NMR Inversion Recovery Analysis in Python

Author(s): Seth D. Veenbaas, Jessica A. Nash, The Molecular Sciences Software Institute

Overview

Objective:

- Use Pandas and Scipy to proceed NMR data.
- Use Matplotlib to visualize data.
- Calculate the T_1 relaxation times.
- Calculate an ideal d_1 delay time.

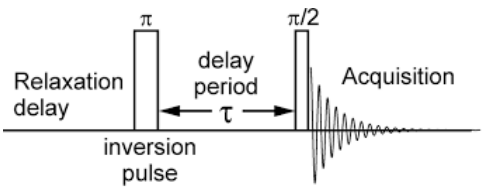
Inversion recovery experiment

The inversion-recovery experiment measures T_1 relaxation times of any nucleus. If the net magnetization is placed along the -z axis, it will gradually return to its equilibrium position along the +z axis at a rate governed by T_1. The equation governing this behavior as a function of the time t after its displacement is:

$$M_z(t) = M_{z,eq} \cdot (1 - 2e^{-t/T_1})$$

The basic pulse sequence consists of an 180° pulse that inverts the magnetization to the -z axis. During the following delay, relaxation along the longitudinal plane takes place. Magnetization comes back to the original equilibrium z-magnetization. A 90° pulse creates transverse magnetization. The experiment is repeated for a series of delay values taken from a variable delay list. A 1D spectrum is obtained for each value of vd and stored in a pseudo 2D dataset. The longer the relaxation delay (d_1) is, the more precise the T_1 measurement is. An ideal relaxation time (d_1) can be calculated (aq = acquisition time):

$$d_1 + aq = 5 \cdot T_1$$



More information: https://imserc.northwestern.edu/downloads/nmr-t1.pdf

Importing Required Libraries

First, let's import the python libraries/packages we need to work with the data.

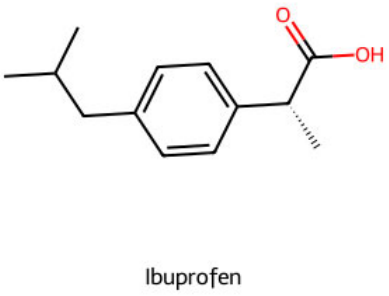
In [16]:
```
import pandas as pd
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import mnova
import rdkit
from rdkit.Chem import Draw

# Enable inline plotting
%matplotlib inline

# Set DataFrame precision to 3 decimal places
pd.set_option("display.precision", 3)
```

Inversion recovery of ibuprofen

We will be analyzing ^{1}H inversion recovery from ibuprofen today.

In [17]:
```
ibuprofen = rdkit.Chem.MolFromSmiles('CC(C)Cc1ccc(cc1)[C@@H](C)C(=O)O')
Draw.MolToImage(ibuprofen, legend='Ibuprofen')
```

Out[17]:



Ibuprofen

Importing the Data

We will now import the NMR inversion recovery data from a CSV file to a Pandas Dataframe (the excel of Python).

The file `Ibuprofen-C13-invrec-data-mnova.csv` contains the experimental data from an inversion recovery experiment for ibuprofen.

In [18]:
```
# Load the data from the CSV file
ibuprofen_inversion_data = pd.read_csv('data/Ibuprofen_CDCl3_1H_inversion_recovery.csv', header=1)

# Display the first 5 row of the dataframe
ibuprofen_inversion_data.head()
```

Out[18]:

	#	X(I)	Y(X)	Y'(X)	Y1(X)	Y1'(X)	
0	Model	ARR_DATA(I)	Integral(7.26560368,7.23132347)	B+F*exp(-x*G)\nB=194347\nF=-362851\nG=0.358889	Integral(7.14617265,7.11376701)	B+F*exp(-x*G)\nB=192994\nF=-361057\nG=0.381787	Integral(3.7617425
1	1	0.05000000	-162885.32421875	-162051.06574915	-162306.76904297	-161236.54185458	-80
2	2	0.11000000	-154924.75878906	-154458.65720112	-153773.73730469	-153214.34000972	-76
3	3	0.23000000	-139766.52441406	-139755.62179638	-137715.46069336	-137710.85487207	-67
4	4	0.46000000	-112673.04699707	-113284.81542128	-109142.05480957	-109909.81439179	-53

Use the `mnova.rename_columns()` function to reformat the Dataframe.

The format will make the data easier to work will.

Time(s)	<#>_ppm
Time of scan (seconds)	Chemical shift of each peak.

