toffee Documentation

Release pd-965-change-log

ProCan Software Engineering at Children's Medical Research Inst

Aug 12, 2019

CONTENTS

CHAPTER

ONE

PYTHON EXAMPLES

1.1 `toffee` : A library for fast access to DIA-MS data

Data generated by DIA-MS exists in proprietary data formats that are opaque to open-source software. Typically, a user will convert this data to the open `mzML` format before utilising a tool such as OpenSWATH for analysis. Unfortunately, the conversion of data formats results in a significant expansion in the file size, and as the downstream algorithms are largely I/O bound, this leads to an unneccessary increase in analysis time.

Based on the physics of mass analyzers, it is possible to create a more highly-compressed format whilst still retaining *profile* raw data. Ultimately, this is an implementation of the Intrinsic Mass Spacing (IMS) as described in Schneider (2016) and reflects the fact that mass analyzers a discrete sensors that operate in the time domain, and transform results into the mass over charge domain.

In order to investigate this fully, we took a Sciex SWATH-MS file from a recent run through the ProCan facility, and a Thermo Q-Exacutive DIA RAW file from Peckner2018. Each of these raw files was converted to profile `mzML` using `msconvert` and then using `pyteomics` we extracted the raw m/z data and saved to HDF5 for fast read access.

[1]:
```
%matplotlib inline
import os
import pandas as pd
import numpy as np
import scipy
import h5py
import seaborn as sns
import matplotlib.pyplot as plt

from IPython.display import clear_output
clear_output(wait=False)  # clear import errors that aren't useful for tutorial

sns.set()
sns.set_color_codes()

base_dir = os.environ.get('DIA_TEST_DATA_REPO', None)
assert base_dir is not None
```

1.1.1 Sciex / Time of Flight (TOF) Mass Analyzers

We recognised an opportunity to represent SWATH-MS data in a different form based on the physics of Time of Flight (TOF) detection (see wikipedia).

Time of Flight Physics

We know that the ions, of charge q, are passed through an electric potential U

$$E_p = qU$$

and will result in a kinetic energy

$$E_k = \frac{1}{2}mv^2$$

assuming no acceleration in the TOF tube (of length d), then velocity $v = d/t$. Equating the energies and solving for time we get

$$t = \frac{d}{\sqrt{2U}}\sqrt{\frac{m}{q}}.$$

From here, we can calculate a relationship between Δt and $\Delta\sqrt{m/q}$

$$t_1 = \frac{d}{\sqrt{2U}}\sqrt{m/q_1}$$

$$t_2 = \frac{d}{\sqrt{2U}}\sqrt{m/q_2}$$

$$\Delta t = t_2 - t_1 = \frac{d}{\sqrt{2U}}\left(\sqrt{m/q_2} - \sqrt{m/q_1}\right)$$

Thus, if there is a discrete and constant spacing in Δt due to the physics of the detector, we can assume there will be a discrete and constant spacing of $\sqrt{m/q_2} - \sqrt{m/q_1}$.

$$\Delta t = \frac{1}{\alpha}\Delta\sqrt{m/q}$$

Schneider (2016) refers to this, α, as the Intrinsic Mass Spacing (IMS).

[2]:
```
class CalculatorTOF(object):
    IMS_TRANSFORM = 'sqrt(m/z)'
    FNAME = base_dir + '/ProCan90/ProCan90-M03-01.ms2-050.h5'

    def __init__(self, mz):
        self.mz = mz
        self.root_mz = np.sqrt(mz)

    def calculate_ims_params_directly(self):
        delta_root_mz = np.abs(np.diff(self.root_mz))
        min_delta_root_mz = delta_root_mz.min()
        min_alpha_list = delta_root_mz[delta_root_mz <= 1.4 * min_delta_root_mz]
        alpha = np.median(min_alpha_list)
        alpha_mad = np.median(np.abs(min_alpha_list - alpha))

        beta = -np.mean(self.residual([alpha, 0.0]))
        return alpha, alpha_mad, beta

    @classmethod
    def calculate_deltas_in_ims_space(cls, mz):
        root_mz = np.sqrt(mz)
        return np.diff(root_mz)
```

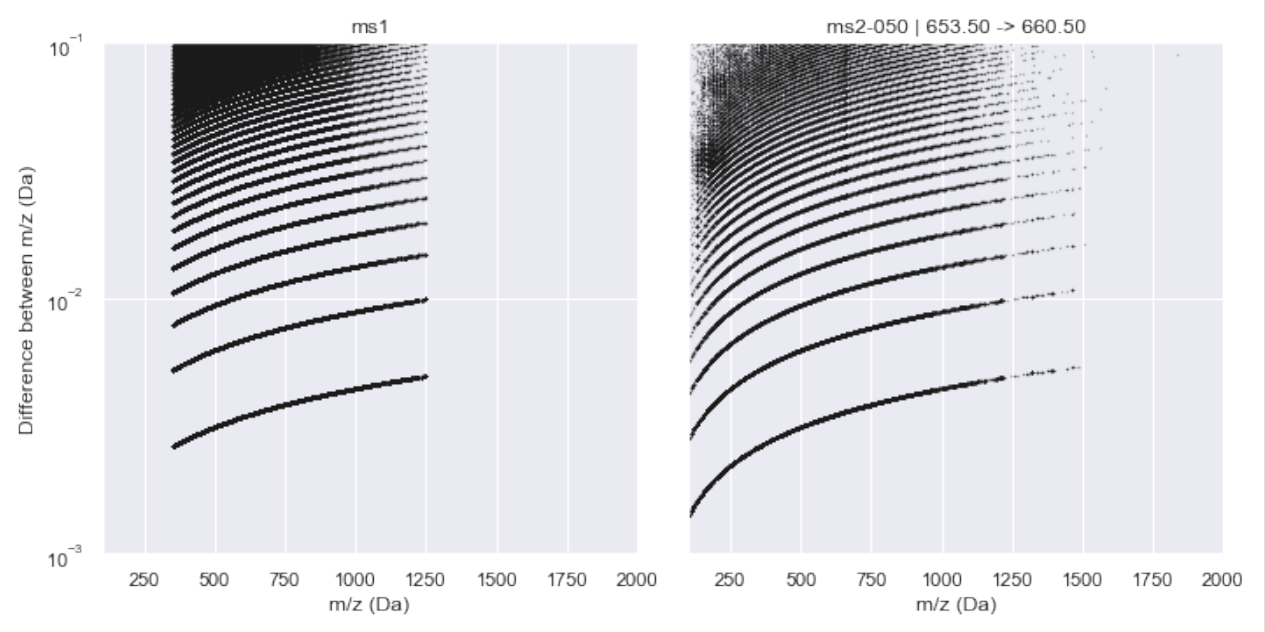
(continues on next page)

(continued from previous page)

```
    def residual_ppm(self, params, offset=0):
        return self.residual(params, offset=offset) * 1e6 / self.mz

    def residual(self, params, offset=0):
        return self._func(params, offset=offset) - self.mz

    def ims_coord(self, params, offset=0):
        a, b = params
        i = (self.root_mz - b) / a
        return np.around(i, decimals=0) + offset

    def _func(self, params, offset=0):
        a, b = params
        i = self.ims_coord(params, offset=offset)
        return (a * i + b) ** 2
```

This raw data is plotted below for both an MS1 window and the middle MS2 window where the x-axis is m/z and the y-axis plots the difference between consecutive m/z values in the raw data.

[3]:
```
def plot_raw_deltas(Calculator, xlim, ylim):
    fname = Calculator.FNAME
    windows = ['ms1', 'ms2-050']
    ncols = len(windows)
    fig, axes = plt.subplots(nrows=1, ncols=ncols, figsize=(5 * ncols, 5), num=1,␣
↪sharey=True, sharex=True)
    if not isinstance(axes, np.ndarray):
        axes = np.array([axes])

    with h5py.File(fname, 'r') as f:
        for (window_name, ax) in zip(windows, axes):
            window = f[window_name]
            title = window_name
            if window_name != 'ms1':
                title += ' | {:.2F} -> {:.2F}'.format(window.attrs['lower'], window.
↪attrs['upper'])
            ax.set_title(title)

            for scan_key in list(window.keys())[500:700]:
                mz = window[scan_key]['mz'].value
                ax.semilogy(mz[1:], np.diff(mz), 'k.', ms=0.5, alpha=0.5)

    for ax in axes:
        ax.set_xlim(xlim)
        ax.set_ylim(ylim)
        ax.set_xlabel('m/z (Da)')
    axes[0].set_ylabel('Difference between m/z (Da)')
    fig.tight_layout()

plot_raw_deltas(CalculatorTOF, xlim=(100, 2000), ylim=(1e-3, 1e-1))
```



It is clear from this plot that a consistent pattern emerges. From visual inspection, this seems to match our hypothesis of a $\sqrt{m/z}$ relationship. Indeed, if we plot this along the y-axis, the Intrinsic Mass Spacing of the TOF spectrum becomes obvious. The figure below mimics Figure 3.2 of Schneider (2016) where the lowest set of data is the IMS and each line above that represents an integer multiple of the IMS. This can be seen clearly by the red lines that show the first 10 multiples of the IMS. The IMS multiplier minus one indicates the number of missing data points in the spectrum (likely due to no ions hitting the detector during this time).

