Getting Started in Python
(and MRSimulator)

P. J. Grandinetti*

L'Ohio State Univ.

NMR Winter School, 2024

*Email: grandinetti.1@osu.edu, web: www.grandinetti.org

Introduction to Python

Python is a high-level, interpreted programming language created by Guido van Rossum and first released in 1991.

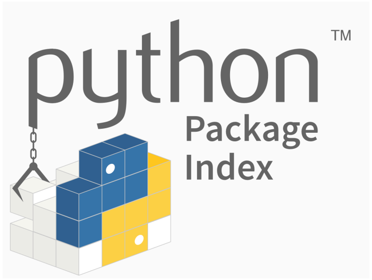
- Python is Interpreted: The interpreter processes Python at runtime. You do not need to compile your program before executing it.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports an Object-Oriented style or programming technique that encapsulates code within objects.
- Python is a great language for beginner-level programmers and supports developing a wide range of applications, from simple text processing to WWW browsers to games.

Popular Python Libraries for Data Analysis and Scientific Computing

- **Jupyter Notebook**: A web application that allows the creation and sharing of documents containing live code, equations, visualizations, and narrative text.
- **NumPy**: Provides support for arrays, matrices, and many mathematical functions.
- **Matplotlib**: A plotting library for creating static, animated, and interactive visualizations.
- **Pandas**: High-level data structures and functions designed for working with structured or tabular data.
- **SciPy**: Used for scientific and technical computing. It builds on NumPy and provides additional modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers, and more.
- **Scikit-learn**: A machine learning library featuring various classification, regression, and clustering algorithms.
- **TensorFlow and PyTorch**: Popular libraries for creating deep learning models.
- **Statsmodels**: Provides classes and functions for the estimation of many different statistical models and for conducting statistical tests and statistical data exploration.
- **Seaborn**: A statistical data visualization library based on Matplotlib.

PyPI: The Python Package Index

The Python Package Index (PyPI) is a software repository for the Python programming language.



- **Package Hosting**: PyPI helps you find and install software developed and shared by the Python community.
- **Tool Integration**: PyPI is compatible with all major package management and installation tools like pip and conda.
- **Wide Variety**: PyPI hosts many packages for various tasks, including web development, data analysis, machine learning, scientific computing, automation, and more.
- **Open Source Contribution**: PyPI allows developers to share their software and collaborate on open-source projects, making it a hub for Python development.

Dependency Hell: A Challenge in Software Use and Development

Dependency Hell occurs when a software application cannot access the additional programming libraries required to operate.

- **Conflicting Dependencies**: Different parts of project depend on incompatible versions of same library.
- **Nested Dependencies**: A dependency of your project has its own dependencies, which in turn have their own dependencies, and so on, leading to a complex and hard-to-manage tree of dependencies.
- **Missing Dependencies**: Project depends on library that is no longer maintained or available.
- **Environment Differences**: The project works in one environment but not in another due to differences in the versions of the dependencies installed in each environment.

Solution: use virtual environments to isolate your project's dependencies from those of other projects.

Each project can have its own dependencies, even if they require different versions of the same package. Python's built-in venv module allows you to create virtual environments, and there are also third-party tools like virtualenv and conda.

Download and Install Visual Studio Code (VSCode)

- **Language Support**: VSCode supports various programming languages with syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets, and more.
- **IntelliSense**: VSCode provides smart completions based on variable types, function definitions, and imported modules.
- **Git Integration**: VSCode has built-in Git support for version control to track changes, stage, commit, pull, and push to a remote repository.
- **Debugging**: VSCode has an integrated debugger, so you can set breakpoints, inspect variables, view call stack, and execute code step by step.
- **Extensions**: VSCode has a rich ecosystem of extensions for enhancing its functionality, including linters, debuggers, formatters, new themes, and language support.

- Install Extensions
  - Python, Pylance by Microsoft
  - Jupyter, Jupyter Cell Tags, Jupyter Kemap, Jupyter Slide Show by Microsoft
  - GitHub Pull Requests and Issues, Github Copilot, Github Copilot Chat

Download and Install Miniconda

Miniconda is a free minimal installer for conda, a package and environment management system.