In [19]:
```
# Runs reformating function
ibuprofen_inversion_data = mnova.rename_columns(ibuprofen_inversion_data)

# Display the first 5 row of the dataframe
ibuprofen_inversion_data.head()
```

Out[19]:

	Time(s)	7.2_ppm	7.1_ppm	3.7_ppm	2.5_ppm	1.9_ppm	1.5_ppm	0.9_ppm
1	0.05	-162885.324	-162306.769	-80722.982	-153871.150	-79104.215	-216684.112	-458164.315
2	0.11	-154924.759	-153773.737	-76314.501	-138198.977	-74199.250	-180093.169	-408792.754
3	0.23	-139766.524	-137715.461	-67926.022	-109190.864	-64802.324	-114482.993	-315141.228
4	0.46	-112673.047	-109142.055	-53136.028	-60301.979	-48382.227	-12551.294	-160680.296
5	0.92	-65336.168	-59692.360	-27959.350	15407.574	-21192.123	122369.146	74055.751

Challange

Use `plot()` to visualize the inversion recovery signals in `ibuprofen_inversion_data`.

Tip: plot `Time(s)` on the x-axis.

`DataFrame.plot()`

Make plots of a DataFrame.

Parameters

- **x** : `str` of column name, *(default: `None`)*

  The column to use for the x-axis (independent variable).

- **y** : `str` or `list` of column name(s), *(default: `None`)*

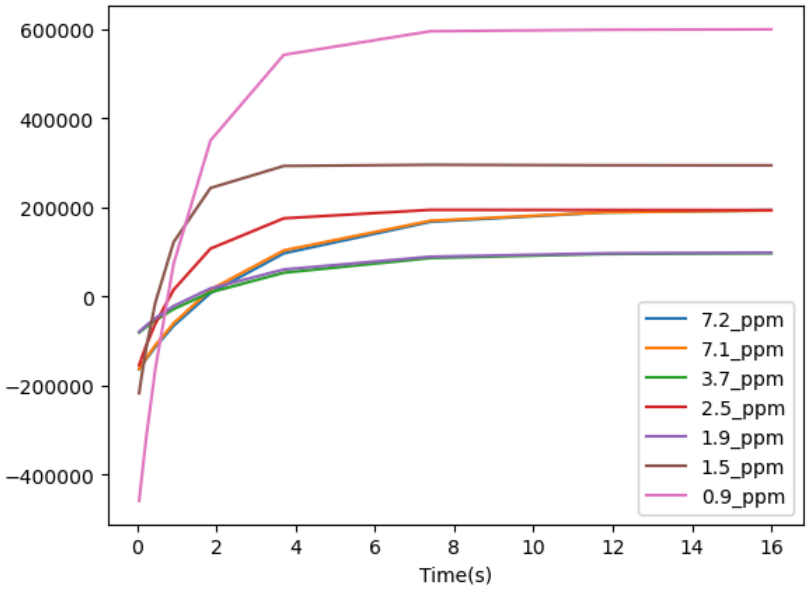
  The column(s) to use for the y-axis (dependent variable).

- **kind** : `str`, *(Default: `'line'`)*

  The kind of plot to produce (e.g., `'bar'`, `'hist'`, `'scatter'`)

In [20]:
```
# Plot the data from all peaks vs. Time(s)
ibuprofen_inversion_data.plot(x='Time(s)')
```

Out[20]:
```
<Axes: xlabel='Time(s)'>
```



Calculate T_1 relaxation time

We will use the `curve_fit()` function from scipy to fit our data to the exponential inversion recovery model:

$$M_z(t) = M_{z,\text{eq}} \cdot \left(1 - 2e^{-t/T_1}\right) + C$$

The `curve_fit()` function with optimize the three parameters in our inversion recovery model using our experimental data.

$$parameters = M_{z,\text{eq}}, T_1, C$$

Let's use `curve_fit()` with our data from the peak at 0.9 ppm to determine an optimal value for T_1!

In [21]:
```
# Extract data for the peak at 0.9 ppm
time_data = ibuprofen_inversion_data['Time(s)']
peak_data = ibuprofen_inversion_data['0.9_ppm']

# Define the inversion recovery model with the parameters (M, T1, and C)
def inversion_recovery_model(time, M, T1, C):
    return M * (1 - 2 * np.exp(-time / T1)) + C

# Initial guess for our three parameters (M, T1, and C)
initial_guess = [max(peak_data), 1, min(peak_data )]

# Fit the curve to get optimized parameters (M, T1, and C)
param_optimal, _ = curve_fit(inversion_recovery_model, time_data, peak_data, p0=initial_guess)

# Print optimized parameters
print(param_optimal)
```

```
[5.50279574e+05 1.24296583e+00 4.87960127e+04]
```

The second value above is the T_1 time (s) for the peak at 0.9 ppm!