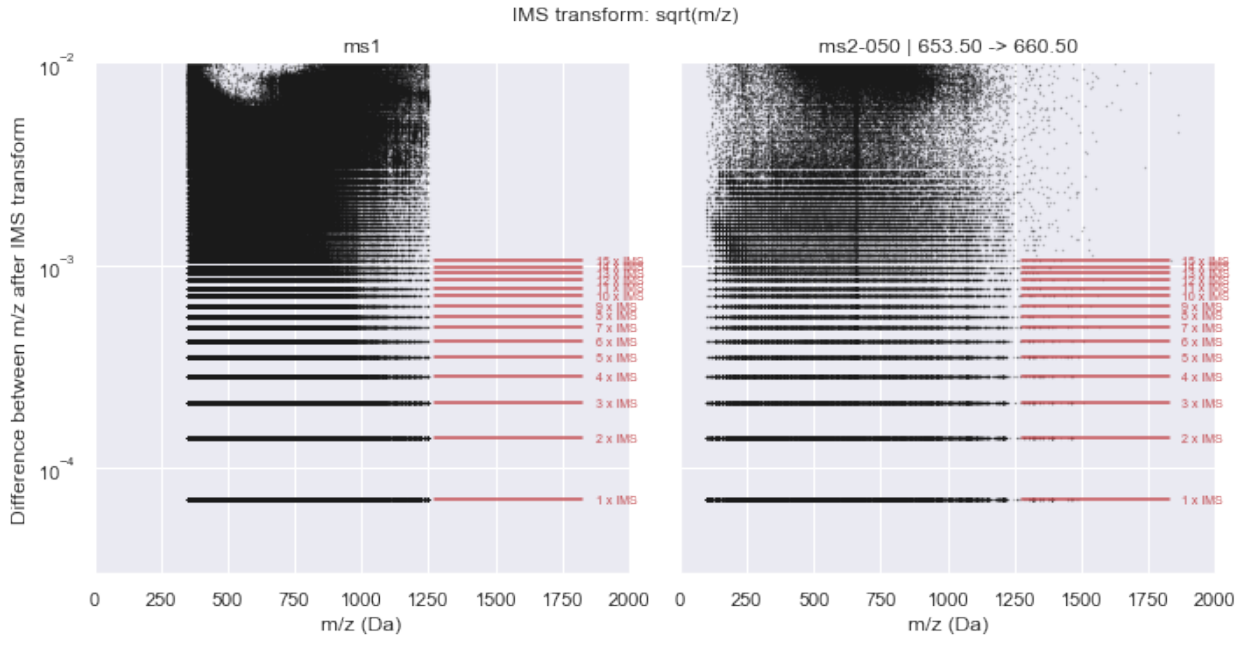
[4]:
```
def plot_in_ims_space(Calculator, xlim, ylim):
    fname = Calculator.FNAME
    windows = ['ms1', 'ms2-050']
    ncols = len(windows)
    fig, axes = plt.subplots(nrows=1, ncols=ncols, figsize=(5 * ncols, 5), num=1,␣
↪sharey=True, sharex=True)
    with h5py.File(fname, 'r') as f:
        for (window_name, ax) in zip(windows, axes):
            window = f[window_name]

            title = window_name
            if window_name != 'ms1':
                title += ' | {:.2F} -> {:.2F}'.format(window.attrs['lower'], window.
↪attrs['upper'])
            ax.set_title(title)

            min_list = list()
            for scan_key in list(window.keys())[500:700]:
                mz = window[scan_key]['mz'].value
                deltas_ims = Calculator.calculate_deltas_in_ims_space(mz)
                ax.semilogy(mz[1:], deltas_ims, 'k.', ms=0.5, alpha=0.5)
                min_val = deltas_ims.min()
                min_list.extend(list(deltas_ims[deltas_ims < 1.1 * min_val]))

            min_list = np.array(min_list)
            min_val = min_list.min()
            min_list = min_list[min_list < 1.1 * min_val]
            min_ims_alpha, stddev_min_ims_alpha = np.mean(min_list), np.std(min_list)
```

(continues on next page)

(continued from previous page)

```
            fmt = '{:<7s} :: IMS alpha = {:.8e} +/- {:.3e}'
            print(fmt.format(window_name, min_ims_alpha, stddev_min_ims_alpha))

            multipliers = [i + 1 for i in range(15)]
            for multiplier in multipliers:
                ims = multiplier * min_ims_alpha
                ax.plot([1275, 1825], [ims, ims], 'r-')
                ax.text(
                    1875,
                    ims,
                    '{} x IMS'.format(multiplier),
                    horizontalalignment='left',
                    verticalalignment='center',
                    color='r',
                    fontsize=7,
                )

            ax.set_xlim(xlim)
            ax.set_ylim(ylim)
            ax.set_xlabel('m/z (Da)')

    axes[0].set_ylabel('Difference between m/z after IMS transform')
    fig.tight_layout()
    fig.suptitle('IMS transform: ' + Calculator.IMS_TRANSFORM, y=1.02)

plot_in_ims_space(CalculatorTOF, xlim=(0, 2000), ylim=(3e-5, 1e-2))
```

```
ms1     :: IMS alpha = 7.01547425e-05 +/- 4.462e-11
ms2-050 :: IMS alpha = 7.01547256e-05 +/- 4.387e-11
```



It is clear from this plot that a consistent pattern emerges. From visual inspection, this seems to match our hypothesis of a $\sqrt{m/z}$ relationship. Indeed, if we plot this along the y-axis, the Intrinsic Mass Spacing of the TOF spectrum becomes obvious. The figure below mimics Figure 3.2 of Schneider (2016) where the lowest set of data is the IMS and each line above that represents an integer multiple of the IMS. This can be seen clearly by the red lines that show

the first 10 multiples of the IMS. The IMS multiplier minus one indicates the number of missing data points in the spectrum (likely due to no ions hitting the detector during this time).

Using IMS for data compression

Knowing that the m/z data is driven by an integer relationship with the IMS allows two opportunities for **near-lossless** data compression:

1. We can drop any m/z values with corresponding zero intensities as these data points can be added back to the spectrum without losing fidelity
2. We can store the IMS value as a double, and convert the m/z vector to a vector of unsigned 32-bit integers that represent an index

The relationship to convert between vectors is summarised as

$$\frac{m}{z} = \left(\alpha_{IMS} i_{\sqrt{m/z}} + \beta_{trunc}\right)^2$$

where α_{IMS} is the IMS value, $i_{\sqrt{m/z}}$ is the unsigned integer index of the m/z value in $\sqrt{m/z}$ space, and β_{trunc} is to control for any truncation error that may be introduced from the mzML file.

In the figure below, we plot out (in black) the mass error, in ppm, of the converted mass over charge compared to the values collected from the mzML. The red dots indicate the mass error if the index ($i_{\sqrt{m/z}}$) is offset by 0.25 in either direction. In general, this shows that the bounds of the indices are tight, but there is a latent mass error that comes from translating the floating-point m/z value into integer space.

