Introduction to Python® Integration Using JASON and BeautifulJASON

Product used：Nuclear Magnetic Resonance (NMR)

Currently, JASON[1] provides an efficient environment for NMR spectral analysis using Python[2], leveraging the hierarchical structure of HDF5 files and the functionality of BeautifulJASON.

This document introduces the following two topics:

- Overview of BeautifulJASON and an example of NMR spectral analysis using Python
- Batch processing example using JASON × BeautifulJASON × Python

1. Overview of BeautifulJASON and Example of NMR Spectral Analysis Using Python

What is BeautifulJASON?

BeautifulJASON is a Python interface library designed to work with JASON and its file format (.jjh5).

It enables direct access and manipulation of spectral data created and stored in JASON from within Python.

This library is suitable for the following use cases:

- Automated spectral processing and analysis
- Custom layout and report generation
- Batch processing and laboratory automation
- Integration into data analysis pipelines

BeautifulJASON is particularly useful for users who regularly work with JASON and wish to handle spectral data programmatically.

By invoking JASON functions through Python scripts, users can perform batch processing, statistical analysis, and data visualization beyond the capabilities of GUI-based operations.

Example: Extracting and Analyzing NMR Data Using BeautifulJASON (Single File)

This section presents a simple example of NMR spectral analysis using Python and BeautifulJASON.

The analysis workflow is outlined in Table 1, with each step and the corresponding Python libraries listed.

The computed results are shown in Table 2.

This example serves as a practical introduction to BeautifulJASON and can be used as a reference for initial spectral analysis.

Table 1. Workflow Example Using BeautifulJASON (1 file)

Step	Process Description	Libraries Used	Representative Code Snippet
Data Loading	Load .jjh5 file	beautifuljason	doc = bjason.Document("file.jjh5", mode='r')
Information Extraction	Retrieve spectral data, peak positions, and integration values	beautifuljason, numpy	for m in spectrum.multiplets: print(m.pos[0], m.value_hz)
Statistical Calculation	Calculate average chemical shift and total integration	numpy	np.mean(peak_positions), np.sum(integral_values)
Statistical Calculation	Calculate average chemical shift and total integration	numpy	purity = (I / Istd) * (Nsstd / Ns) * ...
Visualization & Tabulation	Display peak-wise purity in chart and table format	pandas, matplotlib	plt.bar(peak_positions, peak_purities) pd.DataFrame(...).to_string()

Table 2. Analysis Example

Chemical Shift (ppm)	Purity (%)
7.6	99.4170
6.7	99.4340
4.0	99.3399
1.5	99.3364
1.2	99.8782
0.7	99.6659
Average Purity	**99.5119**

Reading NMR Data and Extracting Information with BeautifulJASON

The code shown in Figure 1 demonstrates how to extract spectral and peak information from .jjh5[3]-formatted NMR measurement data.

This example is useful for understanding the basic usage of BeautifulJASON.

The script retrieves the following information:

- Spectrum title and nucleus type
- Chemical shift positions and integration values of multiplets[4]
- Spectrum structure (e.g., nmr_data, nmr_items)

Such processing is essential for preprocessing and organizing spectral data.

By utilizing BeautifulJASON's powerful data extraction capabilities, Python-based analysis can be performed smoothly and efficiently.

JEOL Ltd. 3-1-2 Musashino Akishima Tokyo 196-8558 Japan Sales Division Tel. +81-3-6262-3560 Fax. +81-3-6262-3577
www.jeol.com ISO 9001・ISO 14001 Certified

```python
import beautifuljason as bjason

# Load NMR data file in JASON format (read mode)
with bjason.Document(input_1H_file, mode='r') as doc:

    # Display the title of each spectrum (nmr_data)
    # → Allows you to check labels or descriptions assigned to each spectrum
    for spec in doc.nmr_data:
        title = bjason.utils.ensure_str(spec.spec_info.get_param("Title"))
        # Retrieve title as a string (convert if it's a byte sequence)
        print(title)

    # For each NMR item (nmr_items), display header and spectrum info
    # → Useful for checking nuclide type and title when multiple spectra
    # are present
    for nmr_item in doc.nmr_items:
        print(nmr_item.header)  # Header information of the item
        for spec in nmr_item.spec_data_list:
            nuclide = spec.spec_info.nuclides[0]  # e.g., '1H' or '13C'
            title = bjason.utils.ensure_str(spec.spec_info.get_param("Title"))
            print("  ", nuclide, title)

    # Display multiplet information within each spectrum
    # → Shows chemical shift (ppm) and intensity (Hz) for each peak
    for spectrum in doc.nmr_data:
        for multiplet in spectrum.multiplets:
            # Get position (ppm) and intensity (Hz) of each multiplet (peak)
            print(f"ppm: {multiplet.pos[0]}, value: {multiplet.value_hz}")
```

