\varsigma^{(\sigma)}_{0,0} &= \omega_0 \delta_{\text{iso}}\end{aligned}\right\} \text{Rank-0}, \tag{20.6}$$

$$\left.\begin{aligned}\varsigma^{(\sigma)}_{2,0} &= -\omega_0 \zeta_\sigma, \\ \varsigma^{(\sigma)}_{2,\pm 1} &= 0, \\ \varsigma^{(\sigma)}_{2,\pm 2} &= \frac{1}{\sqrt{6}} \omega_0 \eta_\sigma \zeta_\sigma,\end{aligned}\right\} \text{Rank-2}, \tag{20.7}$$

where σ_{iso} is the isotropic nuclear shielding, and ζ_σ, η_σ are the shielding anisotropy and asymmetry parameters from the symmetric second-rank irreducible nuclear shielding tensor defined using Haeberlen convention. Here,

$\omega_0 = -\gamma_I B_0$ is the Larmor frequency where, γ_I and B_0 are the gyromagnetic ratio of the nucleus and the magnetic flux density of the external magnetic field, respectively.

For non-zero Euler angles, $\Theta = [\alpha, \beta, \gamma]$, Wigner rotation of $\varsigma^{(\sigma)}_{2,n}$ is applied following,

$$\mathcal{R}'^{(\sigma)}_{2,n}(\Theta) = \sum_{m=-2}^{2} D^2_{m,n}(\Theta)\varsigma^{(\sigma)}_{2,n}, \tag{20.8}$$

where $\mathcal{R}'^{(\sigma)}_{2,n}(\Theta)$ are the tensor components in the frame defined by the Euler angles Θ.

Note

- The method accepts frequency physical quantities, that is, $\omega_0\delta_{\text{iso}}/2\pi$ and $\omega_0\zeta_\sigma/2\pi$, as the isotropic chemical shift and nuclear shielding anisotropy, respectively.
- When $\Theta = [0, 0, 0]$, $\mathcal{R}'^{(\sigma)}_{2,n}(\Theta) = \varsigma^{(\sigma)}_{2,n}$ where $n \in [-2, 2]$.
- $\mathcal{R}'^{(\sigma)}_{0,0}(\Theta) = \varsigma^{(\sigma)}_{0,0} \quad \forall\ \Theta$.
- The method returns $\mathcal{R}'^{(\sigma)}_{0,0}(\Theta)/2\pi$ and $\mathcal{R}'^{(\sigma)}_{2,n}(\Theta)/2\pi$, that is, in **units of frequency**.

Parameters

- `R_0`: A pointer to an array of length 1, where the components of the zeroth-rank irreducible tensor, $\mathcal{R}'^{(\sigma)}_{0,0}(\Theta)/2\pi$, is stored.
- `R_2`: A pointer to a complex array of length 5, where the components of the second-rank irreducible tensor, $\mathcal{R}'^{(\sigma)}_{2,n}(\Theta)/2\pi$, is stored ordered as $\left[\mathcal{R}'^{(\sigma)}_{2,n}(\Theta)/2\pi\right]^{2}_{n=-2}$.
- `omega_0_delta_iso_in_Hz`: The isotropic chemical shift in Hz, $\omega_0\sigma_{\text{iso}}/2\pi$.
- `omega_0_zeta_sigma_in_Hz`: The shielding anisotropy in Hz, $\omega_0\zeta_\sigma/2\pi$.
- `eta`: The shielding asymmetry, $\eta_\sigma \in [0, 1]$.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.

First-order Electric Quadrupole

void `sSOT_1st_order_electric_quadrupole_tensor_components`(void *restrict R_2, const double spin, const double Cq_in_Hz, const double eta, const double *Theta)

The scaled spatial orientation tensors (sSOT) from the first-order perturbation expansion of the electric quadrupole Hamiltonian, in the principal axis system (PAS), include contributions from the second-rank irreducible tensor which follow,

$$\left.\begin{aligned} \varsigma^{(q)}_{2,0} &= \frac{1}{\sqrt{6}}\omega_q, \\ \varsigma^{(q)}_{2,\pm1} &= 0, \\ \varsigma^{(q)}_{2,\pm2} &= -\frac{1}{6}\eta_q\omega_q, \end{aligned}\right\} \text{Rank-2}, \tag{20.9}$$

where $\omega_q = \frac{6\pi C_q}{2I(2I-1)}$ is the quadrupole splitting frequency, and η_q is the quadrupole asymmetry parameter. Here, I is the spin quantum number of the quadrupole nucleus, and C_q is the quadrupole coupling constant.