1.1.2 Thermo / Orbitrap

Old

$$\Delta t = \frac{1}{\alpha}\left(\Delta\sqrt{m/z} - \beta\right)$$

$$i = \frac{\sqrt{m/z} - \beta}{\alpha} - \gamma$$

$$(i+\gamma)\alpha + \beta = \sqrt{m/z}$$

$$therefore,$$

$$m/z = ((i+\gamma)\alpha + \beta)^2$$

New

$$\Delta t = \frac{1}{\alpha}\left(\Delta\frac{1}{\sqrt{m/z}} - \beta\right)$$

$$i = \frac{1}{\alpha}\left(\frac{1}{\sqrt{m/z}} - \beta\right) - \gamma$$

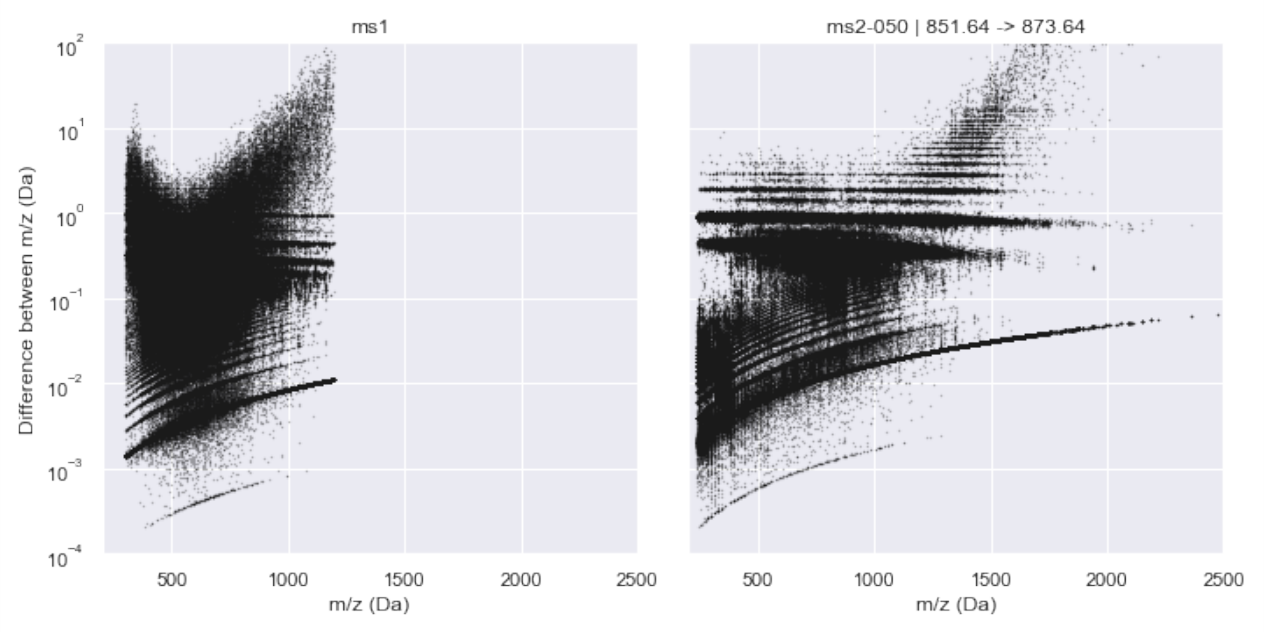
$$(i+\gamma)\alpha + \beta = \frac{1}{\sqrt{m/z}}$$

$$m/z = \frac{1}{((i+\gamma)\alpha + \beta)^2}$$

where γ is the minimum value of the IMS coords such that these are zero-indexed in the file

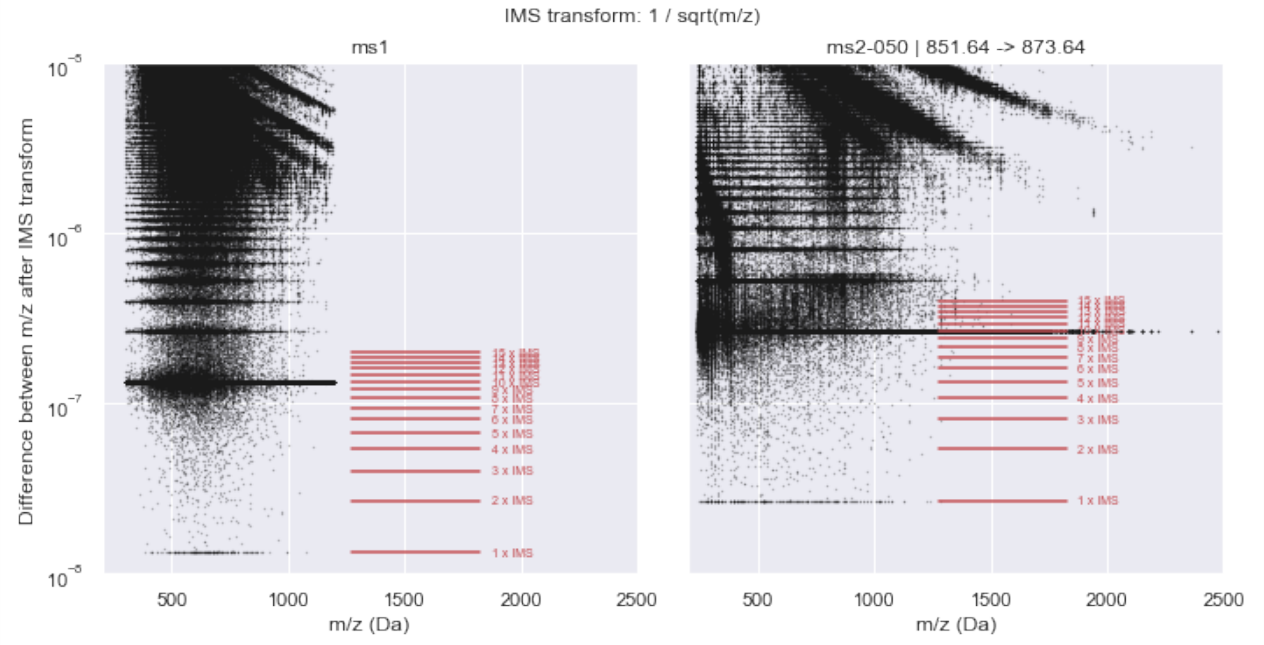
[6]:
```
class CalculatorOrbitrap(object):
    IMS_TRANSFORM = '1 / sqrt(m/z)'
    FNAME = base_dir + '/thermo/CC20160706_P100_Plate34_PC3_T3_P-0034_A01_acq_01.h5'

    def __init__(self, mz):
        self.mz = mz
        self.inverse_root_mz = 1 / np.sqrt(mz)

    def calculate_ims_params_directly(self):
        delta_root_mz = -np.abs(np.diff(self.inverse_root_mz))
        min_delta_root_mz = delta_root_mz.max()
        min_alpha_list = delta_root_mz[delta_root_mz >= 1.4 * min_delta_root_mz]
        alpha = np.median(min_alpha_list)
        alpha_mad = np.median(np.abs(min_alpha_list - alpha))

        beta = -np.mean(self.residual([alpha, 0.0]))
        return alpha, alpha_mad, beta

    @classmethod
    def calculate_deltas_in_ims_space(cls, mz):
        inverse_root_mz = 1 / np.sqrt(mz)
        return -np.diff(inverse_root_mz)

    def residual_ppm(self, params, offset=0):
        return self.residual(params, offset=offset) * 1e6 / self.mz

    def residual(self, params, offset=0):
        return self._func(params, offset=offset) - self.mz

    def ims_coord(self, params, offset=0):
        a, b = params
        i = (self.inverse_root_mz - b) / a
        return np.around(i, decimals=0) + offset

    def _func(self, params, offset=0):
        i = self.ims_coord(params, offset=offset)
        a, b = params
        return (a * i + b) ** -2
```

[7]:
```
plot_raw_deltas(CalculatorOrbitrap, xlim=(200, 2500), ylim=(1e-4, 1e2))
```



[8]:
```
plot_in_ims_space(CalculatorOrbitrap, xlim=(200, 2500), ylim=(1e-8, 1e-5))
```

```
ms1      :: IMS alpha = 1.34410614e-08 +/- 1.244e-10
ms2-050  :: IMS alpha = 2.68852104e-08 +/- 2.746e-10
```



1.1.3 Scan data saved as a point cloud

If we consider the raw data collected within a given window across a full gradient during a SWATH-MS run, each data point represents:

- *Mass over charge:* The abscissa refers to the mass over charge ratio of the peptide ion

- *Retention time:* The ordinate refers to the time at which the peptide ion is elutes from the Liquid Chromatography column
- *Intensity:* The applicate is a count of the number of times a peptide ion is detected on the sensor

We have shown that mass over charge can be represented by an unsigned integer, as can intensity. Furthermore, the retention time for each data point can be considered as an index into an associated (but much smaller) vector of doubles. In this manner, we can save the raw data into a file format as 3 vectors of unsigned 32-bit integers with some associated metadata.

`toffee` takes this exact approach, saving each SWATH window as a group in an HDF5 file. Each group has:

- `double` attributes `lower`, `center`, and `upper` to describe the properties of the window;
- `double` attributes `scanCycleTime` (t_c) and `firstScanRetentionTimeOffset` (t_0) that describe the retention time vector for each window such that $t(i) = t_c i + t_0$ in seconds;
- a `uint32` dataset of `retentionTimeIdx`, of equal length as the retention time vector, that represents indices into the `mzIMSCoord` and `intensity` between which the corresponding value of $t(i)$ is applied;
- `double` attributes `imsAlpha` (α_{IMS}) and `imsBeta` (β_{trunc}) for the Intrinsic Mass Spacing m/z conversion such that $\frac{m}{z} = \left(\alpha_{IMS} i \sqrt{m/z} + \beta_{trunc}\right)^2$; and
- `uint32` datasets of `mzIMSCoord` and `intensity` for the point cloud, with one entry for each point in the cloud.

In taking this approach, we have found a file format with **lossless** compression that can represent a SWATH-MS `mzML` file with a ten-fold decrease in the file size.

