

NMR spectrum analysis

1. Получение спектра

1.1 Загрузка данных

```
In [1]: import numpy as np
import nmrglue as nmr
```

Содержимое папки 1H :

```
In [2]: %ls 1H

1H-0-CDCl3.fid/  1H-2-CDCl3.fid/  1H-4-CDCl3.fid/  README.md
1H-1-CDCl3.fid/  1H-3-CDCl3.fid/  1H-5-CDCl3.fid/  Брутоформулы.txt
```

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

```
In [3]: with open("1H/Брутоформулы.txt", "r") as file:
        print(*file.readlines(), sep="")

0      C4H8O2
1      C3H6O2
2      C6H12O2
3      C8H8O3
4      C5H10O2
5      C4H10O2
```

```
In [4]: # nmrglue.variant.read() принимает папку, а не файл
metadata, signal = nmr.varian.read("1H/1H-2-CDCl3.fid")
type(metadata), type(signal)
```

```
Out[4]: (dict, numpy.ndarray)
```

Метаданные это большой словарь с нечитаемыми полями:

```
In [5]: metadata.keys()
```

```
Out[5]: dict_keys(['nblocks', 'ntraces', 'np', 'ebytes', 'tbytes', 'bbytes', 'vers_id', 'status', 'nheaders', 'S_DATA', 'S_SPEC', 'S_32', 'S_FLOAT', 'S_COMPLEX', 'S_HYPERCOMPLEX', 'S_ACQPAR', 'S_SECND', 'S_TRANSF', 'S_NP', 'S_NF', 'S_NI', 'S_NI2', 'procpar'])
```

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

```
In [7]: nu_0 = float(metadata["procpar"]["sfrq"]["values"][0]) * 1e6 # 500 MHz = nu_0
nu_0
```

```
Out[7]: 499853183.7
```

```
In [8]: signal_t = float(metadata["procpar"]["at"]["values"][0]) # Signal recording time, second
signal_t
```

```
Out[8]: 2.9999424
```

Сигнал это простой одномерный массив комплексных чисел:

```
In [9]: signal.shape, signal.dtype
```

```
Out[9]: ((24038,), dtype('complex64'))
```