- **Lightweight**: Miniconda is small and quickly installed. It provides only conda and its dependencies, making it suitable for environments with limited disk space or for users who do not need the full distribution of Anaconda.
- **Package Management**: Miniconda uses the Conda package manager like Anaconda. This lets you install packages from the Anaconda distribution and the Python Package Index (PyPI).
- **Environment Management**: Conda allows you to create separate environments containing files, packages, and their dependencies that will not interact with other environments.
- **Flexibility**: With Miniconda, you can create a minimal, self-contained Python installation and then use the Conda command to install only the necessary packages.

GitHub: A Platform for Collaborative Coding

GitHub is a web-based hosting service for version control using Git. It is mainly used for computer code.

- **Version Control**: GitHub hosts your code and keeps track of all changes made to every file.
- **Collaboration**: GitHub allows multiple people to work on the same project efficiently. It provides features like issues and pull requests to discuss and review changes before merging.
- **Open Source Contribution**: Many open-source projects use GitHub to accept contributions, report issues, and disseminate information.
- **Integration**: GitHub integrates with many services and tools, such as continuous integration services and project management tools.
- **GitHub Pages**: GitHub allows you to host websites directly from a GitHub repository.

Sign up for Github

- Use your university email address for GitHub Student Developer Pack.
- Download and install Github Desktop

ChatGPT: A Language Model by OpenAI

ChatGPT is a large-scale, AI-based language model developed by OpenAI.

- **Generative Pre-training Transformer (GPT)**: ChatGPT is based on the GPT model, which is trained on a diverse range of internet text.
- **Conversational Agent**: ChatGPT is designed to generate human-like text based on the input it's given, making it suitable for tasks like drafting emails, writing code, creating written content, tutoring, translating languages, simulating characters for video games, and more.
- **Fine-tuning**: After pre-training, ChatGPT is fine-tuned with human supervision on specific tasks for better performance.
- **Applications**: ChatGPT has been used to build applications like AI Dungeon, a text-based adventure game, and GitHub Copilot, an AI pair programmer.

Use it as your personal Python tutor, and to write simple scripts.

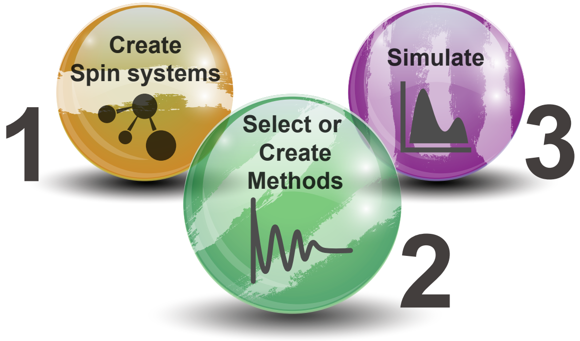
Breakout Session: Getting Started with Python

1. Signup for GitHub, and install Github Desktop
2. Install VSCode and extensions
3. Install Miniconda
4. Create a virtual environment
5. Create a Jupyter Notebook and run a simple Python script
6. Ask ChatGPT for help writing Python scripts,
   e.g., ”Write a Python script to fit my NMR inversion recovery data.”

How can I get started with mrsimulator?

- pip install mrsimulator
- pip install pydantic==1.10 → *temporary fix until mrsimulator 0.8 is released.*

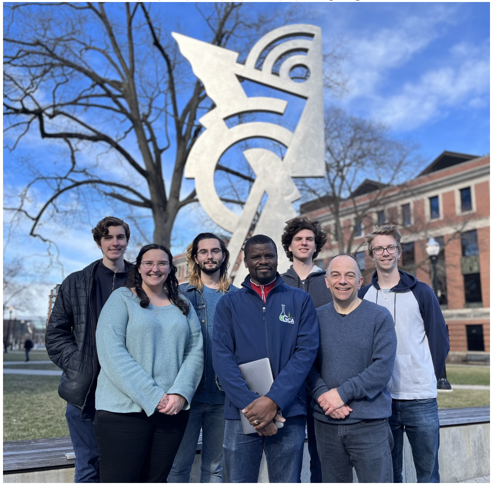
A Python toolbox for simulating solid-state NMR spectra



- inside Jupyter Notebook*: "!pip install mrsimulator" and "!pip install pydantic==1.10"

Deepansh Srivastava
Hyperfine

Grandinetti Lab 2023



Calvin Bostleman,
Lexi McCarthy,
Zack Boothe,
Modeste Tegomoh,
Matthew Giammar,
Dustin Pigg.