Figure 1. NMR Data Extraction Code Example with BeautifulJASON

Official Documentation for BeautifulJASON

For detailed information and API references, please refer to the official documentation:

https://www.jeoljason.com/beautifuljason/docs/

Example: Statistical Analysis of Multiple NMR Datasets Using BeautifulJASON

This section presents an example of statistical analysis performed on multiple .jjh5-formatted NMR datasets.

Using BeautifulJASON, peak information was extracted from each file, followed by quantitative calculations.

The analysis included:

- Coefficient of Variation (CV) for variability assessment (Figure 2)
- Kernel Density Estimation (KDE) plots for visualizing data distribution (Figure 3)
- QQ plots for checking normality (Figure 4)
- Correlation analysis for identifying influential factors (Figure 5)

These methods enable quantitative evaluation of measurement variability, distribution characteristics, and relationships between variables.

This type of analysis is particularly useful for assessing measurement reproducibility and comparing samples.

The analysis steps and key considerations are summarized in Table 3.

Table 3. Analysis Steps and Considerations for NMR Purity Evaluation

Step	Process Description	Libraries / Modules Used	Representative Code Snippet	Key Insight
1. Quantitative Calculation	Calculate purity from peak integration values	beautifuljason, numpy	purity = (I / Istd) * ...	Derive quantitative purity from measured values
2. CV Evaluation	Calculate the coefficient of variation (CV) using mean and standard deviation of purity	numpy, matplotlib	cv = std / mean * 100	Assess variability and reproducibility of measurements
3. KDE Plot	Visualize the distribution of purity values using Kernel Density Estimation	seaborn, matplotlib	sns.kdeplot(purities, ...)	Understand distribution skewness and spread visually
4. QQ Plot	Check normality of purity distribution	scipy.stats, matplotlib	stats.probplot(purities, ...)	Validate statistical assumptions (normality)
5. Correlation Analysis	Evaluate relationships between purity and other variables	scipy.stats	pearsonr(x, y) spearmanr(x, y)	Explore influence and trends among variables

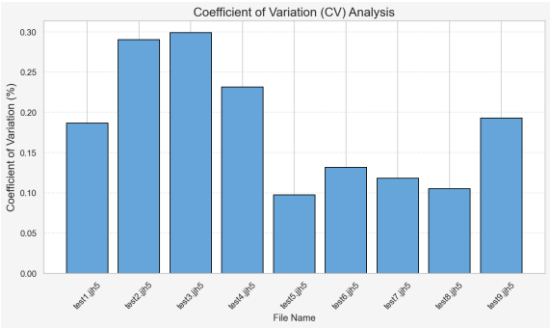
Variability Assessment Using Coefficient of Variation (CV)

For multiple NMR data files, the Coefficient of Variation (CV) was calculated based on the mean and standard deviation of peak purity values.

CV is a metric that expresses the ratio of the standard deviation to the mean, and is commonly used to evaluate the variability and reproducibility of measurements.

As shown in Figure 2, files such as test3 and test2 exhibited relatively high CV values, indicating greater variability.

In contrast, test5 and test8 showed lower CV values, suggesting more consistent measurements.




```python
import beautifuljason as bjason

# Load the file
with bjason.Document(input_1H_file, mode='r') as doc:
    # Retrieve peak information from the multiplets property
    peak_positions = [multiplet.pos[0] for spectrum in doc.nmr_data
                      for multiplet in spectrum.multiplets]
    integral_values = [multiplet.value_hz for spectrum
                       in doc.nmr_data for multiplet in spectrum.multiplets]
```

```python
import numpy as np

# Calculate average purity and coefficient of variation
average_purity = np.mean(peak_purities)
std_purity = np.std(peak_purities)
cv_purity = std_purity / average_purity * 100  # Coefficient of variation
```

Figure 2. Coefficient of Variation-Based Comparison and Code Snippet for CV Calculation

Visualizing Data Distribution Using KDE Plots

Figure 3 shows the distribution of peak purity values for each NMR file, visualized using Kernel Density Estimation (KDE).