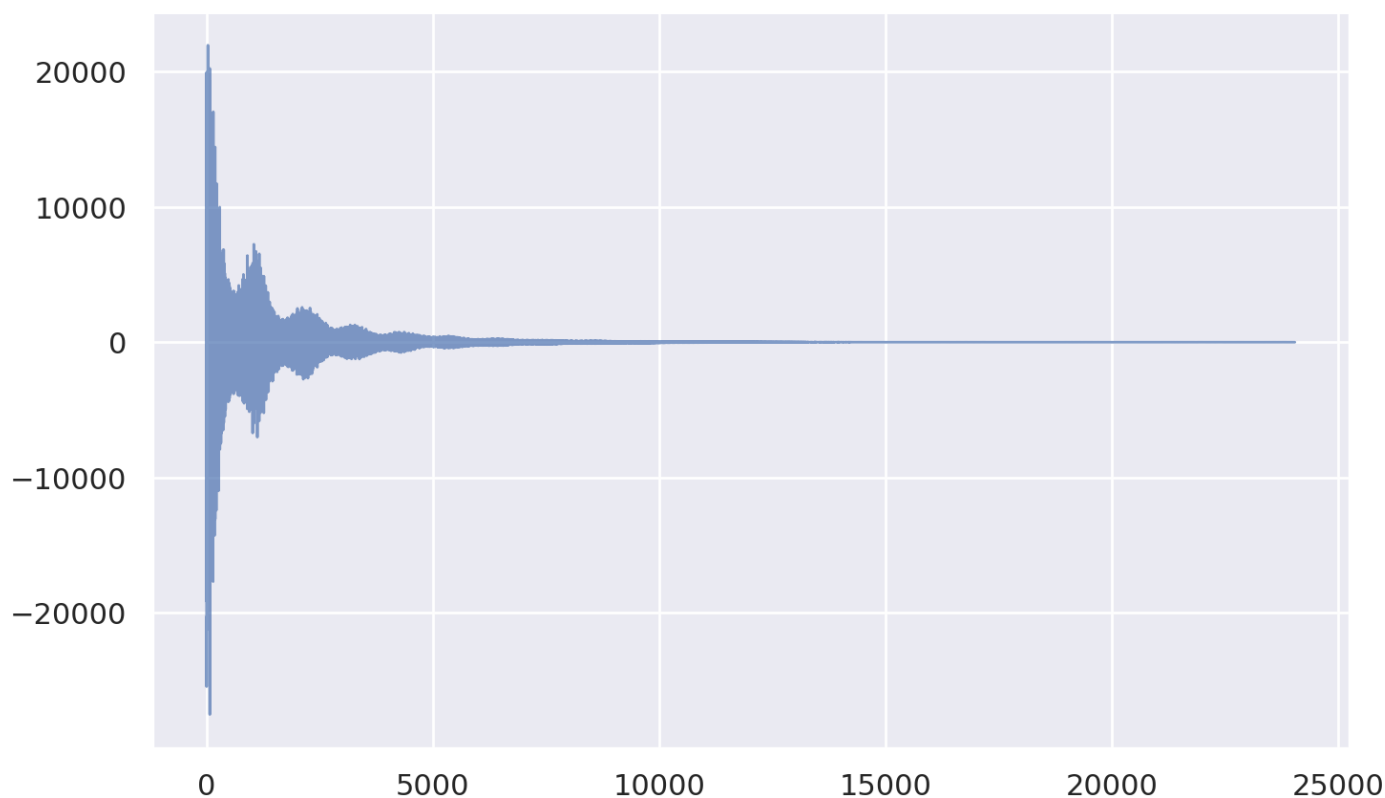
1.2 Необработанный сигнал

```
In [10]: import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

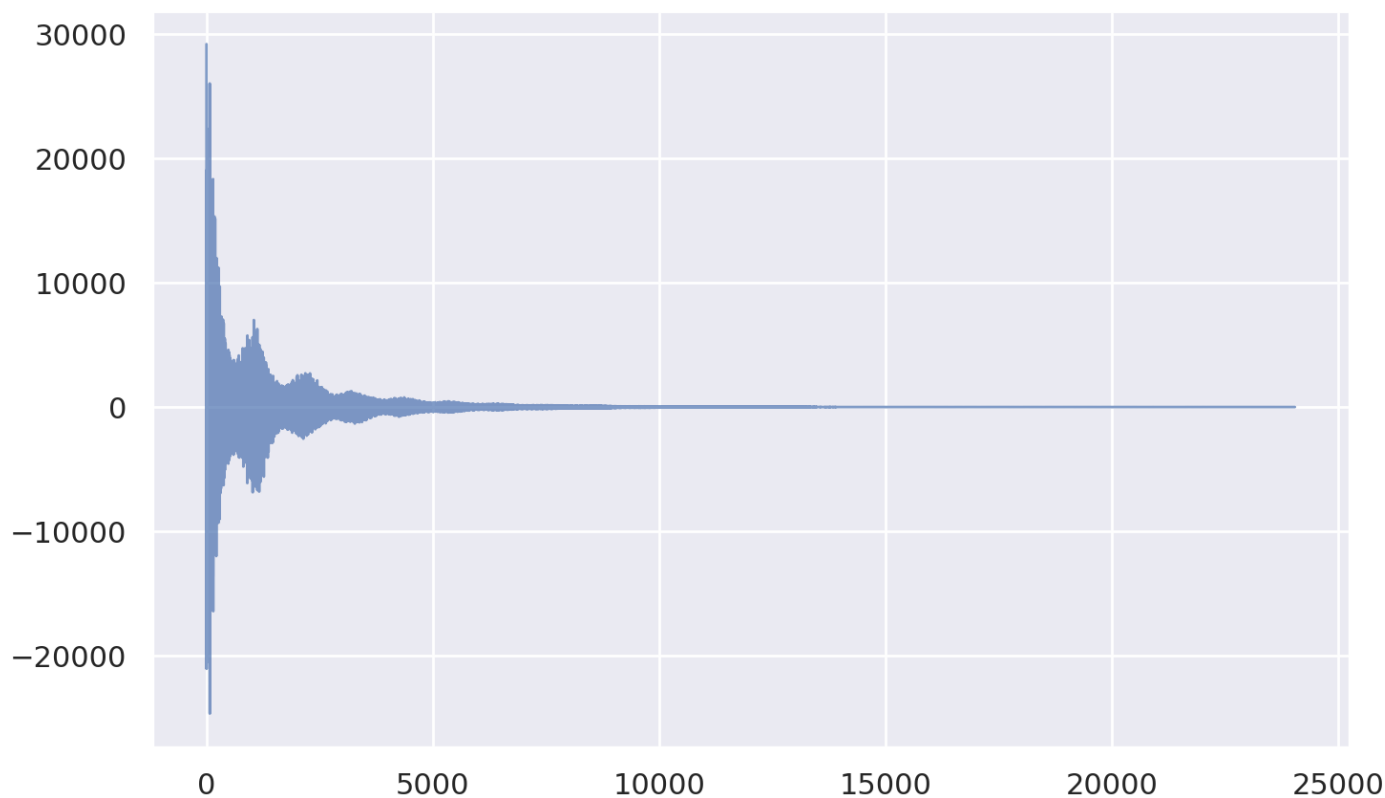
scale = 5
ratio = (1 + 5 ** 0.5) / 2
matplotlib.rcParams["figure.figsize"] = scale * ratio, scale
%config InlineBackend.figure_format='retina'

sns.set()
```

```
In [11]: plt.plot(signal.real, label="Re", linewidth=1, alpha=0.7);
```