Email: grandinetti.1@osu.edu https://www.grandinetti.org/Software

mrsimulator Script Example - 1. Create SpinSystem - Sites

```python
# Import the Site and SymmetricTensor classes
from mrsimulator import Site
from mrsimulator.spin_system.tensors import SymmetricTensor

# Create the Site objects
H_site = Site(isotope="1H")

C_site = Site(isotope="13C",
              isotropic_chemical_shift=100.0,  # in ppm
              shielding_symmetric=SymmetricTensor(zeta=70.0, eta=0.5))

my_sites = [H_site, C_site]
```

mrsimulator Script Example - 1. Create SpinSystem - Coupling

```python
# Import the Coupling class
from mrsimulator import Coupling

# Create the Coupling object
coupling = Coupling(site_index=[0, 1], dipolar=SymmetricTensor(D=-2e4)) # D in Hz
```

```python
# Import the SpinSystem class
from mrsimulator import SpinSystem

# Create the SpinSystem object
spin_system = SpinSystem(sites=my_sites, couplings=[coupling])
```

mrsimulator Script Example - 2. Create Method

```python
# Import the BlochDecaySpectrum class
from mrsimulator.method.lib import BlochDecaySpectrum
from mrsimulator.method import SpectralDimension

# Create a BlochDecaySpectrum object
method = BlochDecaySpectrum(
    channels=["13C"],
    magnetic_flux_density=9.4,  # in T
    rotor_angle=54.735 * 3.14159 / 180,  # in rad (magic angle)
    rotor_frequency=3000,  # in Hz
    spectral_dimensions=[
        SpectralDimension(
            count=2048,
            spectral_width=80e3,  # in Hz
            reference_offset=6e3,  # in Hz
            label=r"$^{13}$C resonances",
        )
    ],
)
```

mrsimulator Script Example - 3. Create Simulator and Run

```python
# Import the Simulator class
from mrsimulator import Simulator

# Create a Simulator object
sim = Simulator(spin_systems=[spin_system], methods=[method])
sim.run()
```

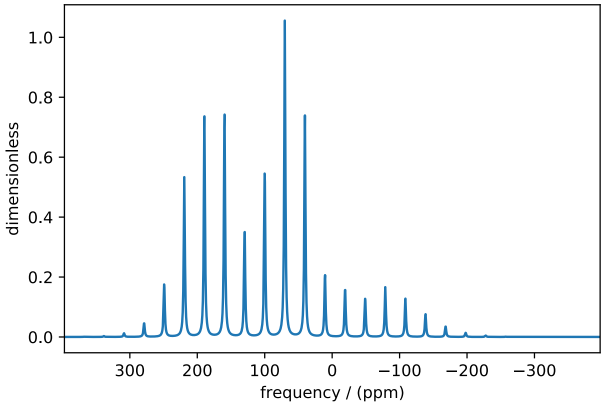
mrsimulator Script Example - Create SignalProcessor to add line broadening