Code reusability: calculate T_1 for all peaks

Great, We calculated T_1! Now just 6 more peaks to go....

Don't worry, we can reuse our code to do repetitive tasks if we design it properly.

Two really powerful tools for reusing code are:

- defining custom functions
- `for` loops

How to Define a Function

1. **Keyword**: Use the `def` keyword to start the definition.
2. **Name**: Choose a descriptive name for the function (e.g., `calculate_energy`).
3. **Parameters**: Enclose optional input parameters in parentheses `()`.
4. **Colon**: Add a colon `:` to indicate the start of the function body.
5. **Indented Body**: Write the function's logic as an indented block (4 spaces).
6. **Optional Docstring**: Explains what a function does, what the parameters are, and what the function returns (if any).
7. **Optional Return**: Use `return` to send a result back to the caller (if needed).

Here is what it looks like to define a function:

```
def function_name(parameters):
    # Optional: explain what your function does in a Docstring
    """
    Docstring
    """
    # Function body (indented code)
    return output  # Optional: Return a result
```

Let's create a function that we can reuse to plot our fitted data!

In [22]:
```
def plot_fitted_data(df, time_col, peak_col, param_optimal):
    """
    Plots the peak intensity vs. time data along with the fitted inversion recovery model.

    Parameters:
    - df (pd.DataFrame): The input DataFrame containing the data to be plotted.
    - time_col (str): The column name in the DataFrame representing time data.
    - peak_col (str): The column name in the DataFrame representing peak intensity data.
    - param_optimal (np.ndarray): Optimal parameters from the curve fitting (M_z,eq, T1, C) returned by `curve_fit`.
```

```
    Returns:
    - None: This function directly displays the plot using `matplotlib.pyplot.show()`.

    The function creates a scatter plot of the peak intensity vs. time and overlays a curve fit based on the
    inversion recovery model. The fitted T1 value is displayed in the plot legend.
    """
    time_data = df[time_col]
    peak_data = df[peak_col]

    # Create a blank figure
    plt.figure(figsize=(8, 6))

    # Plot peak intensity vs. time as a scatter plot
    plt.scatter(time_data, peak_data, label=f'{peak_col} Data')

    # Plot curve fit
    x_model = np.linspace(min(time_data), max(time_data), 100)
    y_model = inversion_recovery_model(x_model, *param_optimal)
    plt.plot(x_model, y_model, label=f'Fit: T1 = {param_optimal[1]:.3f} s', color='red')

    # Add labels, title, and legend
    plt.xlabel('Time (s)', fontsize=12)
    plt.ylabel('Signal intensity', fontsize=12)
    plt.title(f'Inversion Recovery Fit for peak {peak_col}',  fontsize=14)
    plt.legend(loc='lower right')

    # Show the plot
    plt.show()
```

Challange

Create a custom function that performs a curve fit to our inversion recovery model.

You do not need to write a doc string but do incorporate the following:

1. Function Name: `fit_relaxation_data`

2. Parameters:

- `time_data` (list): Time points for the relaxation curve.
- `peak_data` (list): Corresponding peak intensities.

3. Function body (indented code):

```
# Initial guess for our three parameters (M, T1, and C)
initial_guess = [max(peak_data), 1, min(peak_data )]

# Fit the curve to get optimized parameters (M, T1, and C)
param_optimal, _ = curve_fit(inversion_recovery_model, time_data, peak_data, p0=initial_guess)
```

4. Return:

- `param_optimal` Optimal parameters for the inversion recovery model (M_z,eq, T1, C)

In [23]:
```
# Write the custom function `fit_relaxation_data`
def fit_relaxation_data(time_data, peak_data):
    # Initial guess for M, T1, and C
    initial_guess = [max(peak_data), 1.0, min(peak_data)]

    # Fit the curve
    param_optimal, _ = curve_fit(inversion_recovery_model, time_data, peak_data, p0=initial_guess)

    return param_optimal
```

How to Write a for Loop

1. **Keyword**: Start with `for`.
2. **Iterator Variable**: Specify a variable for each item.
3. `in` **Keyword**: Use in to specify the sequence.
4. **Colon**: Add a colon `:` to start the loop body.
5. **Indented Body**: Indent the code to execute in each iteration.

