

NMR spectrum analysis

1. Получение спектра

1.1 Загрузка данных

```
In [ ]: import numpy as np
import nmrglue as nmr
```

Содержимое папки 1H:

```
In [ ]: %ls 1H
```

Брутто-формулы (как видно наша это C6H12O2 под номером 2):

```
In [ ]: %cat 1H/Брутоформулы.txt
```

```
In [ ]: # nmrglue.variant.read() принимает папку, а не файл
metadata, signal = nmr.variant.read("1H/1H-2-CDCl3.fid")
type(metadata), type(signal)
```

Метаданные это большой словарь с нечитаемыми полями:

```
In [ ]: metadata.keys()
```

Описания полей можно найти только в документации производителя (и то не факт, что это открытая информация), nmrglue к большому сожалению их не приводит. В нём существует функция guess_udic, которая экстрагирует наиболее важные значения и представляет в формате, описываемой документацией nmrglue. К сожалению, у меня не получилось заставить её работать с этими данными

К счастью, нам сообщили названия нужных нам полей:

```
In [ ]: # [key for key in metadata["procpa"].keys() if "fr" in key]
```

```
In [ ]: nu_0 = float(metadata["procpa"]["sfrq"]["values"][0]) * 1e6 # 500 MHz = nu_0
nu_0
```

```
In [ ]: signal_t = float(metadata["procpa"]["at"]["values"][0]) # Signal recording time, sec
signal_t
```

Сигнал это простой одномерный массив комплексных чисел:

```
In [ ]: signal.shape, signal.dtype
```

1.2 Необработанный сигнал

```
In [ ]: import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

```
scale = 5
ratio = (1 + 5 ** 0.5) / 2
```

```
matplotlib.rcParams["figure.figsize"] = scale * ratio, scale
%config InlineBackend.figure_format='retina'

sns.set()
```

```
In [ ]: plt.plot(signal.real, label="Re", linewidth=1, alpha=0.7);
```

```
In [ ]: plt.plot(signal.imag, linewidth=1, alpha=0.7);
```

Мнимые и действительные части очень похожи. Также после 10000 временных единиц график не отличается от нуля

```
In [ ]: plt.plot(np.abs(signal), linewidth=.7, alpha=0.7)
plt.xlim(0, 6000);
```

1.3 Необработанный спектр

Спектр можно получить применив преобразование Фурье к сигналу:

```
In [ ]: spectrum = nmr.process.proc_base.fft(signal)
```

```
In [ ]: plt.plot(spectrum.real);
```

Чтобы соотнести номер точки и соответствующую ей частоту, достаточно знать полное время регистрации сигнала (и детали реализации преобразования):

```
In [ ]: spectrum_max_Hz = (signal.shape[0] - 1) / signal_t
spectrum_max_Hz
```

```
In [ ]: spectrum_Hz = np.linspace(0, spectrum_max_Hz, signal.shape[0])
```

2. Обработка спектра

2.1 Коррекция фазы

2.1.a Встроенная ручная коррекция фазы

Коррекция фазы в ручном режиме встроенной функцией:

```
In [ ]: nmr.process.proc_autophase.manual_ps(spectrum, notebook=True)
```

Низкая скорость ответа никуда не годится, поэтому приведём гораздо более функциональный вариант.

2.1.b Ручная коррекция фазы

Для создание минимального UI будем пользоваться виджетами Jupyter-a. Возможно они требуют установки отдельной библиотеки, возможно она уже установилась при установке самого Jupyter

```
In [ ]: import ipywidgets as widgets
```

Ввод коэффициентов для коррекции фазы:

```
In [ ]: p0_slider = widgets.FloatSlider(
        value=-110,
        min=-180,
        max=180,
        step=.5,
    )
p0_input = widgets.FloatText(
    value=-110,
    min=-180,
    max=180,
    step=.5,
)
p0_label = widgets.Label(value="Zero order phase  $p_0$ , degrees:")

link = widgets.link((p0_slider, 'value'), (p0_input, 'value'))

p0_block = widgets.VBox([p0_label, p0_slider, p0_input])
```

```
In [ ]: p1_slider = widgets.FloatSlider(
        min=-180,
        max=180,
        step=.5,
    )
p1_input = widgets.FloatText(
    value=0,
    min=-180,
    max=180,
    step=.5,
)
p1_label = widgets.Label(value="First order phase  $p_1$ , degrees:")

link = widgets.link((p1_slider, 'value'), (p1_input, 'value'))

p1_block = widgets.VBox([p1_label, p1_slider, p1_input])
```

```
In [ ]: phase_correction = widgets.HBox([
    p0_block,
    p1_block
])
```

Дополнительные параметры