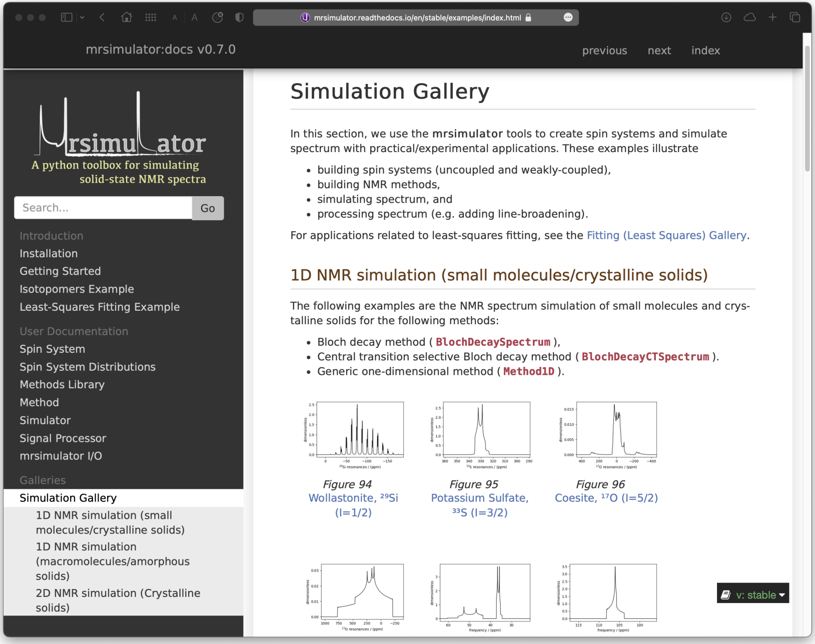
```python
from mrsimulator import signal_processor as sp

# Create the SignalProcessor object
processor = sp.SignalProcessor(
    operations=[
        sp.IFFT(),
        sp.apodization.Exponential(FWHM="200 Hz"),
        sp.FFT(),
    ]
)

# Apply the processor to the simulation dataset
processed_simulation = processor.apply_operations(dataset=sim.methods[0].simulation)
```

Matplotlib - Python library for creating static, animated, and interactive visualizations

https://matplotlib.org

```
import matplotlib.pyplot as plt
plt.figure()
ax = plt.subplot(projection="csdm")
ax.plot(processed_simulation.real)
ax.invert_xaxis()
plt.savefig("plot.pdf")
plt.show()
```





mrsimulator.readthedocs.io/en/stable/examples/index.html

mrsimulator:docs v0.7.0

previous next index

A python toolbox for simulating solid-state NMR spectra

Search... Go

Introduction
Installation
Getting Started
Isotopomers Example
Least-Squares Fitting Example

User Documentation
Spin System
Spin System Distributions
Methods Library
Method
Simulator
Signal Processor
mrsimulator I/O

Galleries
Simulation Gallery
1D NMR simulation (small molecules/crystalline solids)
1D NMR simulation (macromolecules/amorphous solids)
2D NMR simulation (Crystalline solids)

Simulation Gallery

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

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

For applications related to least-squares fitting, see the Fitting (Least Squares) Gallery.

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`),
- Central transition selective Bloch decay method (`BlochDecayCTSpectrum`).
- Generic one-dimensional method (`Method1D`).

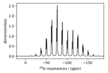


Figure 94 Wollastonite, ^{29}Si (I=1/2)

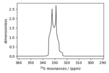


Figure 95 Potassium Sulfate, ^{33}S (I=3/2)

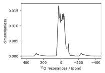
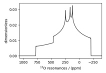
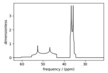
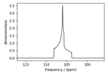


Figure 96 Coesite, ^{17}O (I=5/2)







v: stable

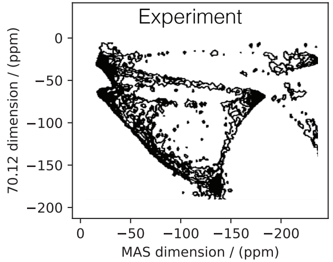
mrsimulator has pre-built specialized Methods for easier setup

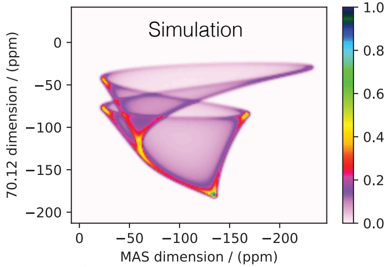
- **BlochDecaySpectrum** - Bloch Decay 1D spectrum
- **BlochDecayCTSpectrum** - Bloch Decay 1D spectrum of quadrupolar central transition
- **ThreeQ_VAS** - 3-quantum 2D spectrum of half-integer quadrupoles
- **FiveQ_VAS** - 5-quantum 2D spectrum of half-integer quadrupoles
- **SevenQ_VAS** - 7-quantum 2D spectrum of half-integer quadrupoles
- **ST1_VAS** - 1st inner satellite transition 2D spectrum of half-integer quadrupoles
- **ST2_VAS** - 2nd inner satellite transition 2D spectrum of half-integer quadrupoles
- **SSB2D** - Finite to infinite MAS speed correlation 2D spectrum, i.e., 2D PASS, 2D MAT.