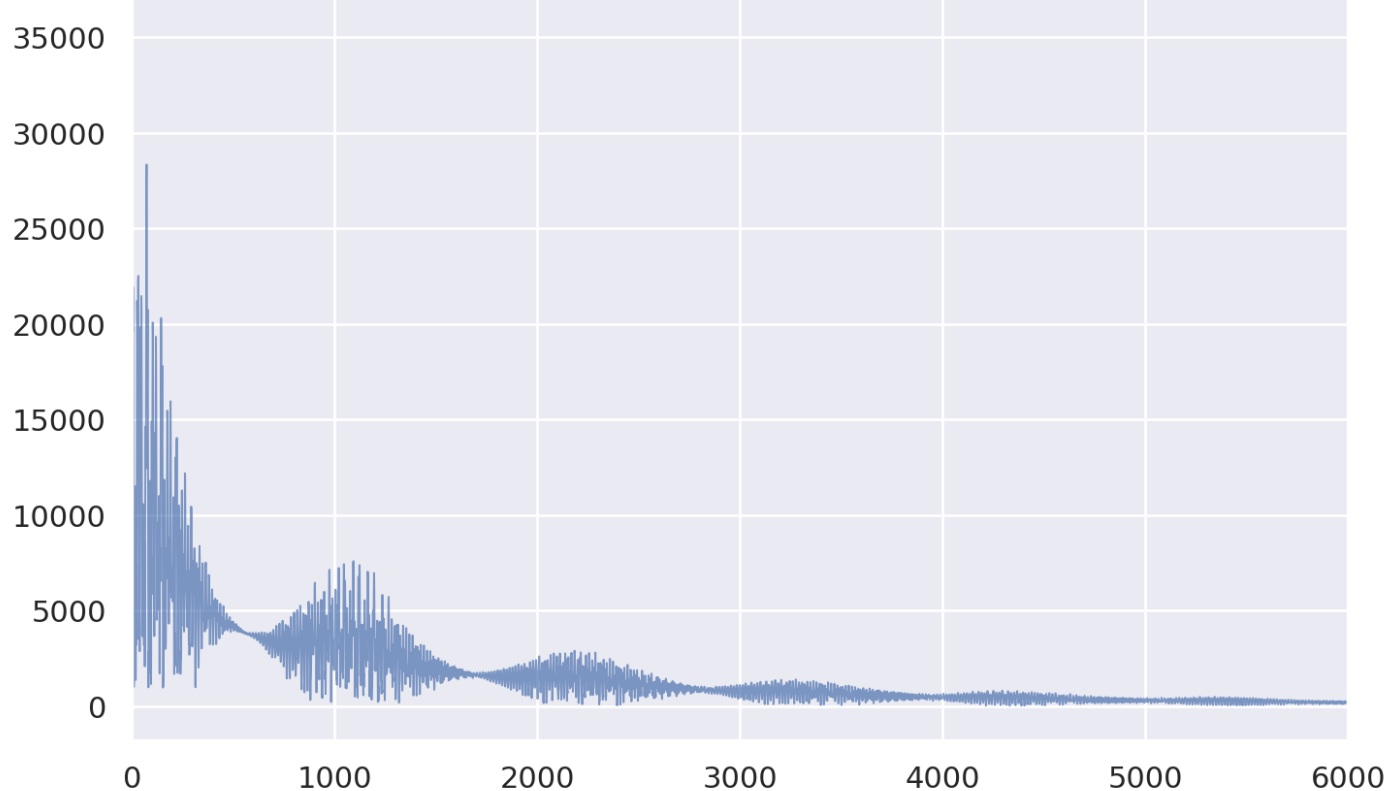


```
In [12]: plt.plot(signal.imag, linewidth=1, alpha=0.7);
```



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

```
In [13]: plt.plot(np.abs(signal), linewidth=.7, alpha=0.7)
plt.xlim(0, 6000);
```