As before, for non-zero Euler angles, $\Theta = [\alpha, \beta, \gamma]$, a Wigner rotation of $\varsigma^{(q)}_{2,n}$ is applied following,

$$\mathcal{R}'^{(q)}_{2,n}(\Theta) = \sum_{m=-2}^{2} D^2_{m,n}(\Theta)\varsigma^{(q)}_{2,n}. \tag{20.10}$$

where $\mathcal{R}'^{(q)}_{2,n}(\Theta)$ are the tensor components in the frame defined by the Euler angles Θ.

Note

- When $\Theta = [0, 0, 0]$, $\mathcal{R}'^{(q)}_{2,n}(\Theta) = \varsigma^{(q)}_{2,n}$ where $n \in [-2, 2]$.
- The method returns $\mathcal{R}'^{(q)}_{2,0}(\Theta)/2\pi$, that is, in **units of frequency**.

Parameters

- `R_2`: A pointer to a complex array of length 5, where the components of the second-rank irreducible tensor, $\mathcal{R}'^{(q)}_{2,n}(\Theta)/2\pi$, is stored ordered as $\left[\mathcal{R}'^{(q)}_{2,n}(\Theta)/2\pi\right]^{2}_{n=-2}$.
- `spin`: The spin quantum number, I.
- `Cq_in_Hz`: The quadrupole coupling constant, C_q, in Hz.
- `eta`: The quadrupole asymmetry parameter, $\eta_q \in [0, 1]$.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.

Second-order Electric Quadrupole

void `sSOT_2nd_order_electric_quadrupole_tensor_components`(double *restrict R_0, void *restrict R_2, void *restrict R_4, const double spin, const double v0_in_Hz, const double Cq_in_Hz, const double eta, const double *Theta)

The scaled spatial orientation tensors (sSOT) from the second-order perturbation expansion of the electric quadrupole Hamiltonian, in the principal axis system (PAS), include contributions from the zeroth, second, and fourth-rank irreducible tensors which follow,

$$\left.\begin{aligned} \varsigma^{(qq)}_{0,0} &= \frac{\omega_q^2}{\omega_0}\frac{1}{6\sqrt{5}}\left(\frac{\eta_q^2}{3}+1\right) \end{aligned}\right\} \text{Rank-0}, \tag{20.11}$$

$$\left.\begin{aligned} \varsigma^{(qq)}_{2,0} &= \frac{\omega_q^2}{\omega_0}\frac{\sqrt{2}}{6\sqrt{7}}\left(\frac{\eta_q^2}{3}-1\right), \\ \varsigma^{(qq)}_{2,\pm1} &= 0, \\ \varsigma^{(qq)}_{2,\pm2} &= -\frac{\omega_q^2}{\omega_0}\frac{1}{3\sqrt{21}}\eta_q, \end{aligned}\right\} \text{Rank-2}, \tag{20.12}$$

$$\left.\begin{aligned} \varsigma^{(qq)}_{4,0} &= \frac{\omega_q^2}{\omega_0}\frac{1}{\sqrt{70}}\left(\frac{\eta_q^2}{18}+1\right), \\ \varsigma^{(qq)}_{4,\pm1} &= 0, \\ \varsigma^{(qq)}_{4,\pm2} &= -\frac{\omega_q^2}{\omega_0}\frac{1}{6\sqrt{7}}\eta_q, \\ \varsigma^{(qq)}_{4,\pm3} &= 0, \\ \varsigma^{(qq)}_{4,\pm4} &= \frac{\omega_q^2}{\omega_0}\frac{1}{36}\eta_q^2, \end{aligned}\right\} \text{Rank-4}, \tag{20.13}$$

where $\omega_q = \frac{6\pi C_q}{2I(2I-1)}$ is the quadrupole splitting frequency, ω_0 is the Larmor angular frequency, and η_q is the quadrupole asymmetry parameter. Here, I is the spin quantum number, and C_q is the quadrupole coupling constant.

For non-zero Euler angles, $\Theta = [\alpha, \beta, \gamma]$, Wigner rotation of $\varsigma^{(qq)}_{2,n}$ and $\varsigma^{(qq)}_{4,n}$ are applied following,

$$\begin{aligned}\mathcal{R}'^{(qq)}_{2,n}(\Theta) &= \sum_{m=-2}^{2} D^2_{m,n}(\Theta)\varsigma^{(qq)}_{2,n},\\ \mathcal{R}'^{(qq)}_{4,n}(\Theta) &= \sum_{m=-4}^{4} D^4_{m,n}(\Theta)\varsigma^{(qq)}_{4,n},\end{aligned} \tag{20.14}$$

where $\mathcal{R}'^{(qq)}_{2,n}(\Theta)$ and $\mathcal{R}'^{(qq)}_{4,n}(\Theta)$ are the second and fourth-rank tensor components in the frame defined by the Euler angles Θ, respectively.