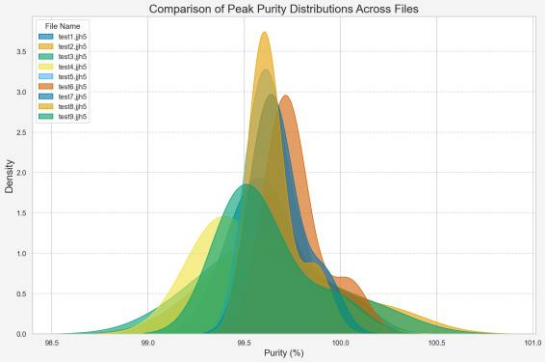
KDE plots provide a smooth curve representation of data distribution, offering more detailed insights than histograms.

They are particularly useful for visually assessing variability and skewness in purity measurements.

For example:

A wider distribution suggests greater variability in purity values. A narrower distribution indicates more stable and consistent measurements.

Overlaying KDE plots for multiple files allows for direct comparison of measurement conditions and differences between samples.




```python
import seaborn as sns

# Create KDE plots
for idx, (file_name, purities) in enumerate(file_peak_purities.items()):
    sns.kdeplot(
        purities,
        fill=True,
        label=file_name,
        alpha=0.6,
        color=color_palette[idx % len(color_palette)]
        # Select corresponding color
    )
```

Figure 3. Peak Purity Distribution Comparison via KDE Plot and Code Snippet for Plot Generation

Checking Normality Using QQ Plots

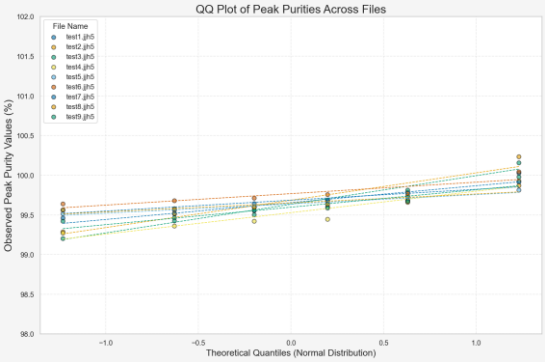
Figure 4 shows the results of evaluating the agreement between peak purity values for each NMR file and a theoretical normal distribution using Quantile-Quantile (QQ) plots.

A QQ plot is a graphical method for assessing whether a dataset follows a theoretical normal distribution by comparing the quantiles of the observed data with those of the theoretical distribution.

The closer the points lie to a straight line, the more closely the data approximates normality.

When outliers are present, the ends of the plot tend to deviate significantly from the line.

In this analysis, all datasets exhibited strong linearity, confirming that the assumption of normality is valid for subsequent statistical analyses.




```python
import scipy.stats as stats

# Generate QQ plots for each file
for file_name, purities in file_peak_purities.items():
    plt.figure(figsize=(10, 6))
    stats.probplot(purities, dist="norm", plot=plt)
    # Compare with normal distribution
    plt.title(f"QQ Plot of Peak Purities for {file_name}")
    # Add file name to the title
    plt.xlabel("Theoretical Quantiles (Normal Distribution)")
    # Description of theoretical distribution
    plt.ylabel("Observed Peak Purity Values (%)")  # Measured values
    plt.grid(linestyle="--", alpha=0.7)
    plt.tight_layout()
    plt.show()
```

Figure 4. Visualization of Normality Using QQ Plot and Excerpt from QQ Plot Generation Code

Evaluating Relationships Through Correlation Analysis

Figure 5 presents the results of assessing the relationships between multiple variables (chemical shift, integration values, CV) and peak purity using Pearson and Spearman correlation coefficients.

The correlation coefficients (r for Pearson, ρ for Spearman) provide a comprehensive evaluation of both linear and non-linear relationships.

In this case, all correlation coefficients were small in absolute value, and p-values were high, indicating no statistically significant correlations.