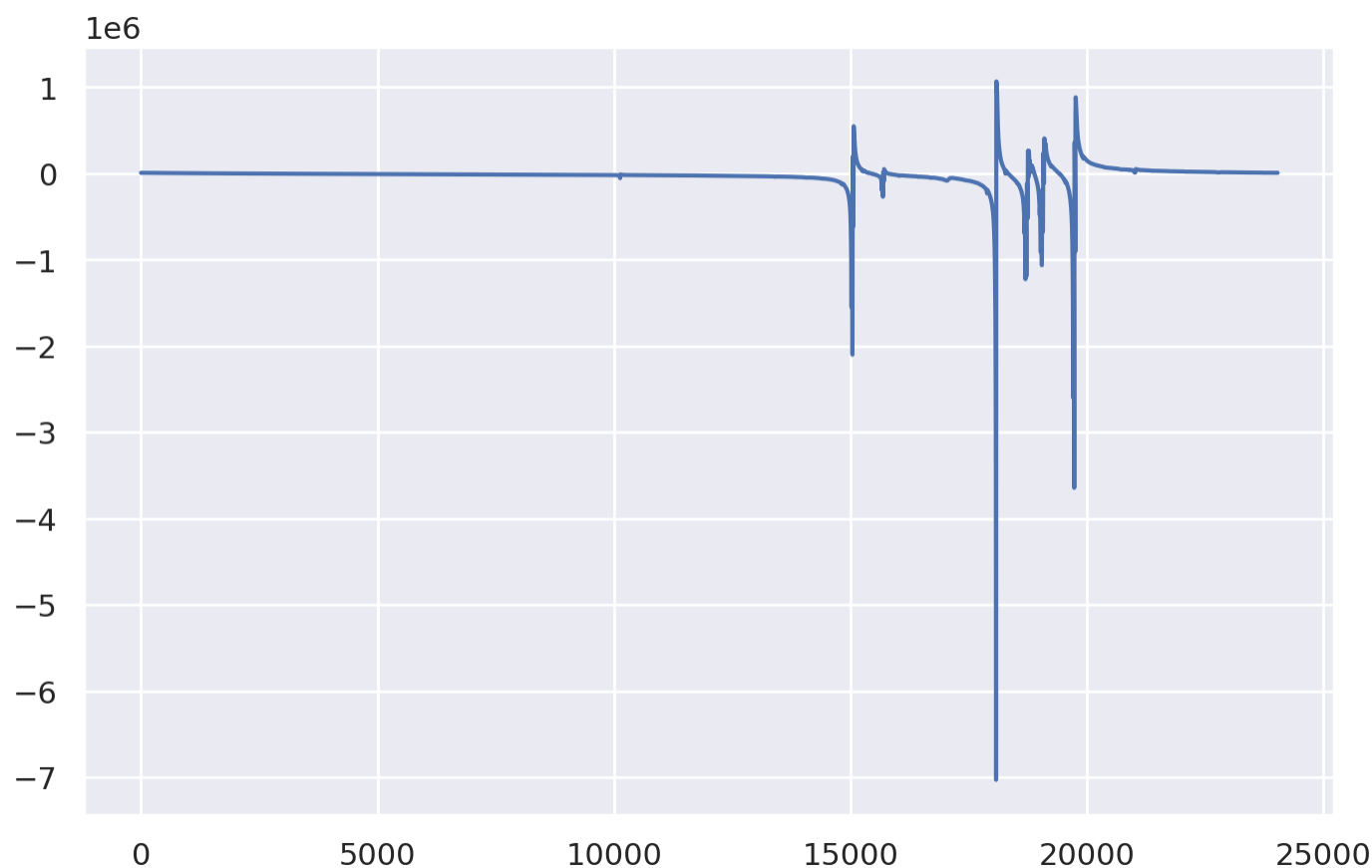


1.3 Необработанный спектр

Спектр можно получить применив преобразование Фурье к сигналу:

```
In [14]: spectrum = nmr.process.proc_base.fft(signal)
```

```
In [15]: plt.plot(spectrum.real);
```



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

```
In [16]: spectrum_max_Hz = (signal.shape[0] - 1) / signal_t
         spectrum_max_Hz
```

```
Out[16]: 8012.487173087056
```

```
In [17]: 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 [18]: nmr.process.proc_autophase.manual_ps(spectrum, notebook=True)

interactive(children=(FloatSlider(value=-0.0015926535897929917, description='phcorr0', ma
x=3.141592653589793, ...
```

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 установки самого Jupyter

```
In [19]: import ipywidgets as widgets
```

Ввод коэффициентов для коррекции фазы:

```
In [20]: 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 [21]: 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 [22]: phase_correction = widgets.HBox([
    p0_block,
    p1_block
])
```