File	`wiff` File Size (MB)	`toffee` File Size (MB)	`mzML` File Size (MB)	`toffee` Percent of `wiff`
ProCan90-M03-01	576.06	533.25	5,596.73	92.57
ProCan90-M06-01	779.12	732.04	7,555.10	93.95

Property	`wiff` File Size (GB)	`toffee` File Size (GB)	`mzML` File Size (GB)	`toffee` Percent of `wiff`	`mzML` Percent of `wiff`
count	90	90	90	90	90
mean	1.12	1.08	11.02	96.81	984.17
std	0.27	0.26	2.77	3.13	16.72
min	0.54	0.49	5.14	91.95	952.54
25%	0.93	0.95	8.91	95.77	973.97
50%	1.21	1.17	12.02	96.60	986.83
75%	1.28	1.24	12.72	97.27	995.92
max	1.57	1.54	16.09	106.05	1021.46

[ ]:

1.2 Using `toffee` as a spatial data structure for fast extraction of SWATH-MS data

Analysis of SWATH-MS data often requires us to slice the data through both the abscissa or the ordinate. Many traditional data structures for sparse data like this offer fast searching in one direction, but not both. However, by defining the data as a point cloud, we can insert this into a spatial data structure such as an R-Tree and enable arbitrary searches of the data using axis-aligned boxes. Moreover, as the point cloud is defined in integer terms, we gain performance while also avoiding issues of binary comparisons of floating point numbers.

In `toffee` we have implemented a boost::rtree index in C++; searching the R-Tree through the C++ library can return sparse or dense Eigen matrices that are exposed to python such that data can be manipulated as numpy.ndarray or scipy.sparse.csc_matrix.

[1]:
```
%matplotlib inline
import os
import pandas as pd
import numpy as np
import scipy
import h5py
import seaborn as sns
import matplotlib.pyplot as plt
import toffee
sns.set()
sns.set_color_codes()
```

[2]:
```
toffee.__version__
```

[2]:
```
'0.12.18'
```

1.2.1 Opening a `toffee` file

Opening a `toffee` file is typically the most computationally expensive operation as the data must be loaded from the file into the RTree search index.

[3]:
```
base_dir = os.environ.get('DIA_TEST_DATA_REPO', None)
assert base_dir is not None
fname = base_dir + '/ProCan90/tof/ProCan90-M03-01.tof'
swath_run = toffee.SwathRun(fname)
ms2_name = toffee.ToffeeWriter.ms2Name(50)
```

1.2.2 Accessing summary statistics

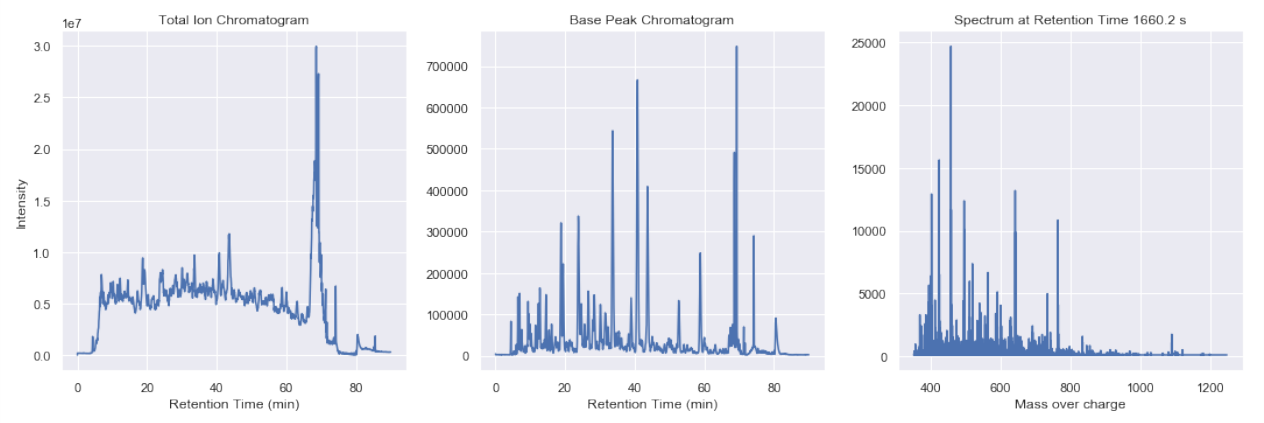
From within this spatial search structure, we can gain high performance access to summary statistics such as total ion chromatogram, base peak chromatogram, or total ion spectra

[4]:
```
%timeit swath_run.loadSwathMapInMemorySpectrumAccess(ms2_name)
ms1_swath_map = swath_run.loadSwathMapInMemorySpectrumAccess(toffee.ToffeeWriter.MS1_
↪NAME)
ms2_swath_map_050 = swath_run.loadSwathMapInMemorySpectrumAccess(ms2_name)
```

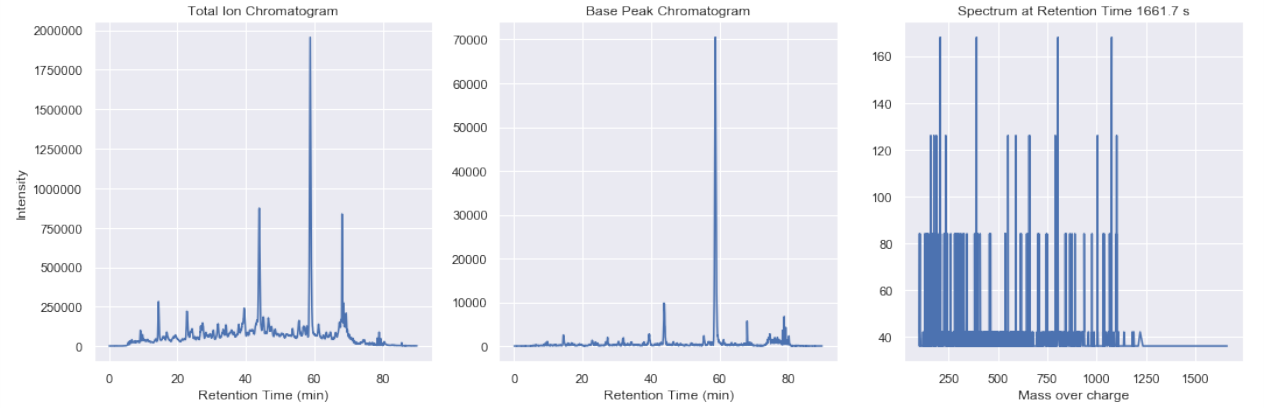
```
109 ms ± 2.49 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[5]:
```
def plot(swath_map):
    print('Summary plots for', swath_map.name())
    fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(15, 5))

    chrom = swath_map.totalIonChromatogram()
    ax1.plot(chrom.retentionTime / 60., chrom.intensities)
    ax1.set_title('Total Ion Chromatogram')
    ax1.set_xlabel('Retention Time (min)')
    ax1.set_ylabel('Intensity')

    chrom = swath_map.basePeakChromatogram()
    ax2.plot(chrom.retentionTime / 60., chrom.intensities)
    ax2.set_title('Base Peak Chromatogram')
    ax2.set_xlabel('Retention Time (min)')

    idx = 500
    spectrum = swath_map.spectrumByIndex(idx)
    ax3.plot(spectrum.massOverCharge, spectrum.intensities)
    ax3.set_title(f'Spectrum at Retention Time {swath_map.retentionTime()[idx]:.1F} s
↪')
    ax3.set_xlabel('Mass over charge')

    fig.tight_layout()
    plt.show()

plot(ms1_swath_map)
plot(ms2_swath_map_050)
```

```
Summary plots for ms1
```




```
Summary plots for ms2-050
```



[6]:
```
%timeit ms2_swath_map_050.totalIonChromatogram()
```

```
943 µs ± 12.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

[7]:
```
%timeit ms2_swath_map_050.basePeakChromatogram()
```

```
1.2 ms ± 19.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

[8]:
```
%timeit ms2_swath_map_050.spectrumByIndex(int(np.random.uniform(0, 1200)))
```

```
15.2 µs ± 86.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

1.2.3 Extracting Chromatograms

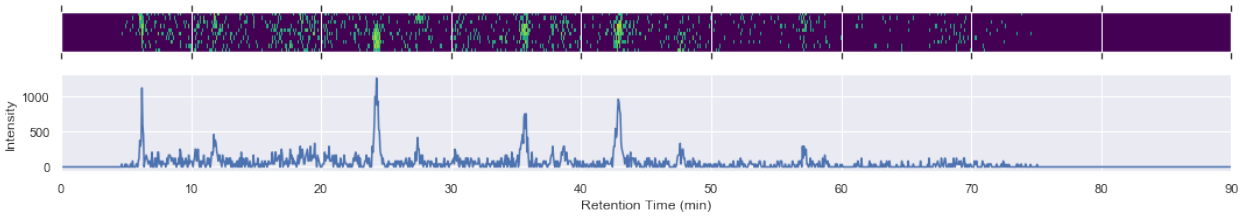
We can also extract specific chromatograms with high performance random access filtering for both mass over charge and retention time ranges.