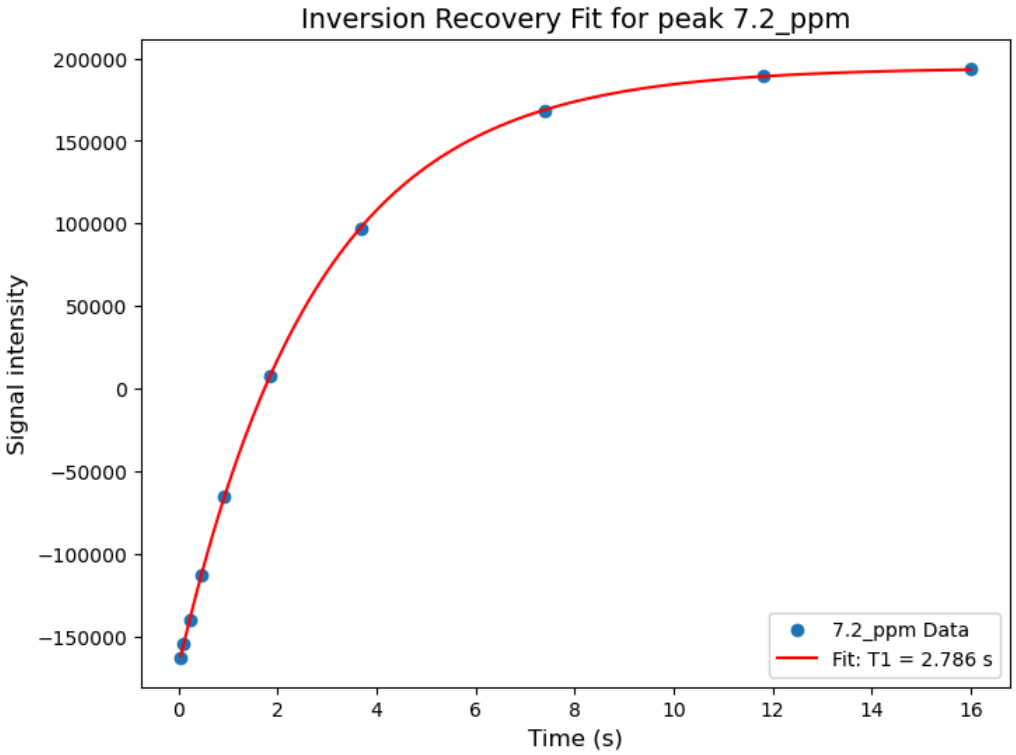
Here is what it looks like to write a `for` loop:

```
for each_item in your_items:
    # Code to be executed for each_item (indented code)
```

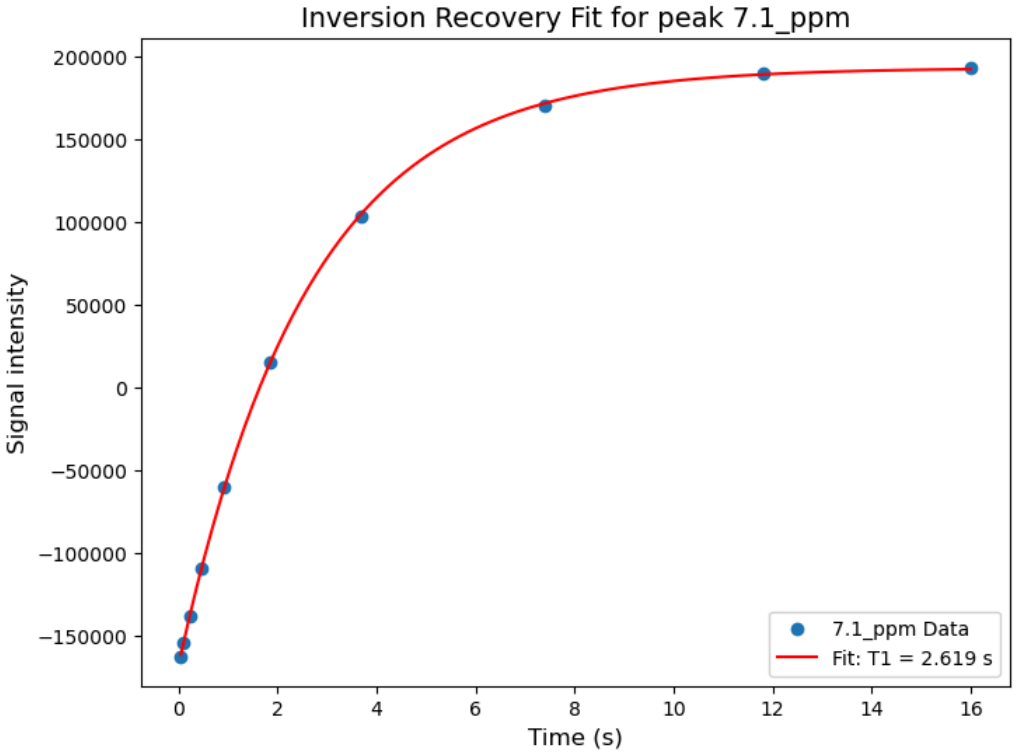
Let's create a `for` loop that:

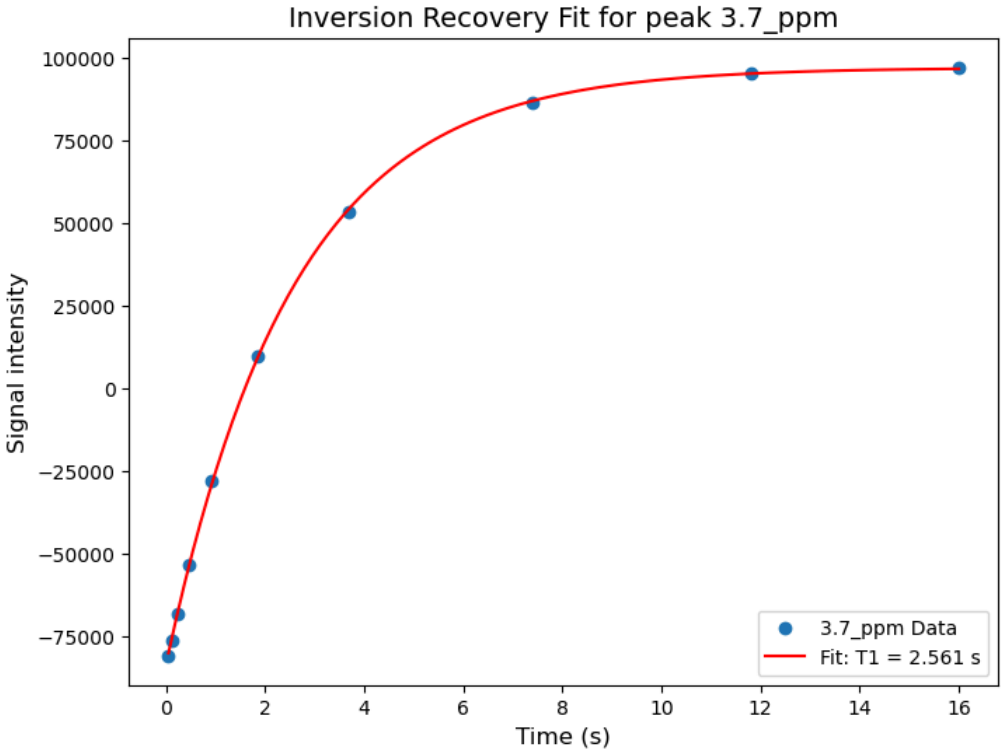
- Uses `fit_relaxation_data` to fit the data for every peaks.
- Uses `plot_fitted_data` to plot the fit for every peaks.
- Creates a DataFrame called `t1_data` to save the T_1 time for all our peaks.