Дополнительные параметры

```
In [23]: 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 [24]: 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 [25]: %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_ylims(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_ylims)

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_ylims(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

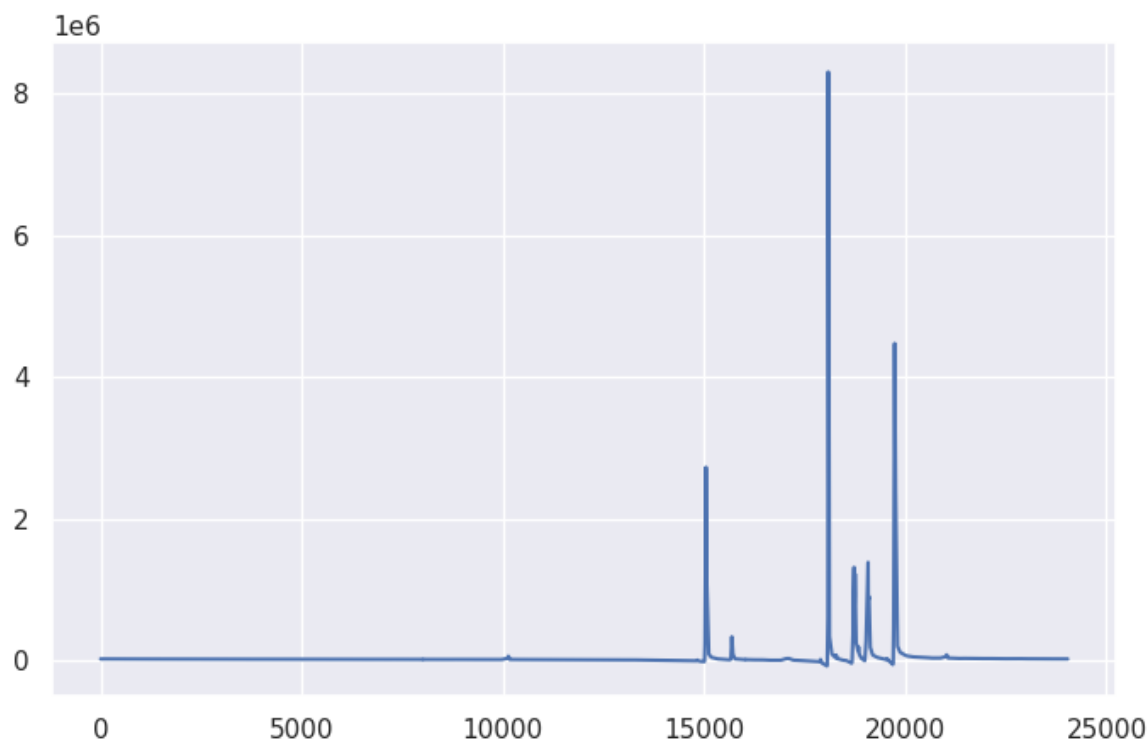
global last_phased_spectrum, last_xrange
last_phased_spectrum = phased_spectrum
last_xrange = xrange_

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

widgets.VBox([controls, out])
```

Out[25]: VBox(children=(Tab(children=(HBox(children=(VBox(children=(Label(value='Zero order phase \$\\;p_0\$, degrees:'),...

Figure



Когда клетка запускается (а не изучается в готовом pdf отчёте), то кроме графика также появляется множество кнопок и ползунков для подбора фазы, а сам график динамически обновляется

Значения только что подобранных вручную коэффициентов:

In [26]: p_manual = (p0_slider.value, p1_slider.value)
p_manual

Out[26]: (-109.99999999999999, 0.0)

2.1.с Автоматическая коррекция фазы

```
In [27]: _, p_automatic = nmr.process.proc_autophase.autops(spectrum, 'acme', p0=0.0, p1=0.0, ret
print(f"\nOptimized phases, degrees\n    p0: {p_automatic[0]}\n    p1: {p_automatic[1]}")
```

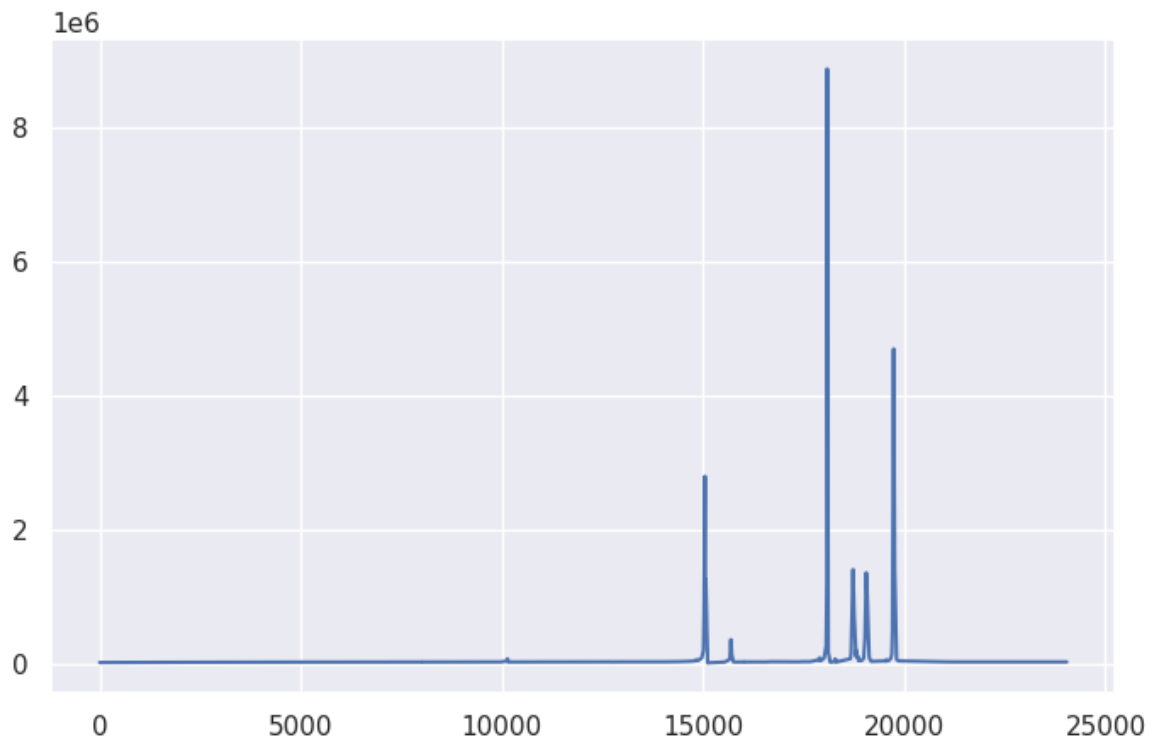
Optimization terminated successfully.
Current function value: 0.000000
Iterations: 66
Function evaluations: 133