This suggests that the quantitative results are stable with respect to these variables and are minimally influenced by measurement conditions.

Although a negative correlation with CV was observed, the correlation coefficient was small and not statistically significant, indicating that further investigation would be required to establish any causal relationship.

Comparison Item	Correlation Coefficient (r / ρ)	p-Value	Trend
Chemical Shift vs. Purity	0.1051 / 0.0950	0.4870 / 0.5298	Weak positive correlation (not significant)
Integration Value vs. Purity	0.0493 / -0.1095	0.7451 / 0.4690	Very weak correlation (not significant)
CV vs. Purity	-0.1272 / -0.1139	0.3996 / 0.4511	Weak negative correlation (not significant)

```python
from scipy.stats import pearsonr, spearmanr

pearson_r, p_val = pearsonr(x, y)
spearman_rho, p_val_s = spearmanr(x, y)
```

Figure 5. Relationship Evaluation via Correlation Analysis and Excerpt from Correlation Analysis Code

JEOL Ltd.
3-1-2 Musashino Akishima Tokyo 196-8558 Japan Sales Division Tel. +81-3-6262-3560 Fax. +81-3-6262-3577
www.jeol.com ISO 9001 • ISO 14001 Certified

2. Batch Processing Example Using JASON × BeautifulJASON × Python

This section introduces an example of automated NMR data analysis using Python scripts.

By utilizing the script files provided in “Batch Processing with BeautifulJASON”, available from the official JASON website, multiple NMR datasets can be processed automatically by applying JASON’s Rule functionality[5], followed by exporting the results as CSV files.

Table 4. Three-Step Workflow for NMR Batch Processing Using JASON

Step	Description	Representative Command
1. Input Preparation	Place multiple NMR data files (.jdf) into the input folder.	—
2. Convert to .jjh5 Format	Use jason_batch_convert.py to convert .jdf files into .jjh5 format and apply JASON rules.	jason_batch_convert.py <input> <output> --formats jjh5 --extensions jdf --rules "rule_file" --execute
3. Export to CSV	Use batch_extract_integrals.py to extract integration values and parameters from .jjh5 files and save them as a CSV file.	batch_extract_integrals.py <output> <csv_path> -p parameters/ACTUAL_START_TIME jason_parameters/SpectrometerFrequencies[0]

Batch Processing with BeautifulJASON: Provided Scripts

Two script files are available:

1. `jason_batch_convert.py`: Performs batch conversion of data using JASON’s Rule functionality
2. `batch_extract_integrals.py`: Extracts integration values from converted data and outputs them in CSV format

Both scripts are included in the BeautifulJASON installation package.

To ensure proper access and usage, users are encouraged to install BeautifulJASON.

Processing Steps

Table 4 summarizes the three main steps for NMR batch processing using JASON**.**

Conversion Script (`jason_batch_convert.py`)

Figure 6 shows a portion of the `jason_batch_convert.py` code.

This script applies JASON’s Rule functionality to convert data into formats such as .jjh5.

In particular, the argument-handling section within the main() function clearly describes the meaning of each command-line option, which helps users understand how to execute the script.

```python
def process_file(execute: bool, file: str, in_dir: str,
                 out_dir: str, jason: bjason.JASON, formats: list[str],
                 rules: str, counter: ValueProxy, total_files: int):
    """Process a single file, converting it to the specified output formats."""
    # Generate output filenames for each requested format
    out_fnames = [unique_output_filename(file, format) for format in formats]

    if execute:
        # If execution is enabled, perform the actual file conversion
        global processed_files
        with jason.create_document(os.path.join(in_dir, file),
                                   rules=rules) as doc:
            doc.close()  # Close the document after processing
            # Save the converted document to the output directory
            # in all specified formats
            jason.save(doc, [os.path.join(out_dir, fname)
                             for fname in out_fnames])

    # Increment the processed file counter
    counter.value += 1
    # Display progress in the console
    print(f"{counter.value}/{total_files}: {file} => {', '.join(out_fnames)}")
```

```python
def main():
    init(autoreset=True)
    # Initialize colorama to reset console colors automatically

    # Set up command-line argument parser
    parser = argparse.ArgumentParser(
        description='Convert files from a specified directory based on extensi
        usage='jason_batch_convert [-h] in_dir out_dir --formats FORMATS [FORMA
    )

    # Input directory containing files to be converted
    parser.add_argument
    ('in_dir', help='Root directory containing files to be converted.')
    # Output directory where converted files will be saved
    parser.add_argument
    ('out_dir', help='Directory where the converted files will be saved.')

    # List of desired output formats (multiple formats allowed)
    parser.add_argument('--formats', nargs='+', required=True,
                        choices=['jjh5', 'jjj', 'jdx', 'jdf', 'pdf', 'png', 'jp
                        help='List of desired output file formats. Multiple for

    # Optional rules file or rule library name
    parser.add_argument('--rules', type=str, default="off",
                        help='Path to the rules file or rule library name.')

    # Execution flag: if not set, performs a dry run listing files
    # to be converted
    parser.add_argument('--execute', action='store_true',
                        help='Execute the file conversions. If not specified, p
```