Note

- When $\Theta = [0, 0, 0]$, $\mathcal{R}'^{(qq)}_{2,n}(\Theta) = \varsigma^{(qq)}_{2,n}$ where $n \in [-2, 2]$.
- When $\Theta = [0, 0, 0]$, $\mathcal{R}'^{(qq)}_{4,n}(\Theta) = \varsigma^{(qq)}_{4,n}$ where $n \in [-4, 4]$.
- $\mathcal{R}'^{(qq)}_{0,0}(\Theta) = \varsigma^{(qq)}_{0,0} \quad \forall\ \Theta$.
- The method returns $\mathcal{R}'^{(qq)}_{0,0}(\Theta)/2\pi$, $\mathcal{R}'^{(qq)}_{2,n}(\Theta)/2\pi$, and $\mathcal{R}'^{(qq)}_{4,n}(\Theta)/2\pi$, that is, in **units of frequency**.

Parameters

- `R_0`: A pointer to an array of length 1 where the zeroth-rank irreducible tensor, $\mathcal{R}'^{(qq)}_{0,0}(\Theta)/2\pi$, will be stored.
- `R_2`: A pointer to a complex array of length 5 where the second-rank irreducible tensor, $\mathcal{R}'^{(qq)}_{2,n}(\Theta)/2\pi$, will be stored ordered according to $\left[\mathcal{R}'^{(qq)}_{2,n}(\Theta)/2\pi\right]^{2}_{n=-2}$.
- `R_4`: A pointer to a complex array of length 9 where the fourth-rank irreducible tensor, $\mathcal{R}'^{(qq)}_{4,n}(\Theta)/2\pi$, will be stored ordered according to $\left[\mathcal{R}'^{(qq)}_{4,n}(\Theta)/2\pi\right]^{4}_{n=-4}$.
- `spin`: The spin quantum number, I.
- `v0_in_Hz`: The Larmor frequency, $\omega_0/2\pi$, in Hz.
- `Cq_in_Hz`: The quadrupole coupling constant, C_q, in Hz.
- `eta`: The quadrupole asymmetry parameter, $\eta_q \in [0, 1]$.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.

First-order J-coupling (weak coupling limit)

void `sSOT_1st_order_weakly_coupled_J_tensor_components`(double *restrict R_0, void *restrict R_2, const double J_iso_in_Hz, const double J_aniso_in_Hz, const double J_eta, const double *Theta)

The scaled spatial orientation tensors (sSOT) from the first-order perturbation expansion of the J-coupling Hamiltonian under weak-coupling limit, in the principal axis system (PAS), include contributions from the zeroth and second-rank irreducible tensors which follow,

$$\left.\varsigma^{(J)}_{0,0} = 2\pi J_{\text{iso}}\right\} \text{Rank-0}, \tag{20.15}$$

$$\left.\begin{aligned}\varsigma^{(J)}_{2,0} &= 2\pi\zeta_J,\\ \varsigma^{(J)}_{2,\pm1} &= 0,\\ \varsigma^{(J)}_{2,\pm2} &= 2\pi\frac{1}{\sqrt{6}}\eta_J\zeta_J,\end{aligned}\right\} \text{Rank-2}, \tag{20.16}$$

where J_{iso} is the isotropic J-coupling, and ζ_J, η_J are the J-coupling tensor anisotropy and asymmetry parameters from the symmetric second-rank irreducible J tensor, defined using Haeberlen convention.

For non-zero Euler angles, $\Theta = [\alpha, \beta, \gamma]$, Wigner rotation of $\varsigma^{(J)}_{2,n}$ is applied following,

$$\mathcal{R}'^{(J)}_{2,n}(\Theta) = \sum_{m=-2}^{2} D^2_{m,n}(\Theta)\varsigma^{(J)}_{2,n}, \tag{20.17}$$

where $\mathcal{R}'^{(J)}_{2,n}(\Theta)$ are the tensor components in the frame defined by the Euler angles, Θ.