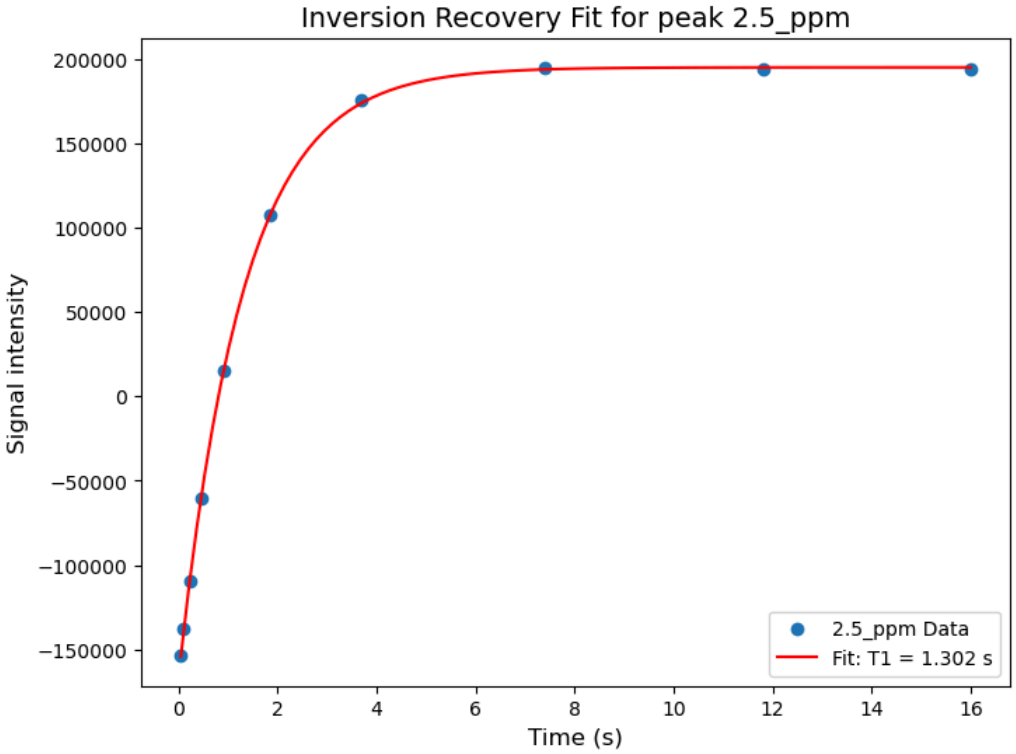
In [24]:
```
# Define what DataFrame to use to improve reusability.
df = ibuprofen_inversion_data

# Create an empty DataFrame for t1_data with columns 'Peak' and 'T1(s)'
t1_data = pd.DataFrame(columns=['Peak', 'T1(s)'])

# Calculate T1 for each peak and plot the fit
for column in df.columns[1:]:

    # Fit relaxation data to calculate T1
    time_data = df['Time(s)']
    peak_data = df[column]
    param_optimal = fit_relaxation_data(time_data, peak_data)
    T1 = param_optimal[1]

    # Plot the fitted data
    plot_fitted_data(df, 'Time(s)', column, param_optimal)

    # Add T1 time to the t1_data DataFrame
    new_row = pd.DataFrame({'Peak': [column], 'T1(s)': [T1]})
    t1_data = pd.concat([t1_data, new_row], ignore_index=True)

# Display `t1_data` DataFrame
t1_data
```