Optimized phases, degrees
p0: -143.38389725979368
p1: 22.502472648582405

```
In [28]: autophased_spectrum = nmr.proc_base.ps(spectrum, *p_automatic).real
```

```
In [29]: if 'figure_autophased' in locals():  
    plt.close(figure_autophased)  
  
    figure_autophased = plt.figure()  
  
    plt.plot(autophased_spectrum);
```

Figure



2.1.d Результирующая фаза

```
In [30]: 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])
```

```
Out[30]: VBox(children=(ToggleButtons(options=('Hardcoded', 'Manual', 'Automatic'), value='Hardc
oded'), Output()))
```

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...

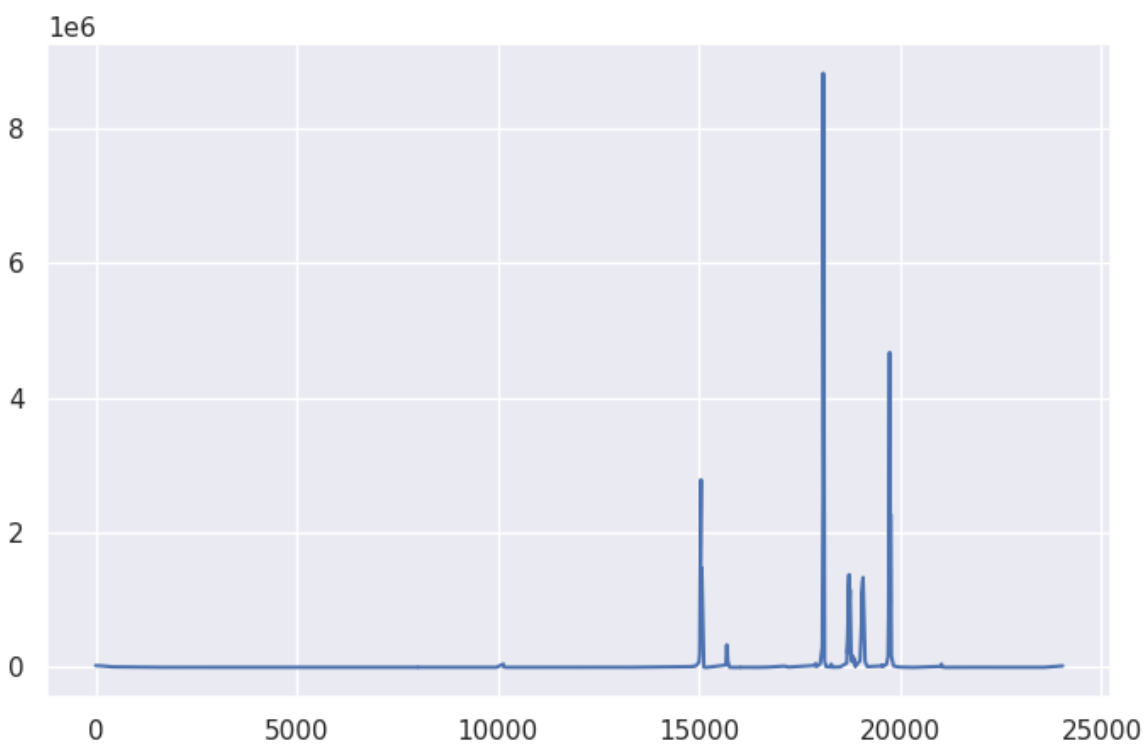
```
In [31]: based_spectrum = nmr.proc_bl.baseline_corrector(phased_spectrum)

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

figure_baseline = plt.figure()

plt.plot(based_spectrum);
```

Figure



Базовая линия действительно улучшилась

2.3 Zero correction

```
In [32]: chloroform_Hz = spectrum_Hz[10134] # Chloroform peak is located at 10134 (from our spect
chloroform_Hz
```

```
Out[32]: 3378.06485884529
```

```
In [33]: choloform_shift = 7.24 # Known value
```