Note

- When $\Theta = [0, 0, 0]$, $\mathcal{R}'^{(J)}_{2,n}(\Theta) = \varsigma^{(J)}_{2,n}$ where $n \in [-2, 2]$.
- $\mathcal{R}'^{(J)}_{0,0}(\Theta) = \varsigma^{(J)}_{0,0} \quad \forall\ \Theta$.
- The method returns $\mathcal{R}'^{(J)}_{0,0}(\Theta)/2\pi$ and $\mathcal{R}'^{(J)}_{2,n}(\Theta)/2\pi$, that is, in **units of frequency**.

Parameters

- `R_0`: A pointer to an array of length 1, where the zeroth-rank irreducible tensor, $\mathcal{R}'^{(J)}_{0,0}(\Theta)/2\pi$, is stored.
- `R_2`: A pointer to a complex array of length 5, where the second-rank irreducible tensor, $\mathcal{R}'^{(J)}_{2,n}(\Theta)/2\pi$, is stored ordered as $\left[\mathcal{R}'^{(J)}_{2,n}(\Theta)/2\pi\right]^2_{n=-2}$.
- `J_iso_in_Hz`: The isotropic J-coupling, J_{iso}, in Hz.
- `J_aniso_in_Hz`: The J-coupling anisotropy, ζ_J, in Hz.
- `J_eta`: The J-coupling asymmetry, $\eta_J \in [0, 1]$.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.

First-order dipolar-coupling (weak coupling limit)

void `sSOT_1st_order_weakly_coupled_dipolar_tensor_components`(void *restrict R_2, const double D_in_Hz, const double *Theta)

The scaled spatial orientation tensors (sSOT) from the first-order perturbation expansion of the dipolar-coupling Hamiltonian under weak-coupling limit, in the principal axis system (PAS), include contributions from the second-rank irreducible tensors which follow,

$$\left.\begin{aligned} \varsigma^{(d)}_{2,0} &= 4\pi D, \\ \varsigma^{(d)}_{2,\pm1} &= 0, \\ \varsigma^{(d)}_{2,\pm2} &= 0, \end{aligned}\right\} \text{Rank-2}, \tag{20.18}$$

where D is the dipolar-coupling.

For non-zero Euler angles, $\Theta = [\alpha, \beta, \gamma]$, Wigner rotation of $\varsigma^{(d)}_{2,n}$ is applied following,

$$\mathcal{R}'^{(d)}_{2,n}(\Theta) = \sum_{m=-2}^{2} D^2_{m,n}(\Theta)\varsigma^{(d)}_{2,n}, \tag{20.19}$$

where $\mathcal{R}'^{(d)}_{2,n}(\Theta)$ are the tensor components in the frame defined by the Euler angles, Θ.

Note

- When $\Theta = [0, 0, 0]$, $\mathcal{R}'^{(d)}_{2,n}(\Theta) = \varsigma^{(d)}_{2,n}$ where $n \in [-2, 2]$.
- The method returns $\mathcal{R}'^{(d)}_{2,n}(\Theta)/2\pi$, that is, in **units of frequency**.

Parameters

- `R_2`: A pointer to a complex array of length 5, where the second-rank irreducible tensor, $\mathcal{R}'^{(d)}_{2,n}(\Theta)/2\pi$, is stored ordered as $\left[\mathcal{R}'^{(d)}_{2,n}(\Theta)/2\pi\right]^2_{n=-2}$.

- `D_in_Hz`: The dipolar coupling, D, in Hz.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.

20.3 Frequency Tensors (FT), $\Lambda^{(k)}_{L,n}(i,j)$

See also:

Frequency component functions in PAS (page 281), *Scaled spatial orientation functions (SOF)* (page 355), *Spin transition functions (STF)* (page 353)

20.3.1 Single nucleus frequency tensor components

First-order Nuclear shielding

void `FCF_1st_order_nuclear_shielding_tensor_components`(double *restrict Lambda_0, void *restrict Lambda_2, const double omega_0_delta_iso_in_Hz, const double omega_0_zeta_sigma_in_Hz, const double eta, const double *Theta, const float mf, const float mi)

The frequency tensors (FT) components from the first-order perturbation expansion of the nuclear shielding Hamiltonian, in a given frame, $\mathcal{F}$, described by the Euler angles $\Theta = [\alpha, \beta, \gamma]$ are

$$\begin{aligned} \Lambda'^{(\sigma)}_{0,0} &= \mathcal{R}'^{(\sigma)}_{0,0}(\Theta) \;\; \mathbb{p}(i,j), \text{ and} \\ \Lambda'^{(\sigma)}_{2,n} &= \mathcal{R}'^{(\sigma)}_{2,n}(\Theta) \;\; \mathbb{p}(i,j), \end{aligned} \tag{20.20}$$

where $\mathcal{R}'^{(\sigma)}_{0,0}(\Theta)$ and $\mathcal{R}'^{(\sigma)}_{2,n}(\Theta)$ are the spatial orientation functions in frame $\mathcal{F}$, and $\mathbb{p}(i,j)$ is the spin transition function for $|i\rangle \rightarrow |j\rangle$ transition.