[9]:
```
%timeit swath_run.loadSwathMap(ms2_name)
ms1_swath_map = swath_run.loadSwathMap(toffee.ToffeeWriter.MS1_NAME)
ms2_swath_map_050 = swath_run.loadSwathMap(ms2_name)
```

```
534 ms ± 22.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[10]:
```
mz_range = toffee.MassOverChargeRangeWithPPMFullWidth(659.35, 50)
rt_range = toffee.RetentionTimeRange(20 * 60.0, 30 * 60.0)  # between 20 & 30 mins
```

[11]:
```
chrom = ms2_swath_map_050.extractedIonChromatogram(mz_range)
xlim = [chrom.retentionTime[0] / 60.0, chrom.retentionTime[-1] / 60.0]

fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(15, 3), sharex=True)

intensities = np.log10(1 + chrom.intensities.T)
ax1.matshow(intensities, extent=xlim + [0, 3], cmap=plt.cm.viridis)
ax1.set_xlim(xlim)
ax1.set_ylabel('m/z (Da)')
ax1.yaxis.set_visible(False)

ax2.plot(chrom.retentionTime / 60.0, np.sum(chrom.intensities, axis=1))
ax2.set_xlim(xlim)
ax2.set_xlabel('Retention Time (min)')
ax2.set_ylabel('Intensity')

fig.tight_layout()
```



[12]:
```
%timeit ms2_swath_map_050.extractedIonChromatogram(mz_range)
%timeit ms2_swath_map_050.extractedIonChromatogramSparse(mz_range)
```

```
63.8 µs ± 5.91 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
57.6 µs ± 15.6 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

[13]:
```
%timeit ms2_swath_map_050.extractedIonChromatogram(\
    toffee.MassOverChargeRangeWithPPMFullWidth(np.random.uniform(100, 2000), 50)\
    )
%timeit ms2_swath_map_050.extractedIonChromatogramSparse(\
    toffee.MassOverChargeRangeWithPPMFullWidth(np.random.uniform(100, 2000), 50)\
    )
```

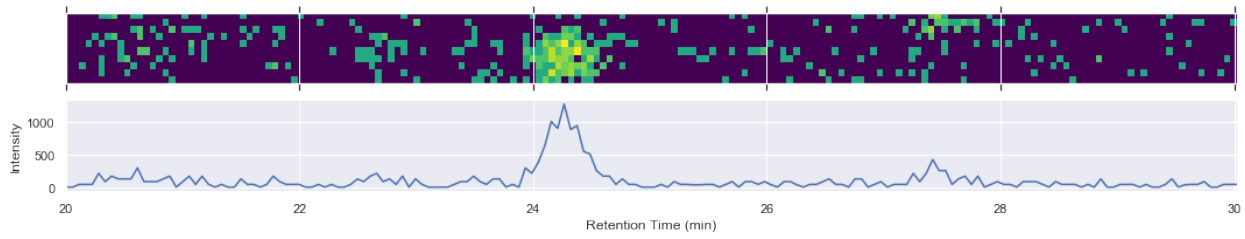
```
26.1 µs ± 968 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
24.2 µs ± 1.99 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

[14]:
```
chrom = ms2_swath_map_050.filteredExtractedIonChromatogram(mz_range, rt_range)
xlim = [chrom.retentionTime[0] / 60.0, chrom.retentionTime[-1] / 60.0]

fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(15, 3), sharex=True)

intensities = np.log10(1 + chrom.intensities.T)
ax1.matshow(intensities, extent=xlim + [0, 0.6], cmap=plt.cm.viridis)
ax1.set_xlim(xlim)
ax1.set_ylabel('m/z (Da)')
ax1.yaxis.set_visible(False)

ax2.plot(chrom.retentionTime / 60.0, np.sum(chrom.intensities, axis=1))
ax2.set_xlim(xlim)
ax2.set_xlabel('Retention Time (min)')
ax2.set_ylabel('Intensity')
```

(continues on next page)

(continued from previous page)

```
fig.tight_layout()
```



[15]:
```
%timeit ms2_swath_map_050.filteredExtractedIonChromatogram(mz_range, rt_range)
%timeit ms2_swath_map_050.filteredExtractedIonChromatogramSparse(mz_range, rt_range)
```

```
11.6 µs ± 93.8 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
10.8 µs ± 61.6 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

[16]:
```
%timeit ms2_swath_map_050.filteredExtractedIonChromatogram(\
    toffee.MassOverChargeRangeWithPPMFullWidth(np.random.uniform(100, 2000), 50),\
    toffee.RetentionTimeRangeWithPixelHalfWidth(np.random.uniform(20 * 60.0, 70 * 60.
↪0), 50)\
    )
%timeit ms2_swath_map_050.filteredExtractedIonChromatogramSparse(\
    toffee.MassOverChargeRangeWithPPMFullWidth(np.random.uniform(100, 2000), 50),\
    toffee.RetentionTimeRangeWithPixelHalfWidth(np.random.uniform(20 * 60.0, 70 * 60.
↪0), 50)\
    )
```

```
13.1 µs ± 1.1 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
12.4 µs ± 890 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

1.3 Interactive Visualisation of Toffee Data

Through the `ToffeeFragmentsPlotter` we can start to explore the raw data contained within a toffee file. These plots are specifically aimed at extracting the same data that one might use when calculating scores in a DIA-pipeline such as `OpenMSToffee`. There are also highly useful for manually validating results, for example when investigating missing values in a broader downstream analysis.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
import os

from IPython.display import SVG

import pandas as pd

import plotly
import plotly.offline as po

import toffee
from toffee.util import calculate_isotope_mz
from toffee.viz import ToffeeFragmentsPlotter
```

(continues on next page)

(continued from previous page)

```
po.init_notebook_mode(connected=True)
```

Data type cannot be displayed: text/html, text/vnd.plotly.v1+html

[3]:
```
toffee.__version__
```

[3]:
```
'0.12.18'
```

[4]:
```
base_dir = os.environ.get('DIA_TEST_DATA_REPO', None)
assert base_dir is not None
```

For this purpose, we are specifically using a spectral library to define the precursor/product mass over charge ratios

[5]:
```
columns = [
    'PrecursorMz',
    'ProductMz',
    'LibraryIntensity',
    'ProteinId',
    'PeptideSequence',
    'ModifiedPeptideSequence',
    'TransitionGroupId',
    'TransitionId'
]
srl = pd.read_csv(base_dir + '/ProCan90/srl/hek_srl.OpenSwath.iRT.tsv', sep='\t')
```

so that we can calculate the C12/C13 isotope offsets and use these in our visualisation.

[6]:
```
n_isotopes = 2
precursors, products = calculate_isotope_mz(
    precursor_and_product_df=srl,
    n_isotopes=n_isotopes,
    sort_by_intensity=True,
)
precursors.set_index('Id', inplace=True)
products.set_index('GroupId', inplace=True)
```

Constructing the plotters is very simple and requires a path to an existing toffee file.

[7]:
```
plotter = ToffeeFragmentsPlotter(
    base_dir + '/ProCan90/tof/ProCan90-M06-07.tof',
    use_ms1=True,
)
```

Finally, we can extract out the mass over charge values for specific transition groups and plot these out. In this example, we have picked out the retention time of the peak so that the plots can be zoomed in

[8]:
```
transition_groups = [
    ('TPVISGGPYYER_2', 1779.903),
    ('LLPSESALLPAPGSPYGR_2', 3304.652),
    ('LLPSESALLPAPGSPYGR_2', None),
    ('ILAAEESVGTMGNR_2', 1664.507),
]
```

(continues on next page)

(continued from previous page)

```
for transition_group, retention_time_line in transition_groups:
    precursors_tg = precursors.loc[transition_group].IsotopeMz.tolist()
    products_tg = products.loc[transition_group].IsotopeMz.tolist()[:6 * (1 + n_
↪isotopes)]

    zoom = True and retention_time_line is not None

#     output_fname = None
    output_fname = f'toffee_fragments.{transition_group}-zoom_{zoom}.svg'
    kwargs = dict(
        psm_name=transition_group,
        number_of_isotopes=n_isotopes,
        retention_time_line=retention_time_line,
        output_fname=output_fname,
    )

    if zoom:
        delta_rt = 600
        kwargs['retention_time_zoom'] = [retention_time_line - delta_rt, retention_
↪time_line + delta_rt]
        kwargs['aspect_scale'] = 0.15

    figure = plotter.create_plotly_figure(
        precursors_tg,
        products_tg,
        **kwargs,
    )
    if output_fname is not None:
        display(SVG(output_fname))
        os.remove(output_fname)
    elif figure is not None:
        po.iplot(figure, show_link=False)
```

1.4 Sub-sampling the data of a `toffee` file to just include standard peptides

In order to build a set of test files with known properties that can be used within a testing framework, we wish to take a full experimental file and extract out a subset of the data. In this example, we are going to take a HEK293 control that has a collection of spike-in retention time standards, extract the data just for those peptides such that this file can be used in regression tests.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
%matplotlib inline
import os
import pandas as pd
import numpy as np
import scipy
import scipy.sparse
```

(continues on next page)

(continued from previous page)