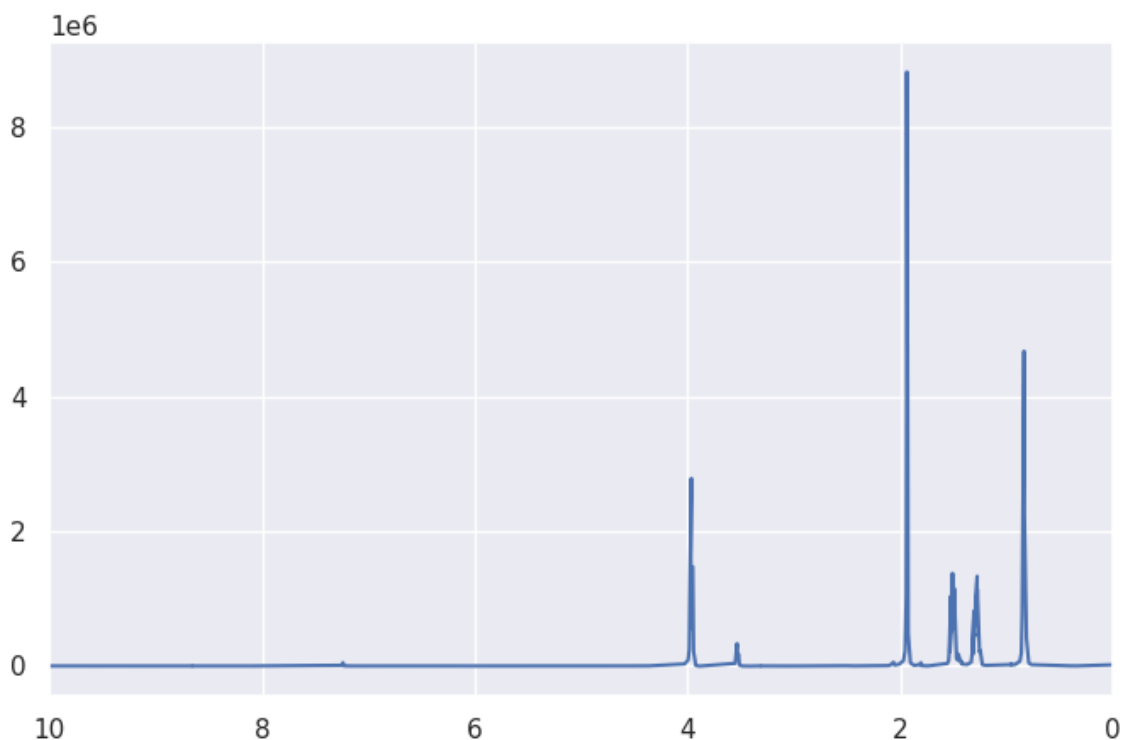
```
In [34]: chem_shifts = choloform_shift - (spectrum_Hz - chloroform_Hz) / nu_0 * 1e6
```

```
In [35]: 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);
```


Figure



3 Анализ спектров

3.1 Интегрирование пиков

```
In [36]: 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="connecte
```

```
Out[36]: rec.array([(15046., 1, 41.1728444 , 5.89258974e+07),
 (15690., 2, 40.52777177, 5.64456616e+06),
 (17893., 3, 0. , 5.69264233e+04),
 (18084., 4, 5.11376582, 8.94375513e+07),
 (18727., 5, 52.03812127, 6.82337219e+07),
 (18854., 6, 6. , 4.13833074e+05),
 (19075., 7, 70.70914333, 6.73099608e+07),
 (19736., 8, 26.4407869 , 1.02809785e+08)],
 dtype=[('X_AXIS', '<f8'), ('cID', '<i8'), ('X_LW', '<f8'), ('VOL', '<f8')])
```

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 вариана этой функции нет. Можно сконвертировать в `pipe`, но эти значения будут заполнены случайно.

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

```
In [37]: peak_integrals = [
    based_spectrum[x_min:x_max].sum()
```

```
        for x_min, x_max in peak_x_ranges
    ]

peak_integrals
```

```
Out[37]: [58056325.13919005,
          93985694.46787688,
          71620800.39459197,
          70827736.34286395,
          106728405.53075895]
```

```
In [38]: relative_integrals = peak_integrals / peak_integrals[0]

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

1.00
1.62
1.23
1.22
1.84
```