Parameters

- `Lambda_0`: A pointer to an array of length 1, where the frequency component from $\Lambda'^{(\sigma)}_{0,0}$ is stored.
- `Lambda_2`: A pointer to a complex array of length 5, where the frequency components from $\Lambda'^{(\sigma)}_{2,n}$ is stored ordered as $\left[\Lambda'^{(\sigma)}_{2,n}\right]^{2}_{n=-2}$.
- `omega_0_delta_iso_in_Hz`: The isotropic chemical shift in Hz ($\omega_0\delta_{\text{iso}}/2\pi$).
- `omega_0_zeta_sigma_in_Hz`: The shielding anisotropy quantity in Hz ($\omega_0\zeta_\sigma/2\pi$) defined using Haeberlen convention.
- `eta`: The shielding asymmetry, $\eta_\sigma \in [-1, 1]$, defined using Haeberlen convention.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.
- `mf`: The spin quantum number of the final energy state.
- `mi`: The spin quantum number of the initial energy state.

First-order Electric Quadrupole

void `FCF_1st_order_electric_quadrupole_tensor_components`(void *restrict Lambda_2, const double spin, const double Cq_in_Hz, const double eta, const double *Theta, const float mf, const float mi)

The frequency tensor (FT) components from the first-order perturbation expansion of electric quadrupole Hamiltonian, in a given frame, $\mathcal{F}$, described by the Euler angles $\Theta = [\alpha, \beta, \gamma]$ are

$$\Lambda'^{(q)}_{2,n} = \mathcal{R}'^{(q)}_{2,n}(\Theta) \;\; \mathbb{d}(i,j), \tag{20.21}$$

where $\mathcal{R}^{(q)}_{2,n}(\Theta)$ are the spatial orientation functions in frame $\mathcal{F}$, and $\mathbb{d}(i,j)$ is the spin transition function for $|i\rangle \rightarrow |j\rangle$ transition.

Parameters

- `Lambda_2`: A pointer to a complex array of length 5, where the frequency components from $\Lambda'^{(q)}_{2,n}$ is stored ordered as $\left[\Lambda'^{(q)}_{2,n}\right]^{2}_{n=-2}$.
- `spin`: The spin quantum number, I.
- `Cq_in_Hz`: The quadrupole coupling constant, C_q, in Hz.
- `eta`: The quadrupole asymmetry parameter, $\eta_q \in [0, 1]$.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.
- `mf`: The spin quantum number of the final energy state.
- `mi`: The spin quantum number of the initial energy state.

Second-order Electric Quadrupole

void `FCF_2nd_order_electric_quadrupole_tensor_components`(double *restrict Lambda_0, void *restrict Lambda_2, void *restrict Lambda_4, const double spin, const double v0_in_Hz, const double Cq_in_Hz, const double eta, const double *Theta, const float mf, const float mi)

The frequency tensor (FCF) components from the second-order perturbation expansion of electric quadrupole Hamiltonian, in a given frame, $\mathcal{F}$, described by the Euler angles $\Theta = [\alpha, \beta, \gamma]$, are

$$\begin{aligned}
\Lambda'^{(qq)}_{0,0} &= \mathcal{R}'^{(qq)}_{0,0}(\Theta) \;\; \mathbb{c}_0(i,j), \\
\Lambda'^{(qq)}_{2,n} &= \mathcal{R}'^{(qq)}_{2,n}(\Theta) \;\; \mathbb{c}_2(i,j), \text{ and} \\
\Lambda'^{(qq)}_{4,n} &= \mathcal{R}'^{(qq)}_{4,n}(\Theta) \;\; \mathbb{c}_4(i,j),
\end{aligned} \tag{20.22}$$

where $\mathcal{R}'^{(qq)}_{0,0}(\Theta)$, $\mathcal{R}'^{(qq)}_{2,n}(\Theta)$, and, $\mathcal{R}'^{(qq)}_{4,n}(\Theta)$ are the spatial orientation functions in frame $\mathcal{F}$, and $\mathbb{c}_k(i,j)$ are the composite spin transition functions for $|i\rangle \rightarrow |j\rangle$ transition.