```
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm

import toffee
from toffee import util
from toffee.manipulation import Subsampler

sns.set()
sns.set_color_codes()
```

[3]:
```
toffee.__version__
```

[3]:
```
'0.12.18'
```

1.4.1 Sub-sample using the Spectral Reference Library (SRL)

Open the transition library for the iRT peptides

[4]:
```
base_dir = os.environ.get('DIA_TEST_DATA_REPO', None)
assert base_dir is not None
srl_fname = base_dir + '/ProCan90/srl/hek_srl.OpenSwath.iRT.tsv'
srl = pd.read_csv(srl_fname, sep='\t')
```

and then create the object for sub-sampling a `toffee` file using the peptides in teh reference library.

[5]:
```
tof_fname = base_dir + '/ProCan90/tof/ProCan90-M03-01.tof'
subsampler = Subsampler(
    tof_fname=tof_fname,
    precursor_and_product_df=srl,
    filter_ms2_windows=['ms2-025', 'ms2-041'],
)
```

Running the sub-sampling is as simple as passing in the output file name.

[6]:
```
test_tof_fname = os.path.basename(tof_fname).replace('.tof', '.test.tof')
subsampler.run(test_tof_fname)
```

```
100%|| 2/2 [00:25<00:00, 11.43s/it]
```

1.4.2 Properties of the `toffee` HDF5 file

Using standard HDF tools we can interrogate the new `toffee` file for things such as compression, dataset and attribute names, or even the Intrinsic Mass Spacing parameters.

[7]:
```
!ls -lhF $test_tof_fname
```

```
-rw-r-r-  1 btully  470603294   867K 15 Apr 20:43 ProCan90-M03-01.test.tof
```

[8]:
```
!h5ls -vg $test_tof_fname/ms1
```

```
Opened "ProCan90-M03-01.test.tof" with sec2 driver.
ms1                      Group
    Attribute: IMSAlpha scalar
```

(continues on next page)

(continued from previous page)

```
        Type:      native double
        Data:  7.01548e-05
    Attribute: IMSBeta scalar
        Type:      native double
        Data:  -5.92871e-05
    Attribute: IMSGamma scalar
        Type:      native int
        Data:  266664
    Attribute: firstScanRetentionTimeOffset scalar
        Type:      native double
        Data:  0.17
    Attribute: precursorCenter scalar
        Type:      native double
        Data:  -1
    Attribute: precursorLower scalar
        Type:      native double
        Data:  -1
    Attribute: precursorUpper scalar
        Type:      native double
        Data:  -1
    Attribute: scanCycleTime scalar
        Type:      native double
        Data:  3.32
    Location:  1:405261
    Links:     1
```

[9]:
```
!h5ls -v $test_tof_fname/ms1
```

```
Opened "ProCan90-M03-01.test.tof" with sec2 driver.
IMSAlphaPerScan          Dataset {1627/1627}
    Location:  1:413217
    Links:     1
    Chunks:    {1627} 13016 bytes
    Storage:   13016 logical bytes, 6975 allocated bytes, 186.61% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native double
IMSBetaPerScan           Dataset {1627/1627}
    Location:  1:422560
    Links:     1
    Chunks:    {1627} 13016 bytes
    Storage:   13016 logical bytes, 7470 allocated bytes, 174.24% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native double
imsCoord                 Dataset {257400/257400}
    Location:  1:432398
    Links:     1
    Chunks:    {257400} 1029600 bytes
    Storage:   1029600 logical bytes, 198728 allocated bytes, 518.10% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native unsigned int
intensity                Dataset {257400/257400}
    Location:  1:633670
    Links:     1
    Chunks:    {257400} 1029600 bytes
    Storage:   1029600 logical bytes, 251727 allocated bytes, 409.01% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native unsigned int
```

(continues on next page)

(continued from previous page)

```
retentionTimeIdx         Dataset {1627/1627}
    Location:  1:406565
    Links:     1
    Chunks:    {1627} 6508 bytes
    Storage:   6508 logical bytes, 3956 allocated bytes, 164.51% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native unsigned int
```

[10]:
```
!h5ls -vg $test_tof_fname/ms2-041
```

```
Opened "ProCan90-M03-01.test.tof" with sec2 driver.
ms2-041                  Group
    Attribute: IMSAlpha scalar
        Type:      native double
        Data:  7.01547e-05
    Attribute: IMSBeta scalar
        Type:      native double
        Data:  -3.32516e-05
    Attribute: IMSGamma scalar
        Type:      native int
        Data:  142548
    Attribute: firstScanRetentionTimeOffset scalar
        Type:      native double
        Data:  1.467
    Attribute: precursorCenter scalar
        Type:      native double
        Data:  611.5
    Attribute: precursorLower scalar
        Type:      native double
        Data:  608.5
    Attribute: precursorUpper scalar
        Type:      native double
        Data:  614.5
    Attribute: scanCycleTime scalar
        Type:      native double
        Data:  3.32
    Location:  1:174499
    Links:     1
```

[11]:
```
!h5ls -v $test_tof_fname/ms2-041
```

```
Opened "ProCan90-M03-01.test.tof" with sec2 driver.
IMSAlphaPerScan          Dataset {1627/1627}
    Location:  1:181206
    Links:     1
    Chunks:    {1627} 13016 bytes
    Storage:   13016 logical bytes, 8244 allocated bytes, 157.88% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native double
IMSBetaPerScan           Dataset {1627/1627}
    Location:  1:191818
    Links:     1
    Chunks:    {1627} 13016 bytes
    Storage:   13016 logical bytes, 4755 allocated bytes, 273.73% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native double
imsCoord                 Dataset {96051/96051}
```

(continues on next page)

(continued from previous page)

```
    Location:  1:198941
    Links:     1
    Chunks:    {96051} 384204 bytes
    Storage:   384204 logical bytes, 172435 allocated bytes, 222.81% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native unsigned int
intensity               Dataset {96051/96051}
    Location:  1:373920
    Links:     1
    Chunks:    {96051} 384204 bytes
    Storage:   384204 logical bytes, 28973 allocated bytes, 1326.08% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native unsigned int
retentionTimeIdx        Dataset {1627/1627}
    Location:  1:175803
    Links:     1
    Chunks:    {1627} 6508 bytes
    Storage:   6508 logical bytes, 2707 allocated bytes, 240.41% utilization
    Filter-0:  deflate-1 OPT {6}
    Type:      native unsigned int
```

1.4.3 Compare to the original data

By opening up the file we just created, we can ensure that the data it contains matches exactly the same data as the original file. In doing so, we can prove that the data used in testing later is indistiguisable regardless of which file is being used.

First off, we need a series of functions that check data inside numpy and scipy matrices.

[12]:
```
def sparse_assert_allclose(A, B, atol=1e-8):
    """
    Test to see if two sparse matrices compare equal.
    Modified from: https://stackoverflow.com/a/47771340/758811
    """
    np.testing.assert_array_equal(A.shape, B.shape)
    r1, c1, v1 = scipy.sparse.find(A)
    r2, c2, v2 = scipy.sparse.find(B)
    np.testing.assert_array_equal(r1, r2)
    np.testing.assert_array_equal(c1, c2)
    np.testing.assert_allclose(v1, v2, atol=atol)

def chromatogram_assert_allclose(A, B):
    np.testing.assert_allclose(
        A.retentionTime,
        B.retentionTime
    )
    np.testing.assert_allclose(
        A.massOverCharge,
        B.massOverCharge
    )
    sparse_assert_allclose(
        A.intensities,
        B.intensities
    )
```

For further checking, we can plot the raw data of each file alongside one another

[13]:
```
def plot_raw(original_chrom, test_chrom, mz_value, window):
    def _plot_raw_impl(chrom, ax1, ax2, title, label_y=True):
        xlim = [chrom.retentionTime[0] / 60.0, chrom.retentionTime[-1] / 60.0]
        dense_intensities = chrom.intensities.toarray()

        intensities = np.log10(1 + dense_intensities.T)
        ax1.matshow(intensities, extent=xlim + [0, 6.0], cmap=plt.cm.viridis)
        ax1.set_xlim(xlim)
        ax1.set_ylabel('m/z (Da)')
        ax1.yaxis.set_visible(False)
        ax1.set_title(title, y=0.9)

        ax2.plot(chrom.retentionTime / 60.0, np.sum(dense_intensities, axis=1))
        ax2.set_xlim(xlim)
        ax2.set_xlabel('Retention Time (min)')
        if label_y:
            ax2.set_ylabel('Intensity')