Figure 6. Code snippet demonstrating .jdf file conversion and the use of JASON rule-based processing

Output Script (`batch_extract_integrals.py`)

Figure 7 shows part of the `batch_extract_integrals.py` code.

This script extracts integration values and parameters from .jjh5 files and writes them to a CSV file.

```python
def escape_newlines_for_csv(value):
    """Escapes newline characters in a string for CSV compatibility."""
    # Ensure the value is a string (handles various input types)
    value = str(bjason.utils.ensure_str(value))
    # Replace newline characters with literal \n and \r
    # to prevent breaking CSV formatting
    return value.replace('\n', '\\n').replace('\r', '\\r')

def extract_integrals(jjh5_file_path: str,
                      csv_writer: csv.writer, params: list[str]):
    """
    Extracts integral values and specified parameters from a JJH5 file,
    and writes them to a CSV file using the provided csv.writer.
    """
    try:
        # Get the base filename without extension for labeling the output row
        base_name = os.path.basename(jjh5_file_path)
        file_name_without_extension, _ = os.path.splitext(base_name)

        # Open the JJH5 file using the bjason.Document context manager
        with bjason.Document(jjh5_file_path, mode="r") as doc:
            spec = doc.nmr_data[0]  # Access the first NMR dataset
            row = [file_name_without_extension]
            # Initialize the CSV row with the filename

            # Loop through each parameter string provided
            for param in params:
             # Split the parameter into group and name (e.g., "group/name[index]")
                param_parts = param.split('/')
                param_group = param_parts[0]
                param_name = param_parts[1]
                param_index = None
```

```python
                # Check if the parameter includes an index (e.g., name[0])
                if '[' in param_name:
                    param_name, param_index = param_name.split('[')
                    param_index = int(param_index[:-1])
                    # Remove closing bracket

                # Retrieve the parameter value based on its group
                if param_group == "jason_parameters":
                    param_value = spec.spec_info.get_param(param_name)
                else:
                    param_value = spec.raw_data.spec_info.get_orig_param
                    (param_group, param_name)

            # If the parameter is indexed and supports indexing,
            # extract the specific value
                if param_index is not None and param_value is not None:
                    if hasattr(param_value, '__getitem__'):
                        param_value = param_value[param_index]

                # Escape newlines and append the parameter value to the row
                row.append(escape_newlines_for_csv(param_value))

            # Append integral values (in Hz) from the multiplets to the row
            row.extend(integral.value_hz for integral in spec.multiplets)

            # Write the completed row to the CSV file
            csv_writer.writerow(row)

    except Exception as e:
        # Print an error message if the file could not be processed
        print(f"Error processing file '{jjh5_file_path}': {e}")
```

Figure 7. Code snippet demonstrating extraction of integrals and writing to CSV

Output Results

Figure 8 shows an example of the output.

In this example, nine NMR datasets were processed using JASON's Rule functionality, and the results were compiled into a CSV file.

The contents of the output data and the CSV file are shown in Figure 9.