Parameters

- `Lambda_0`: A pointer to an array of length 1, where the frequency component from $\Lambda'^{(qq)}_{0,0}$ is stored.
- `Lambda_2`: A pointer to a complex array of length 5, where the frequency components from $\Lambda^{(qq)}_{2,n}$ are stored ordered as $\left[\Lambda'^{(qq)}_{2,n}\right]^{2}_{n=-2}$.
- `Lambda_4`: A pointer to a complex array of length 9, where the frequency components from $\Lambda'^{(qq)}_{4,n}$ are stored ordered as $\left[\Lambda'^{(qq)}_{4,n}\right]^{4}_{n=-4}$.
- `spin`: The spin quantum number, I.

- `Cq_in_Hz`: The quadrupole coupling constant, C_q, in Hz.
- `eta`: The quadrupole asymmetry parameter, $\eta_q \in [0, 1]$.
- `v0_in_Hz`: The Larmor frequency, ν_0, in Hz.
- `Theta`: A pointer to an array of Euler angles, in radians, of length 3, ordered as $[\alpha, \beta, \gamma]$.
- `mf`: The spin quantum number of the final energy state.
- `mi`: The spin quantum number of the initial energy state.

20.3.2 Two coupled nucleus frequency tensor components

First-order J-coupling (weak coupling limit)

void `FCF_1st_order_weak_J_coupling_tensor_components`(double *restrict Lambda_0, void *restrict Lambda_2, const double J_iso_in_Hz, const double J_aniso_in_Hz, const double J_eta, const double *Theta, const float mIf, const float mIi, const float mSf, const float mSi)

The frequency tensor (FT) components from the first-order perturbation expansion of the J-coupling Hamiltonian (weak coupling limit), in a given frame, $\mathcal{F}$, described by the Euler angles $\Theta = [\alpha, \beta, \gamma]$ are

$$\begin{aligned} \Lambda'^{(J)}_{0,0} &= \mathcal{R}'^{(J)}_{0,0}(\Theta) \;\; \mathbb{d}_{IS}(m_{i_I}, m_{i_S}, m_{f_I}, m_{f_S}), \text{ and} \\ \Lambda'^{(J)}_{2,n} &= \mathcal{R}'^{(J)}_{2,n}(\Theta) \;\; \mathbb{d}_{IS}(m_{i_I}, m_{i_S}, m_{f_I}, m_{f_S}), \end{aligned} \tag{20.23}$$

where $\mathcal{R}'^{(J)}_{0,0}(\Theta)$ and $\mathcal{R}'^{(J)}_{2,n}(\Theta)$ are the spatial orientation functions in frame $\mathcal{F}$, and $\mathbb{d}_{IS}(m_{i_I}, m_{i_S}, m_{f_I}, m_{f_S})$ is the spin transition function for $|m_{i_I}, m_{i_S}\rangle \rightarrow |m_{f_I}, m_{f_S}\rangle$ transition.

Parameters

- `Lambda_0`: A pointer to an array of length 1, where the frequency component from $\Lambda'^{(J)}_{0,0}$ is stored.
- `Lambda_2`: A pointer to a complex array of length 5, where the frequency components from $\Lambda'^{(J)}_{2,n}$ is stored ordered as $\left[\Lambda'^{(J)}_{2,n}\right]_{n=-2}^{2}$.
- `J_iso_in_Hz`: The isotropic J-coupling, J_{iso}, in Hz.
- `J_aniso_in_Hz`: The J-coupling anisotropy, ζ_J, in Hz and defined using Haeberlen convention.
- `J_eta`: The J-coupling anisotropy asymmetry parameter, $\eta_J \in [-1, 1]$, defined using Haeberlen convention.
- `Theta`: A pointer to an array of Euler angles of length 3 ordered as $[\alpha, \beta, \gamma]$.
- `mIf`: The spin quantum number of the final energy state of site I.
- `mIi`: The spin quantum number of the initial energy state of site I.
- `mSf`: The spin quantum number of the final energy state of site S.
- `mSi`: The spin quantum number of the initial energy state of site S.