    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 3), sharex=True, sharey=
↪'row')
    _plot_raw_impl(original_chrom, axes[0, 0], axes[1, 0], 'Original Data')
    _plot_raw_impl(test_chrom, axes[0, 1], axes[1, 1], 'Test Data', label_y=False)
    fig.suptitle('{} Data | Mass Over Charge = {:.2F}'.format(window, mz_value))
    fig.tight_layout()
    plt.show()
```

Again, loading the MS1 data is more expensive than others, so we only want to do it once

[14]:
```
test_swath_run = toffee.SwathRun(test_tof_fname)
test_ms1_swath_map = test_swath_run.loadSwathMap(toffee.ToffeeWriter.MS1_NAME)
```

Finally, let's loop through the precursors and products that were used by the sub-samplerand compare the new file to the existing data

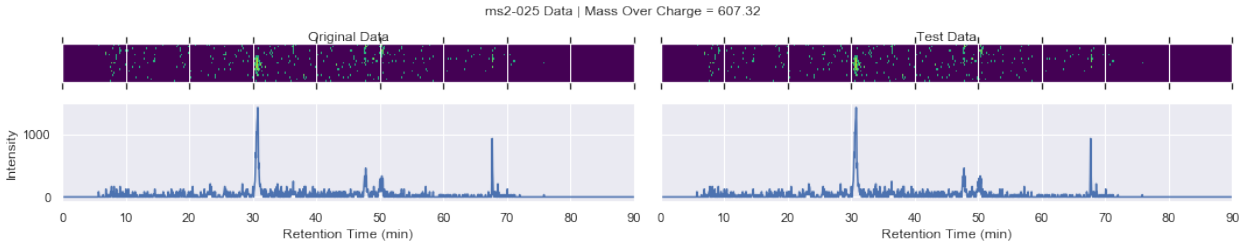
[15]:
```
ppm_width = 100
for k, ms2_name in enumerate(tqdm(sorted(subsampler.precursors.ms2Name.unique().
↪tolist()))):
    # filter the library
    try:
        precursors = subsampler.precursors[subsampler.precursors.ms2Name == ms2_name]
        products = subsampler.products[subsampler.products.ms2Name == ms2_name]
    except KeyError:
        # this will happen if the window is not in the library
        continue

    # load the MS2 swath map
    ms2_swath_map = subsampler.swath_run.loadSwathMap(ms2_name)
    test_ms2_swath_map = test_swath_run.loadSwathMap(ms2_name)

    # collect the MS1 data
    for i, (_, row) in enumerate(precursors.iterrows()):
        mz = row.IsotopeMz
        mz_range = toffee.MassOverChargeRangeWithPPMFullWidth(mz, ppm_width)
        ms1_xic = subsampler.ms1_swath_map.extractedIonChromatogramSparse(mz_range)
        test_ms1_xic = test_ms1_swath_map.extractedIonChromatogramSparse(mz_range)
        if i == 0 and k < 4:
            plot_raw(ms1_xic, test_ms1_xic, mz, window='MS1')
```

(continues on next page)

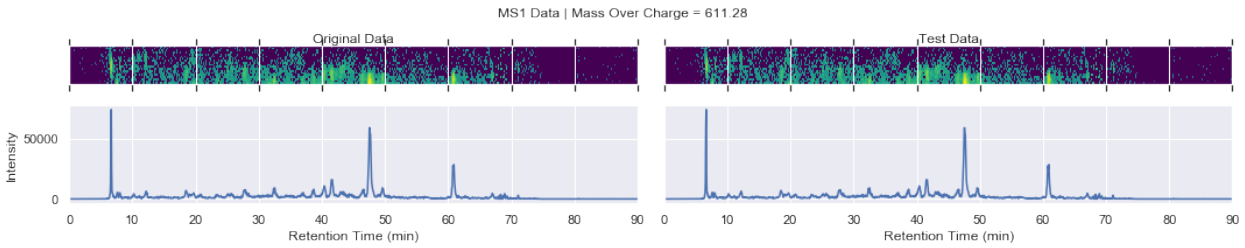
(continued from previous page)

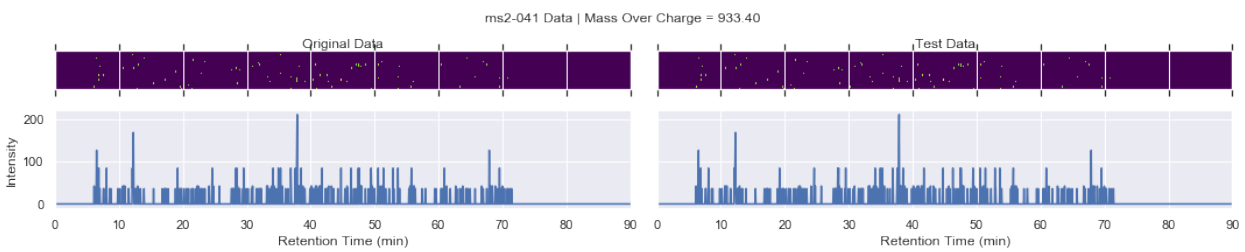
```
        try:
            chromatogram_assert_allclose(ms1_xic, test_ms1_xic)
        except AssertionError:
            plot_raw(ms1_xic, test_ms1_xic, mz, window=f'Error! MS1')

    # collect the MS2 data
    for i, (_, row) in enumerate(products.iterrows()):
        mz = row.IsotopeMz
        mz_range = toffee.MassOverChargeRangeWithPPMFullWidth(mz, ppm_width)
        ms2_xic = ms2_swath_map.extractedIonChromatogramSparse(mz_range)
        test_ms2_xic = test_ms2_swath_map.extractedIonChromatogramSparse(mz_range)
        if i == 0 and k < 4:
            plot_raw(ms2_xic, test_ms2_xic, mz, window=ms2_name)
        try:
            chromatogram_assert_allclose(ms2_xic, test_ms2_xic)
        except AssertionError:
            plot_raw(ms2_xic, test_ms2_xic, mz, window=f'Error! {ms2_name}')
```

```
  0%|          | 0/2 [00:00<?, ?it/s]
```




```
 50%|     | 1/2 [00:01<00:01,  1.96s/it]
```





```
100%|| 2/2 [00:03<00:00,  1.82s/it]
```

And now, clean up

[16]:
```
if os.path.isfile(test_tof_fname):
    os.remove(test_tof_fname)
```

1.5 Re-Quantifying detections using toffee and 2D modified Gaussian

Through the many visualisations of toffee data, and appreciation for the TOF detector, one recognises that the data is roughly gaussian in both retention time and m/z index space. Indeed, as we will see below, it is possible to model the raw data as an analytic function to improve the quantification results. In doing so, the least-squares method used to fit the function to the data mimicks a high-frequency filter and performs basic noise removal. By fixing the retention time of the analytic function for all fragments, it is also possible to pull apart co-eluting peaks and count only the contribution from the peak of interest.

The elution profile from the LC column gives a log-normal distribution that is skewed towards the left, while the data is normally distributed in index m/z space that we attribute to the distribution of kinetic energy that individual ions obtain within the TOF mass analyser. These observations imply that we can perform an optimization to find the peak location, spread, and amplitude of the gaussian functions for each fragment, j.

$$F_j\left(\sigma_t, \sigma_m, t_0, m_{0,j}, a_j, c_j\right) = \frac{1}{t} \cdot \frac{a_j}{\sigma_t\sqrt{2\pi}} \exp\left[-\frac{(\log t - \log t_0)^2}{2\sigma_t^2} - \frac{(m - m_{0,j})^2}{2\sigma_m^2}\right] + C_j\left(\sigma_t, t_0, c_j\right)$$

with chemical noise, for MS1 signal only, defined as

$$C_j\left(\sigma_m, m_{0,j}, c_j\right) = c_j \exp\left[-\frac{(m - m_{0,j})^2}{2\sigma_m^2}\right]$$

and the minimisation function

$$G\left(\sigma_t, \sigma_m, t_0, \vec{m_0}, \vec{a}, \vec{c}\right) = \sum_j G_j \text{ , where } G_j = F_j\left(\sigma_t, \sigma_m, t_0, m_{0,j}, a_j, c_j\right) - I_j$$

where - t and m represent the retention time and $i_{\sqrt{m/z}}$ space respectively - t_0 and $\vec{m_0}$ are the peak location, where we assume that the retention time for all fragments must be constant, but the mass offset is allowed to be different for each one to account for calibration offsets - σ_t, σ_m are the spread of each gaussian - $\vec{a}$ is the amplitude for each fragment - $\vec{c}$ is the amplitude of chemical noise for each fragment, and - I_j is the raw intensity data for a given fragment.

By building a model such as a 2D gaussian, we are then able to calculate the volume under the curve, with a hypothesis that this should give a more robust measure of the intensity from sample to sample.

Furthermore, to improve robustness during least-squares fitting, we apply the following transforms to the parameters:

- $f(n, x) = n(1 + \tanh x)/2$ is a sigmoid function for constraining x to the range of $0, n$
- $\sigma_t = \sigma_t'^2$, where σ_t' is the value used during optimisation
- $\sigma_m = \sigma_m'^2$, where σ_m' is the value used during optimisation
- $t_0 = f(num_t, t_0')$, where t_0' is the value used during optimisation, and num_t is the number of retention time pixels. This constrains t_0 to fall within the valid range of retention times
- $m_0 = f(num_m, m_0')$, where m_0' is the value used during optimisation, and num_m is the number of index m/z pixels. This constrains m_0 to fall within the valid range of index m/z values

- $a_j = a'^2_j$, where a'_j is the value used during optimisation
- $c_j = c'^2_j$, where c'_j is the value used during optimisation

[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2

import os
import datetime

from IPython.display import SVG, Image, display

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

import toffee
from toffee import requant
from toffee.util import calculate_isotope_mz
from toffee.viz import ToffeeFragmentsPlotter

from tqdm import tqdm

base_dir = os.environ.get('DIA_TEST_DATA_REPO', None)
assert base_dir is not None

sns.set()
```

[2]:
```
print(' Analysis date:', datetime.datetime.now())
print('toffee version:', toffee.__version__)
```

```
 Analysis date: 2019-08-12 10:15:31.826510
toffee version: 0.14.2
```

1.5.1 Fitting the data to an analytic model using the C++ library

In `toffee.requant` there are two classes that enable the requantification to be performed using `PyProphet` results as the input.