```
In [ ]: logscale = widgets.Checkbox(
    description="Use logscale for y axis (experimental)",
    value=False,
    indent=False,
)

subsampling = widgets.Checkbox(
    description="Operate on every 5th point",
    value=False,
    indent=False,
)

fit_y = widgets.Button(
    description="Fit y axis limits",
)

double_x = widgets.Button(
    description="Expand x axis limits",
    layout=widgets.Layout(width='auto')
)
```

```
options = widgets.VBox([
    subsampling,
    logscale,
    widgets.HBox([fit_y, double_x]),
])
```

Общий блок с виджетами:

```
In [ ]: controls = widgets.Tab(
    children=[phase_correction, options],
    titles=["Phase correction", "Options"]
)
```

Чтобы получить быстрое обновление графиков (и бонусом навигацию), заменим бэкенд `matplotlib` на `ipympl`. Чтобы это можно было сделать, нужно установить библиотеку с таким же названием.

```
In [ ]: %matplotlib ipympl

matplotlib.style.use('fast') # Could theoretically speed up rendering

if 'figure' in globals():
    # With ipympl you have to manually close previous figures,
    # even from previous runs of the same cell
    plt.close(figure)

figure = plt.figure()
line, = plt.plot(nmr.proc_base.ps(spectrum, p0_slider.value, p1_slider.value).real)
xrange = np.arange(spectrum.shape[0])

xrange_subsampled = xrange[::5]
spectrum_subsampled = spectrum[::5]

last_phased_spectrum = np.array([])
last_xrange = np.array([])

def update_ylimits(button_clicked):
    x_min, x_max = plt.xlim()
    visible_spectrum = last_phased_spectrum[
        (x_min <= last_xrange) & (last_xrange <= x_max)
    ]
    y_min, y_max = visible_spectrum.min(), visible_spectrum.max()
    yrange = y_max - y_min
    plt.ylim(y_min - 0.1 * yrange, y_max + 0.1 * yrange)

fit_y.on_click(update_ylimits)

x_min_min, x_max_max = plt.xlim()

def double_x_range(button_clicked):
    x_min, x_max = plt.xlim()
    x_range = x_max - x_min
    new_x_min = max(x_min_min, x_min - 0.5 * x_range)
    new_x_max = min(x_max_max, x_max + 0.5 * x_range)
    plt.xlim(new_x_min, new_x_max)
    update_ylimits(button_clicked)

double_x.on_click(double_x_range)

def redraw_spectrum(p0, p1, subsampling, logscale):
    xrange_ = xrange if not subsampling else xrange_subsampled
    spectrum_ = spectrum if not subsampling else spectrum_subsampled
```

```

phased_spectrum = nmr.proc_base.ps(spectrum_, p0, p1).real

if logscale:
    xrange_ = xrange_[phased_spectrum > 0]
    phased_spectrum = np.log(phased_spectrum[phased_spectrum > 0])

line.set_data(xrange_, phased_spectrum)

figure.canvas.draw()
figure.canvas.flush_events() # Probably not needed

global last_phased_spectrum, last_xrange
last_phased_spectrum = phased_spectrum
last_xrange = xrange_

out = widgets.interactive_output(
    redraw_spectrum,
    {
        'p0': p0_slider,
        'p1': p1_slider,
        'subsampling': subsampling,
        'logscale': logscale,
    }
)

widgets.VBox([controls, out])

```

Значения только что подобранных вручную коэффициентов:

```

In [ ]: p_manual = (p0_slider.value, p1_slider.value)
p_manual

```

2.1.с Автоматическая коррекция фазы

```

In [ ]: _, p_automatic = nmr.process.proc_autophase.autops(spectrum, 'acme', p0=0.0, p1=0.0,
print(f"\nOptimized phases, degrees\n    p0: {p_automatic[0]}\n    p1: {p_automatic[1]}

```