First-order dipolar-coupling (weak coupling limit)

void `FCF_1st_order_weak_dipolar_coupling_tensor_components`(void *restrict Lambda_2, const double D_in_Hz, const double *Theta, const float mIf, const float mIi, const float mSf, const float mSi)

The frequency tensor (FT) components from the first-order perturbation expansion of the direct dipolar coupling Hamiltonian (weak coupling limit), in a given frame, $\mathcal{F}$, described by the Euler angles $\Theta = [\alpha, \beta, \gamma]$ are

$$\Lambda'^{(d)}_{2,n} = \mathcal{R}'^{(d)}_{2,n}(\Theta) \;\; \mathbb{d}_{IS}(m_{i_I}, m_{i_S}, m_{f_I}, m_{f_S}), \tag{20.24}$$

where $\mathcal{R}'^{(d)}_{2,n}(\Theta)$ are the spatial orientation functions in frame $\mathcal{F}$, and $\mathbb{d}_{IS}(m_{i_I}, m_{i_S}, m_{f_I}, m_{f_S})$ is the spin transition function for $|m_{i_I}, m_{i_S}\rangle \rightarrow |m_{f_I}, m_{f_S}\rangle$ transition.

Parameters

- `Lambda_2`: A pointer to a complex array of length 5, where the frequency components from $\Lambda'^{(d)}_{2,n}$ is stored ordered as $\left[\Lambda'^{(d)}_{2,n}\right]_{n=-2}^{2}$.
- `D_in_Hz`: The dipolar coupling, D, in Hz.
- `Theta`: A pointer to an array of Euler angles of length 3 ordered as $[\alpha, \beta, \gamma]$.
- `mIf`: The spin quantum number of the final energy state of site I.
- `mIi`: The spin quantum number of the initial energy state of site I.
- `mSf`: The spin quantum number of the final energy state of site S.
- `mSi`: The spin quantum number of the initial energy state of site S.

Part VII

Project details

CHAPTER

TWENTYONE

CHANGELOG

21.1 v0.6.0

21.1.1 What's new

- ⋆ Improved simulation performance. ⋆ See our *Performance benchmark* (page 271).
- Simulation of one-dimensional spectra of coupled spin systems. The frequency contributions from the coupled sites include weak J-couplings and weak dipolar couplings.
- New *Coupling* (page 306) class.
- Added a new `Sites` class that holds a list of Site objects. The Sites class method, `to_pd()`, exports the sites as a pandas data frame.
- A new method, `sites()`, is added to the Simulator object, which returns a list of unique `Sites` objects within the Simulator object across multiple spin systems.
- Added three new arguments to the `single_site_system_generator()` method, 'site_labels', 'site_names', and 'site_descriptions'.

21.1.2 Changes

- The *get_isotopes()* (page 300) method from the SpinSystem object, will now return *Isotope* (page 336) objects by default. Use the *symbol=True* argument of the method to get a list of string isotopes.
- The `to_freq_dict()` function is deprecated.
- The *D* symmetry of *transition_query* attribute from *Method2D* method is now None by default.
- *BlochDecayCTSpectrum* is an alias for *BlochDecayCentralTransitionSpectrum* class.

21.1.3 Bug fixes

- Fixed a bug related to *get_spectral_dimensions()* utility method in cases when CSDM dimension objects have negative increment.
- Fixed a bug resulting in the non-conserved spectral area after a Gaussian apodization.
- Fixed a bug in Gaussian apodization, which raised an error when the FWHM argument is a scalar.
- Fixed bug causing multi-dataset fit to fail.

21.2 v0.5.1

21.2.1 Bug fixes

- Fixed a bug that was causing incorrect spectral binning when the frequency contribution is pure isotropic.

21.2.2 Other changes

- The `to_dict_with_units()` method is deprecated and is replaced with `json()`
- The `json()` function returns a python dictionary object with minimal required keywords, where the event keys are globally serialized at the root method object. In the case where the event key value is different from the global value, the respective key is serialized within the event object.
- The `json()` function will no longer serialize the *transition_ query* objects for the named objects.

21.3 v0.5.0

21.3.1 What's new

- ⋆ Improved simulation performance. ⋆ See our *Performance benchmark* (page 271).

The update introduces various two-dimensional methods for simulating NMR spectrum.

- Introduces a generic one-dimensional method, *Method1D* (page 319).
- Introduces a generic two-dimensional method, *Method2D* (page 322).
- Specialized two-dimensional methods for multi-quantum variable-angle spinning with build-in affine transformations.
  - *ThreeQ_VAS* (page 325),
  - *FiveQ_VAS* (page 326),
  - *SevenQ_VAS* (page 327)
- Specialized two-dimensional methods for satellite-transition variable-angle spinning with build-in affine transformations.
  - *ST1_VAS* (page 329),
  - *ST2_VAS* (page 330),
- Specialized two-dimensional isotropic/anisotropic sideband correlation method, *SSB2D* (page 332).

21.3.2 Other changes

- The *get_transition_pathways()* (page 312) method no longer return a numpy array, instead a python list.