Efficient 1D and 2D tenting algorithms for anisotropic spectra

^{87}Rb Switched-Angle Spinning of Rb_2CrO_4,
Shore et al., J. Chem. Phys., 105, 9412 (1996)

```
sas = Method(channels=["87Rb"], magnetic_flux_density=4.2,
    rotor_frequency=np.inf,
    spectral_dimensions=[
        SpectralDimension(count=256,
            spectral_width=1.5e4,  # in Hz
            reference_offset=-5e3,  # in Hz
            label="70.12 dimension",
            events=[
                SpectralEvent(rotor_angle=70.12 * 3.14159 / 180,
                    transition_queries=[
                        {"ch1": {"P": [-1], "D": [0]}}])]),
        SpectralDimension(count=512,
            spectral_width=15e3,  # in Hz
            reference_offset=-7e3,  # in Hz
            label="MAS dimension",
            events=[
                SpectralEvent(rotor_angle=54.74 * 3.14159 / 180,
                        transition_queries=[
                            {"ch1": {"P": [-1], "D": [0]}}])]),
    ],
)
```





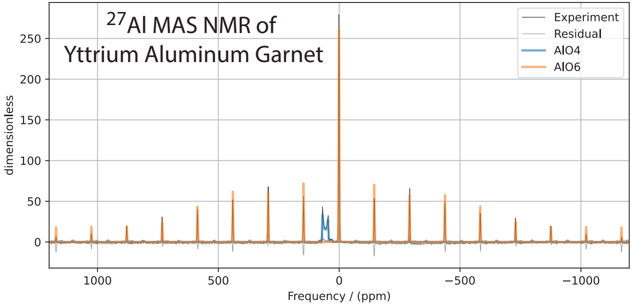
Simulation time: 40 milliseconds

Breakout Session: Simulating Spectra

- Simulating Protein GB1, ^{13}C and ^{15}N MAS spectra
- Simulating spectrum of amorphous sample - using single_site_system_generator

Solve least-squares problems with mrsimulator + LMFit

https://lmfit.github.io/lmfit-py



Three steps in least-squares calculations with mrsimulator + LMFit

First, Import the experimental spectrum.

```python
import csdmpy as csdm
from mrsimulator.utils import get_spectral_dimensions
from mrsimulator import Site, SpinSystem
from mrsimulator.spin_system.tensors import SymmetricTensor
from mrsimulator.methods import BlochDecaySpectrum
from mrsimulator import signal_processor as sp

host = "https://nmr.cemhti.cnrs-orleans.fr/Dmfit/Help/csdm/"
filename = "27Al Quad MAS YAG 400MHz.csdf"
experiment = csdm.load(host + filename)
sigma = 0.4895381 # standard deviation of noise from the dataset
experiment = experiment.real # For spectral fitting, use real part of the complex dataset
experiment.x[0].to("ppm", "nmr_frequency_ratio") # Convert coordinates from Hz to ppm.

# get the count, spectral_width, and reference_offset information from the experiment.
spectral_dims = get_spectral_dimensions(experiment)
```

Three steps in least-squares calculations with mrsimulator + LMFit

Second: Setup the Spin System, Method, and Simulation.