```

In [ ]: autophased_spectrum = nmr.proc_base.ps(spectrum, *p_automatic).real

```

```

In [ ]: if 'figure_autophased' in locals():
    plt.close(figure_autophased)

figure_autophased = plt.figure()

plt.plot(autophased_spectrum);

```

2.1.d Результирующая фаза

```

In [ ]: phase_correction_type = widgets.ToggleButtons(
    options=["Hardcoded", "Manual", "Automatic"]
)

p = (None, None)
p_hardcoded = (-125, 0)
phased_spectrum = []

out = widgets.Output()

def update_phase_correction_type(phase_correction_type):
    global p, phased_spectrum

    if phase_correction_type == "Manual":

```

```

    p = (p0_slider.value, p1_slider.value)
    elif phase_correction_type == "Automatic":
        p = p_automatic
    else:
        p = p_hardcoded

    phased_spectrum = nmr.proc_base.ps(spectrum, *p).real
    print(f"Using '{phase_correction_type}' value for phase correction.")
    print(f"p0: {p[0]:.2f}, p1: {p[1]:.2f}")

out = widgets.interactive_output(
    update_phase_correction_type,
    dict(phase_correction_type=phase_correction_type)
)

widgets.VBox([phase_correction_type, out])

```

2.2 Коррекция базового уровня

Коррекция базового уровня выполняется с помощью функций подмодуля

`nmrglue.proc_bl`. К сожалению их документация слишком скудна чтобы понять, как их правильно использовать, поэтому воспользуемся одной из тех, что не требуют обязательных параметров...

```

In [ ]: based_spectrum = nmr.proc_bl.baseline_corrector(phased_spectrum)

if 'figure_baseline' in globals():
    plt.close(figure_baseline)

figure_baseline = plt.figure()

plt.plot(based_spectrum);

```

Базовая линия действительно улучшилась

2.3 Zero correction

```

In [ ]: chloroform_Hz = spectrum_Hz[10134] # Chloroform peak is located at 10134 (from our sp
chloroform_Hz

In [ ]: choloform_shift = 7.24 # Known value

In [ ]: chem_shifts = choloform_shift - (spectrum_Hz - chloroform_Hz) / nu_0 * 1e6

In [ ]: if 'figure_zero' in globals():
    plt.close(figure_zero)

figure_zero = plt.figure()
plt.xlim(0, 10)
plt.gca().invert_xaxis()

plt.plot(chem_shifts, based_spectrum);

```

3 Анализ спектров

3.1 Интегрирование пиков

```

In [ ]: peak_x_ranges = [

```

```

(15000, 15100),
#(15600, 15800),
(17900, 18300),
(18600, 18900),
(18900, 19200),
(19600, 19900),
]

# Peaks can also be determined automatically:
nmr.analysis.peakpick.pick(based_spectrum, pthres=50000, table=True, algorithm="conne

```

Автоматическое определение пиков занижает ширину некоторых пиков и даёт ложные срабатывания, поэтому будем использовать размеченные вручную.

Интегрирование пиков будем производить вручную. Хотя `nmrglue` и имеет подмодуль `nmrglue.analysis.integration`, он содержит всего две функции (для одномерного и многомерного интегрирования). По неизвестным причинам он в качестве обязательного параметра принимает `filebaseio.unit_conversion` - объект, предназначенный для работы с размерностями. При создании напрямую он требует 5 обязательных параметров, которые не хочется искать. Адекватный способ создания в документации приведён, но зависит от формата данных (так как подтягивает эти параметры из метаданных, которые у каждого формата свои). У варианта этой функции нет. Можно сконвертировать в pipe, но эти значения будут заполнены случайно.

Вместо всего это безобразие вызванного незрелостью библиотеки достаточно просто сделать `np.sum()`.

```

In [ ]: peak_integrals = [
        based_spectrum[x_min:x_max].sum()
        for x_min, x_max in peak_x_ranges
    ]

peak_integrals

```

```

In [ ]: relative_integrals = peak_integrals / peak_integrals[0]

print(*[f"{value:.2f}" for value in relative_integrals], sep="\n")

```

```

In [ ]: mult = 5
print(*[f"{value:.2f}" for value in relative_integrals * mult], sep=" ")

```

Нормировка на 12 водородов в молекуле:

```

In [ ]: mult = 12 / relative_integrals.sum()
print(*[f"{value:.2f}" for value in relative_integrals * mult], sep=" ")

```

3.2 Расщепления

Ну тут все смогут пик приблизить и посмотреть на него... Правда в нескольких местах действительно неочевидно

Результаты/Выводы

ту ту та ту ту то