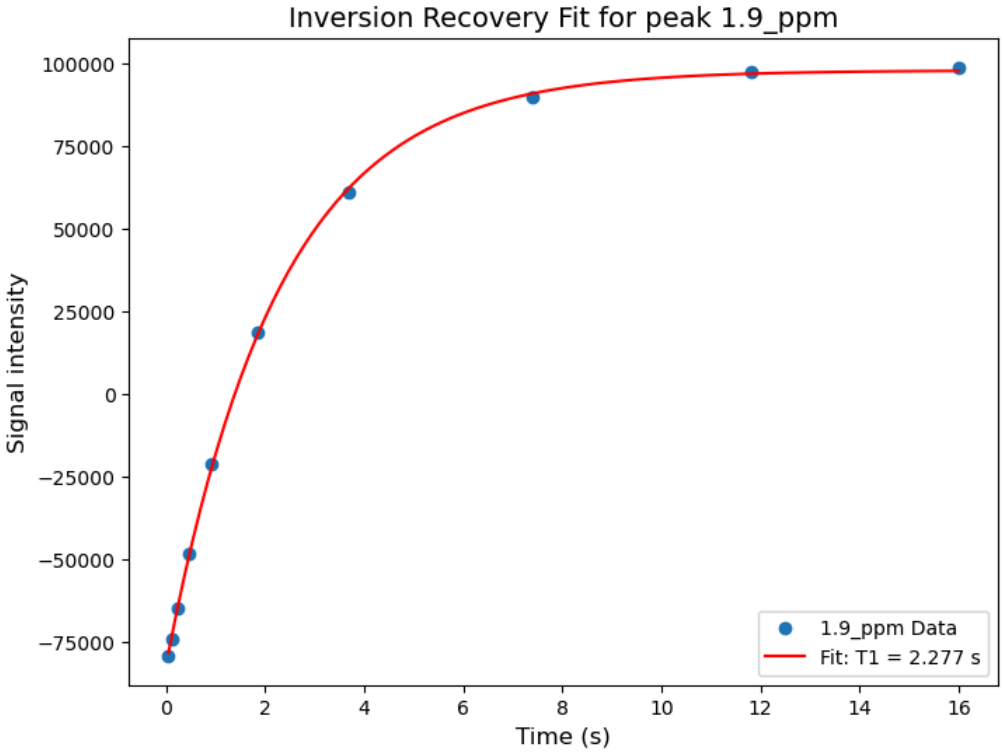


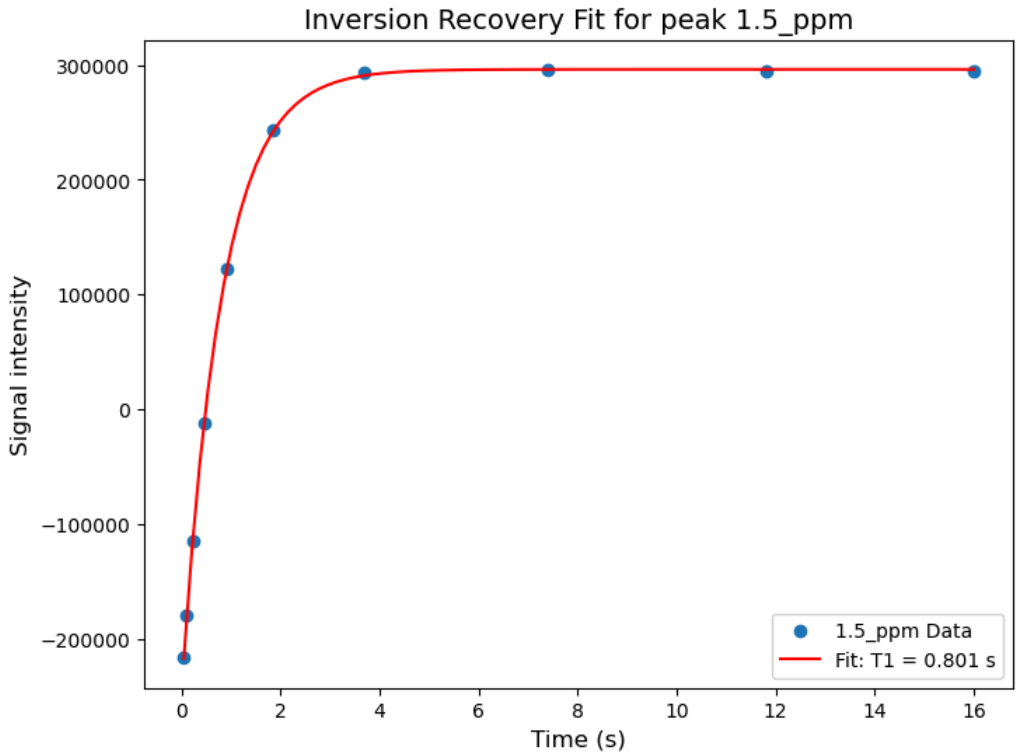

```
/tmp/ipykernel_53148/2119918285.py:21: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is d
eprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To reta
in the old behavior, exclude the relevant entries before the concat operation.
  t1_data = pd.concat([t1_data, new_row], ignore_index=True)
```