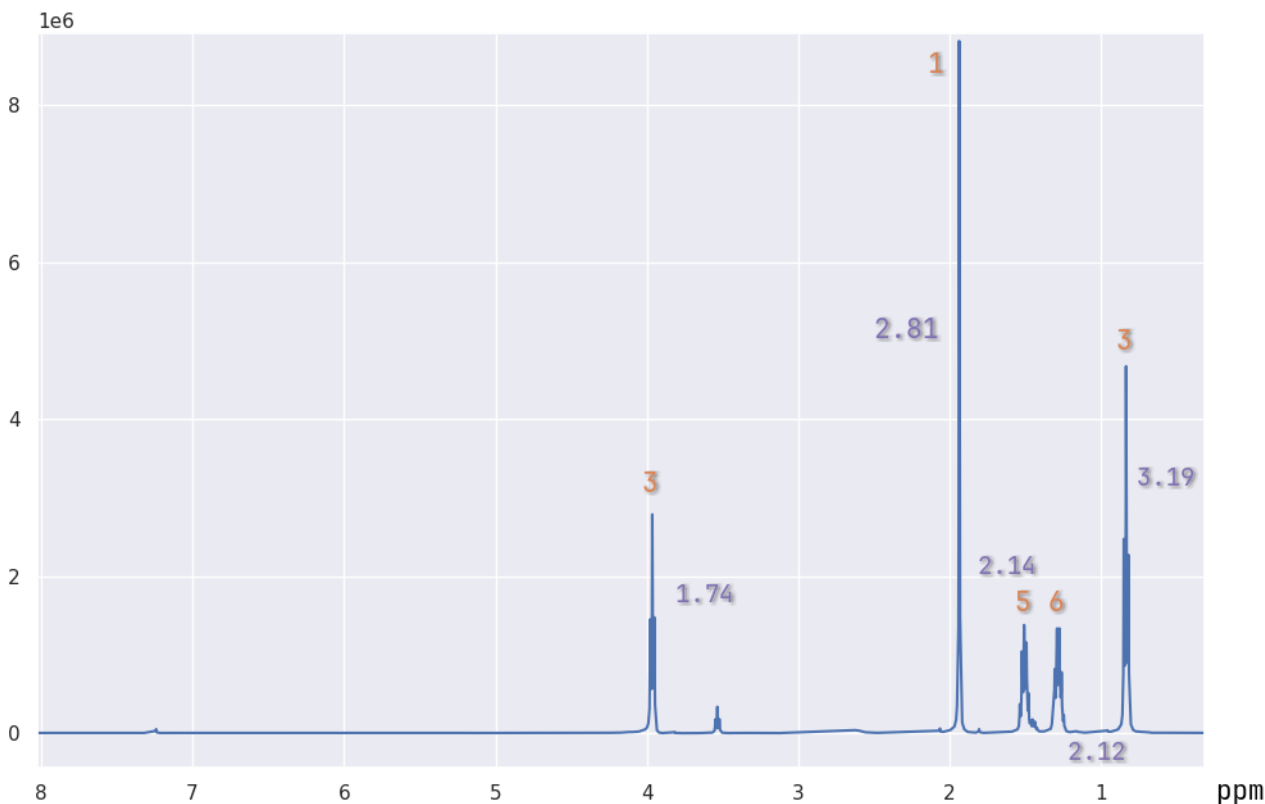
Нормировка на 12 водородов в молекуле:

```
In [39]: mult = 12 / relative_integrals.sum()
print(*[f"{value:.2f}" for value in relative_integrals * mult], sep=" ")

1.74  2.81  2.14  2.12  3.19
```

3.2 Расщепления

Изучая вблизи пики, можно увидеть их расщепления. На картинке приведён весь спектр с подписанными оранжевым величинами расщеплений и фиолетовым интегралами пиков. Интегралы отнормированы на количество водородов в молекуле - 12.



4. Получение структуры молекулы

Идентифицируем пик в районе двух ppm и пик с меньшим чем один ppm. Оба соответствуют примерно трем водородам, т.е. каждый соответствует либо одной группе с тремя водородами, либо трем группам по одному. Меньший чем один ppm пик очень хорошо соответствует группе **-C-CH₃** (либо трём **-C-OH**, но у нас только два кислорода), в то время как для второго может быть много кандидатов, но отсутствие у него расщепления (это означает что у соседних с углеродом атомов не может быть дополнительных водородов) и требование иметь 1 или 3 водорода оставляет только группу **-C(=O)-CH₃**. Таким же образом можно определить группу с химическим сдвигом в районе 4 ppm. Она состоит из двух водородов (или две такие группы по одному), притом она расщепляется. Группы с таким сдвигом соответствуют соседнему кислороду. Так как расщепления есть, то он может быть лишь с одной стороны, и группа выглядит как **-C-**

0-CH2-C- . Расщепление на три пика свидетельствует о двух соседних водородах, и группа увеличивается до -C-0-CH2-CH2- . Эта новая группа CH2 соответствует линии либо с расщеплением на 5, либо на 6 пиков, что означает что рядом с ней с права либо -CH2- , либо уже найденная -CH3 . Во втором случае тогда в -C-0-CH2-CH2-CH3 слева есть ещё H2 и -C0-CH3 , которые собираются в CH3-C0-CH2-0-CH2-CH2-CH3 , у которог на левом CH2 не будет расщепления, хотя ожидается очень большое. Тогда справа стоит -CH2- , и осталось понять как к -C-0-CH2-CH2-CH2- прикрепляются -C-CH3 и -C0-CH3 . Несложно понять, что нужные расщепления получатся если прикрепить -C-CH3 справа, а -C0-CH3 слева. Тогда получится вещество CH3-C0-0-CH2-CH2-CH2-CH3 , известное как бутилацетат.

Butyl Acetate