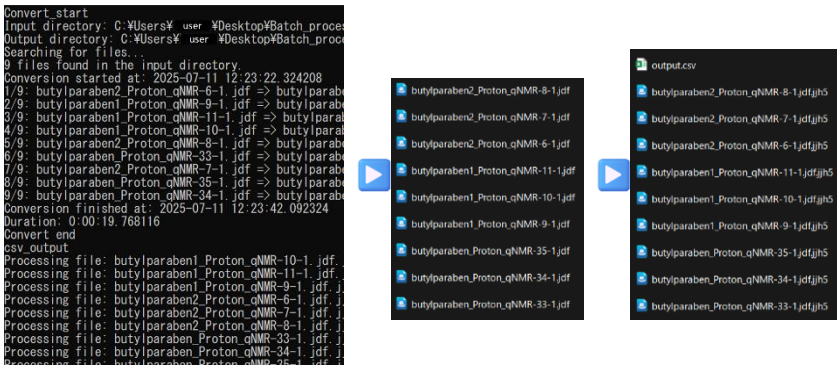
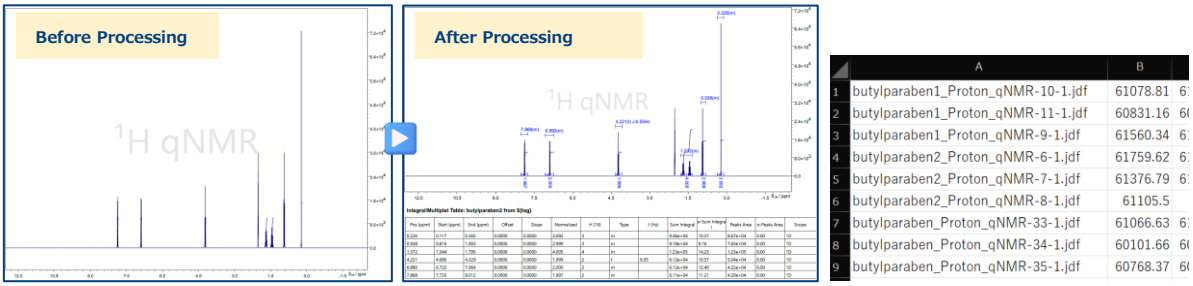


Figure 8. Example of automated analysis using BeautifulJASON batch processing



	A	B
1	butylparaben1_Proton_qNMR-10-1.jdf	61078.81
2	butylparaben1_Proton_qNMR-11-1.jdf	60831.16
3	butylparaben1_Proton_qNMR-9-1.jdf	61560.34
4	butylparaben2_Proton_qNMR-6-1.jdf	61759.62
5	butylparaben2_Proton_qNMR-7-1.jdf	61376.79
6	butylparaben2_Proton_qNMR-8-1.jdf	61105.5
7	butylparaben_Proton_qNMR-33-1.jdf	61066.63
8	butylparaben_Proton_qNMR-34-1.jdf	60101.66
9	butylparaben_Proton_qNMR-35-1.jdf	60768.37

Figure 9. Output results from automated batch processing with BeautifulJASON

Notes and Precautions

- Execute the scripts in a Python 3 environment.
- JASON Rule files (.jjr) must be created in JASON beforehand.
- Use the exact parameter names as displayed in the JASON GUI.

Summary

This document introduced methods for automating NMR spectral analysis using JASON and BeautifulJASON.

By leveraging BeautifulJASON, users can directly access and analyze JASON data from Python, enabling flexible processing and statistical analysis.

In particular, using the official scripts (`jason_batch_convert.py` and `batch_extract_integrals.py`) makes it easy to perform batch processing of multiple datasets and export results to CSV format.

References

JASON Official Website: https://www.jeoljason.com/
BeautifulJASON Documentation: https://www.jeoljason.com/beautifuljason/docs/
Batch Processing Script Files: https://www.jeoljason.com/resources-external-nmr-processing-scripts/
Python Official Documentation: https://docs.python.org/3/

[1] JEOL Analytical Software Network
[2] Python is a trademark or registered trademark of the Python Software Foundation.
[3] .jjh5 is the HDF5-based file format used in JASON.
[4] Multiplet refers to a group of peaks in an NMR spectrum resulting from spin-spin coupling.
[5] JASON's Rule functionality uses configuration files (.jjr) that define spectral processing steps and report layouts, enabling batch processing of multiple datasets.

JEOL Ltd.
3-1-2 Musashino Akishima Tokyo 196-8558 Japan Sales Division Tel. +81-3-6262-3560 Fax. +81-3-6262-3577
www.jeol.com ISO 9001・ISO 14001 Certified