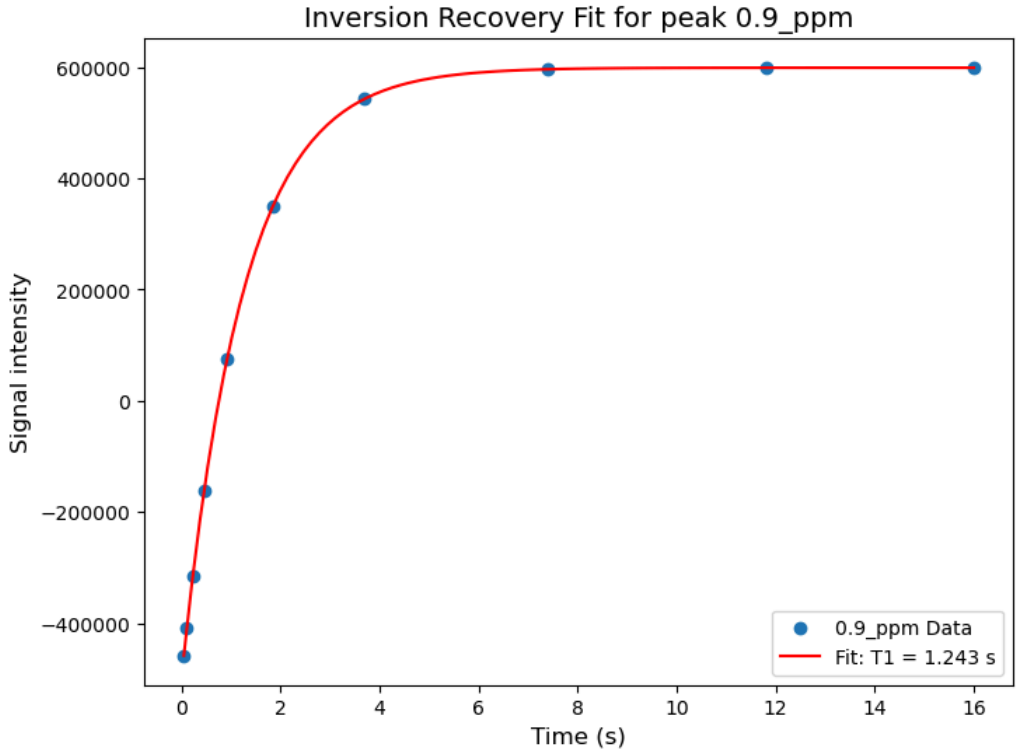

Inversion Recovery Fit for peak 7.1_ppm
Signal intensity
Time (s)
7.1_ppm Data
Fit: T1 = 2.619 s


Inversion Recovery Fit for peak 3.7_ppm
Signal intensity
Time (s)
3.7_ppm Data
Fit: T1 = 2.561 s


Inversion Recovery Fit for peak 2.5_ppm
Signal intensity
Time (s)
2.5_ppm Data
Fit: T1 = 1.302 s


Inversion Recovery Fit for peak 1.9_ppm
Signal intensity
Time (s)
1.9_ppm Data
Fit: T1 = 2.277 s





Out[24]:

	Peak	T1(s)
0	7.2_ppm	2.786
1	7.1_ppm	2.619
2	3.7_ppm	2.561
3	2.5_ppm	1.302
4	1.9_ppm	2.277
5	1.5_ppm	0.801
6	0.9_ppm	1.243

Calculate Ideal Delay Time (d_1)

We can calculate an ideal delay time d_1 for future NMR experiments based on the measured T_1 times. The relationship is given by the formula:

$$d_1 + aq = 5 \times T_1$$

or

$$d_1 = (5 \times T_1) - aq$$

Where:

- d_1 is the ideal delay time (seconds).
- aq (acquisition time) = 0.7 seconds (Bruker default for ^1H).
- T_1 is the longitudinal relaxation time (seconds).

Challange

Use the T_1 times in `t1_data['T1(s)']` to calculate ideal delay times d_1 and save the results to `t1_data['D1(s)']`.

Display the `t1_data` DataFrame to check your work.

Syntax for setting a new column in a DataFrame

```
DataFrame_name['Grams'] = DataFrame_name['Kilograms'] / 1000
```

In [25]:
```
# Define the acquisition time (aq)
aq = 0.7 # Bruker default for 1H experiment

# Calculate D1 delay time using vectorized operations
t1_data['D1(s)'] = (5 * t1_data['T1(s)'] - aq)

# Display Dataframe
t1_data
```

Out[25]:

	Peak	T1(s)	D1(s)
0	7.2_ppm	2.786	13.232
1	7.1_ppm	2.619	12.396
2	3.7_ppm	2.561	12.103
3	2.5_ppm	1.302	5.810
4	1.9_ppm	2.277	10.685
5	1.5_ppm	0.801	3.305
6	0.9_ppm	1.243	5.515

Export results

You can export a DataFrame from pandas to a CSV file using the `DataFrame.to_csv()` method.

DataFrame.to_csv()

Write a DataFrame to a CSV file.

Parameters

- **path_or_buf** : `str`, *(default: `None`)*

  The file path or object to write the CSV data. If `None`, the result is returned as a string.

- **index** : `bool`, *(default: `True`)*

  Whether to write row names (index). If `False`, the index is not written.

Let's export the formatted `ibuprofen_inversion_data` DataFrame!

In [26]:
```
ibuprofen_inversion_data.to_csv(path_or_buf='results/ibuprofen_CDCl3_1H_inversion_recovery_data.csv', index=False)
```

Challange

Export the `t1_data` DataFrame to the path `'results/ibuprofen_CDCl3_1H_T1_data.csv'`.

See what happens if you set `index=True`.

In [27]:
```
# Export `t1_data` DataFrame to 'results/ibuprofen_CDCl3_1H_T1_data.csv'
t1_data.to_csv(path_or_buf='results/ibuprofen_CDCl3_1H_T1_data.csv', index=True)
```