21.4 v0.4.0

21.4.1 What's new!

- ⋆ Improved simulation performance. ⋆ See our *Performance benchmark* (page 271).
- New *CzjzekDistribution* (page 347) and *ExtCzjzekDistribution* (page 348) classes for generating Czjzek and extended Czjzek second-rank symmetric tensor distribution models for use in simulating amorphous materials.
- New utility function, *single_site_system_generator()* (page 339), for generating a list of single-site spin systems from a 1D list/array of respective tensor parameters.

21.5 v0.3.0

21.5.1 What's new!

- ⋆ Improved simulation performance. ⋆ See our *Performance benchmark* (page 271).
- Removed the `Dimension` class and added a new `Method` class instead.
- New methods for simulating the NMR spectrum:
  - `BlochDecaySpectrum` and
  - `BlochDecayCentralTransitionSpectrum`.

  The Bloch decay spectrum method simulates all p=Δm=-1 transition pathways, while the Bloch decay central transition selective spectrum method simulates all transition pathways with p=Δm=-1 and d=0.
- New `Isotope`, `Transition`, and `ZeemanState` classes.
- Every class now includes a `reduced_dict()` method. The reduced_dict method returns a dictionary with minimal key-value pairs required to simulate the spectrum. Note, this may cause metadata loss, if any.
- Added a `label` and `description` attributes to the `Site` class.
- Added a new `label` attribute to the `SpinSystem` class.
- New `SignalProcessor` class for post-simulation signal processing.
- Improved usage of least-squares minimization using python LMFIT package.
- Added a new `get_spectral_dimensions` utility function to extract the spectral dimensions information from the CSDM object.

21.5.2 Bug fixes

- Fixed bug resulting from the rotation of the fourth rank tensor with non-zero euler angles.
- Fixed bug causing a change in the spectral area as the sampling points change. Now the area is constant.
- Fixed bug resulting in an incorrect spectrum when non-coincidental quad and shielding tensors are given.
- Fixed bug causing incorrect generation of transition pathways when multiple events are present.

21.5.3 Other changes

- Renamed the `decompose` attribute from the `ConfigSimulator` class to `decompose_spectrum`. The attribute is an enumeration with the following literals:
  - `none`: Computes a spectrum which is an integration of the spectra from all spin systems.
  - `spin_system`: Computes a series of spectra each corresponding to a single spin system.
- Renamed `Isotopomer` class to `SpinSystem`.
- Renamed `isotopomers` attribute from `Simulator` class to `spin_systems`.
- Renamed `dimensions` attribute from `Simulator` class to `methods`.
- Changed the default value of `name` and `description` attribute from the `SpinSystem` class from `""` to `None`.

21.6 v0.2.x

21.6.1 What's new!

- Added more isotopes to the simulator. Source NMR Tables (https://apps.apple.com/bn/app/nmr-tables/id1030899609?mt=12).
- Added two new keywords: *atomic_ number* and *quadrupole_ moment*.
- Added documentation for every class.
- Added examples for simulating NMR quadrupolar spectrum.

21.6.2 Bug fixes

- Fixed amplitude normalization. The spectral amplitude no longer change when the *integration_ density*, integration_volume`, or the *number_ of_ sidebands* attributes change.

21.6.3 Other changes

- Removed plotly-dash app to its own repository.
- Renamed the class `Spectrum` to `Dimension`

21.7 v0.1.3

- Fixed missing files from source tar.

21.8 v0.1.2

- Initial release on pypi.

21.9 v0.1.1

- Added solid state quadrupolar spectrum simulation.
- Added mrsimulator plotly-dash app.

21.10 v0.1.0

- Solid state chemical shift anisotropy spectrum simulation.

CHAPTER

TWENTYTWO

AUTHORS AND CREDITS

- Deepansh Srivastava (Ohio State University)
- Maxwell C. Venetos (UC Berkeley)
- Philip J. Grandinetti (Ohio State University)
- Shyam Dwaraknath (LBNL)
- Alexis McCarthy (Ohio State University)

CHAPTER

TWENTYTHREE

LICENSE

23.1 Mrsimulator License

CHAPTER

TWENTYFOUR

ACKNOWLEDGMENT

The development of the mrsimulator project was supported in part by the US National Science Foundation under Grant No. DIBBS OAC 1640899 and Grant No. CHE 1807922.

Part VIII

Reporting Bugs

The preferred location for submitting feature requests and bug reports is the Github issue tracker. Reports are also welcomed by directly contacting Deepansh Srivastava.

Discussions are welcome on Github discussion

INDEX

Q

R

S

T

U

Z