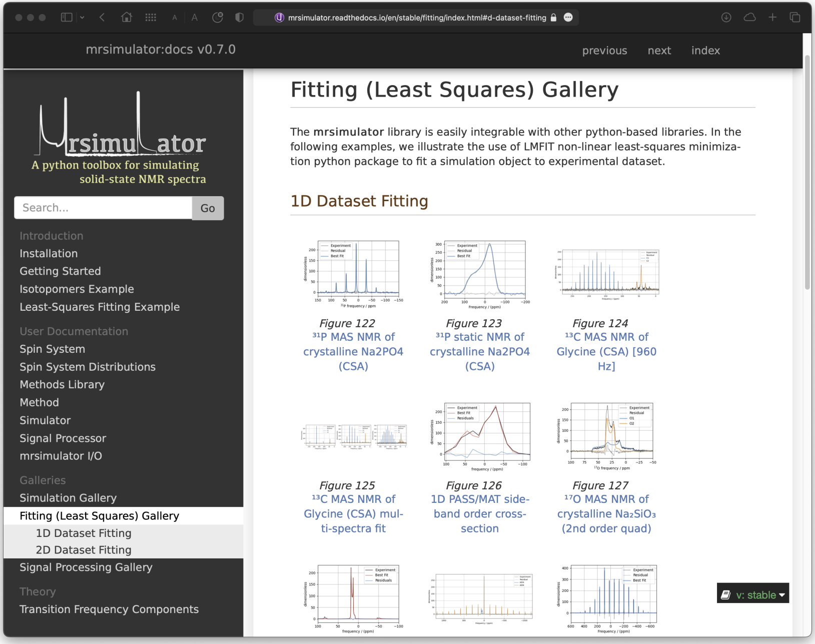
```
Al_1 = Site(isotope="27Al",isotropic_chemical_shift=76,
            quadrupolar=SymmetricTensor(Cq=6e6, eta=0.0))
Al_2 = Site(isotope="27Al",isotropic_chemical_shift=1,
            quadrupolar=SymmetricTensor(Cq=5e5, eta=0.3))
spin_systems = [SpinSystem(sites=[Al_1],name="AlO4"),SpinSystem(sites=[Al_2], name="AlO6")]

MAS = BlochDecaySpectrum(channels=["27Al"], magnetic_flux_density=9.395,
                         rotor_frequency=15250, spectral_dimensions=spectral_dims,
                         experiment=experiment)  # add the measurement to the method.

sim = Simulator(spin_systems=spin_systems, methods=[MAS])
sim.config.decompose_spectrum = "spin_system"
sim.run()

processor = sp.SignalProcessor(operations=[sp.IFFT(), sp.apodization.Gaussian(FWHM="300 Hz"),
                                           sp.FFT(), sp.Scale(factor=50)])
processed_dataset = processor.apply_operations(dataset=sim.methods[0].simulation).real
```

Three steps in least-squares calculations with mrsimulator + LMFit

Last: Create LMFit parameters from Simulator and SignalProcessor, and minimize!

```
from mrsimulator.utils import spectral_fitting as sf

params = sf.make_LMFIT_params(sim, processor, include={"rotor_frequency"})

from lmfit import Minimizer
minner = Minimizer(sf.LMFIT_min_function, params, fcn_args=(sim, processor, sigma))
result = minner.minimize()
```

With mrsimulator+LMFit, you can simultaneously fit spectra from different methods for a single set of spin system parameters.



mrsimulator.readthedocs.io/en/stable/fitting/index.html#d-dataset-fitting

mrsimulator:docs v0.7.0

previous next index

Mrsimulator
A python toolbox for simulating solid-state NMR spectra

Search... Go

Fitting (Least Squares) Gallery

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 dataset.

1D Dataset Fitting

Figure 122 ^{31}P MAS NMR of crystalline Na2PO4 (CSA)

Figure 123 ^{31}P static NMR of crystalline Na2PO4 (CSA)

Figure 124 ^{13}C MAS NMR of Glycine (CSA) [960 Hz]

Figure 125 ^{13}C MAS NMR of Glycine (CSA) multi-spectra fit

Figure 126 1D PASS/MAT sideband order cross-section

Figure 127 ^{17}O MAS NMR of crystalline Na_2SiO_3 (2nd order quad)

v: stable

Breakout Session: Fitting Spectra

- Fitting ^{13}C MAS sidebands of glycine.