Swath Gold Standard data

[3]:
```
sgs_files = [2, 5, 8, 10]
force = False

sgs_quantifiers = {}
sgs_requantified = []
for f in tqdm(sgs_files):
    name = f'{f:03d}'
    sample = f'napedro_l120224_{name}_sw'
    output = sample + '.csv.gz'
```

(continues on next page)

(continued from previous page)

```
    tof_fname = f'{base_dir}/SwathGoldStandard/tof/{sample}.tof'
    pyprophet_fname = f'{base_dir}/SwathGoldStandard/osw/{sample}.osw'
    q = requant.PyProphetSQLiteRequantifier(
        output_filename=output,
        toffee_filename=tof_fname,
        pyprophet_filename=pyprophet_fname,
        max_protein_q_value_rs=1.0,
        max_protein_q_value_experiment_wide=1.0,
        max_protein_q_value_global=1.0,
    )
    sgs_quantifiers[f] = q

    if not os.path.exists(output) or force:
        q.run()

    df = pd.read_csv(output)
    df['InjectionName'] = name
    df['PQM'] = df.FullPeptideName + '_' + df.Charge.map(str)
    sgs_requantified.append(df)

sgs_requantified = pd.concat(sgs_requantified)
sgs_requantified.head()
```

```
100%|| 4/4 [00:00<00:00, 10.48it/s]
```

[3]:

	run_id	peptide_id	Sequence	FullPeptideName	Charge	peak_group_rank	RT	Intensity	m_score	m_score_peptide_run_specific	...	MS1Intensity	MS2Intensity	ModelParamSigmaRT	ModelParamSigmaMz	ModelParamRT0	ModelParamMz0MS1	ModelParamMz0MS2	ModelParamAmplitudes
0	1368487973123354311	1	AAEDFTLLVK	AAEDFTLLVK(UniMod:259)	2	1	4209.56	80.0	0.032005	0.032005	...	5.028338	1.668722	0.064002	11.891527	15.757658	14.731366	12.945118	8.66231,8.142178.14217;1.04211.0421,2.753782.7...
1	1368487973123354311	4	AAGASAQVLGQEGK	AAGASAQVLGQEGK(UniMod:259)	2	1	2017.42	822.0	0.003899	0.003899	...	76.928972	15.230757	0.088230	5.231933	14.949945	20.636998	20.908421	305.804,170.775170.775;41.441141.4411,16.10781...
2	1368487973123354311	17	ADSTGTLVITDPTR	ADSTGTLVITDPTR(UniMod:267)	2	1	3105.00	180.0	0.003899	0.003899	...	8.074463	2.976146	0.065825	2.685659	15.040877	20.335423	21.117461	49.0696,42.791442.7914;2.926292.92629,8.41438....
3	1368487973123354311	22	AEVAALAAENK	AEVAALAAENK(UniMod:259)	2	1	2221.28	2658.0	0.003899	0.003899	...	189.337273	57.394183	0.082998	5.633617	15.131340	20.949316	21.025522	810.276,374.571374.571;70.898270.8982,87.69587...
4	1368487973123354311	26	AFGYYGPLR	AFGYYGPLR(UniMod:267)	2	1	3723.90	320.0	0.034796	0.034796	...	19.167878	9.666694	0.110380	9.718412	14.489525	23.953597		

(continues on next page)

(continued from previous page)

```
4         19.389373  35.5049,38.795338.7953;8.423798.42379,7.632937...

   InjectionName                          PQM
0            002      AAEDFTLLVK(UniMod:259)_2
1            002  AAGASAQVLGQEGK(UniMod:259)_2
2            002  ADSTGTLVITDPTR(UniMod:267)_2
3            002     AEVAALAAENK(UniMod:259)_2
4            002       AFGYYGPLR(UniMod:267)_2

[5 rows x 24 columns]
```

ProCan90 data

[4]:
```
procan90_files = ['03-01', '06-07']
force = False

procan90_quantifiers = {}
procan90_requantified = []
for f in tqdm(procan90_files):
    sample = f'ProCan90-M{f}'
    output = sample + '.csv.gz'
    tof_fname = f'{base_dir}/ProCan90/tof/{sample}.tof'
    pyprophet_fname = f'{base_dir}/ProCan90/osw/{sample}.osw'
    q = requant.PyProphetSQLiteRequantifier(
        output_filename=output,
        toffee_filename=tof_fname,
        pyprophet_filename=pyprophet_fname,
    )
    procan90_quantifiers[f] = q

    if not os.path.exists(output) or force:
        q.run()

    df = pd.read_csv(output)
    df['InjectionName'] = sample
    df['PQM'] = df.FullPeptideName + '_' + df.Charge.map(str)
    procan90_requantified.append(df)

procan90_requantified = pd.concat(procan90_requantified)
procan90_requantified.head()
```

```
100%|| 2/2 [00:14<00:00,  7.43s/it]
```

[4]:
```
                run_id  peptide_id  \
0  5145023687975356512           0
1  5145023687975356512           3
2  5145023687975356512          13
3  5145023687975356512          16
4  5145023687975356512          19

                                      Sequence  \
0                          AAAAAAAAAAAAAAAAGAGAGAK
1                                AAAAAAAAAAGRVGM
2  AAAAAADPNAAWAAYYSHYYQQPPGPVPGPAPAPAAPPAQGEPPQP...
3                                     AAAAADLANR
4                                AAAAASAAGPGGLVAGK
```

(continues on next page)

(continued from previous page)

```
                                  FullPeptideName  Charge  peak_group_rank  \
0                            AAAAAAAAAAAAAAAGAGAGAK       3                1
1                                  AAAAAAAAAAGRVGM       2                1
2  AAAAAADPNAAWAAYYSHYYQQPPGPVPGPAPAPAAPPAQGEPPQP...       5                1
3                                        AAAAADLANR       2                1
4                                  AAAAASAAGPGGLVAGK       2                1

         RT  Intensity   m_score  m_score_peptide_run_specific  ...  \
0  2277.710     4944.0  0.016517                       0.016516  ...
1  1609.310    14029.0  0.000583                       0.000583  ...
2  3275.670     2988.0  0.044867                       0.044866  ...
3   475.722     1992.0  0.012483                       0.012482  ...
4   978.743      978.0  0.004057                       0.004057  ...

   MS1Intensity  MS2Intensity  ModelParamSigmaRT ModelParamSigmaMz  \
0    611.732180     88.886325           0.289517          3.981082
1   2086.219858    201.586612           0.218611          3.505546
2    289.512315     87.778535           0.291877          2.737636
3    479.286636    134.605709           0.355179         11.511312
4    355.547847     31.256479           0.307980          7.227377

   ModelParamRT0  ModelParamMz0MS1  ModelParamMz0MS2  \
0      14.301614         19.422064         20.641688
1      15.515581         22.769187         21.015362
2      13.559004         21.539127         21.589122
3      17.247381         19.503196         16.965990
4      11.911672          9.497525         16.879286

                               ModelParamAmplitudes    InjectionName  \
0  929.679,4790.634790.63;198.689198.689,133.7331...  ProCan90-M03-01
1  14288.9,6270.76270.7;757.188757.188,649649,387...  ProCan90-M03-01
2  404.298,2301.282301.28;27.238927.2389,136.0631...  ProCan90-M03-01
3  545.168,1321.941321.94;221.035221.035,120.5912...  ProCan90-M03-01
4  1193.04,604.938604.938;17.968217.9682,66.87976...  ProCan90-M03-01

                                        PQM
0                    AAAAAAAAAAAAAAAGAGAGAK_3
1                          AAAAAAAAAAGRVGM_2
2  AAAAAADPNAAWAAYYSHYYQQPPGPVPGPAPAPAAPPAQGEPPQP...
3                                AAAAADLANR_2
4                          AAAAASAAGPGGLVAGK_2

[5 rows x 24 columns]
```

1.5.2 Compare intensities calculated by the `toffee.requant` C++ library

[5]:
```
def plot_intensities(requantified, cols, colname, divisor=None, expected_
↪intensity=None, ylim=None):
    log_func = np.log2
    log_func_name = '(log2)'
    colname = f'{colname} {log_func_name}'

    def load(df, col):
        df = df[['PQM', 'InjectionName', col]].set_index(['PQM', 'InjectionName']).
↪unstack('InjectionName')
```

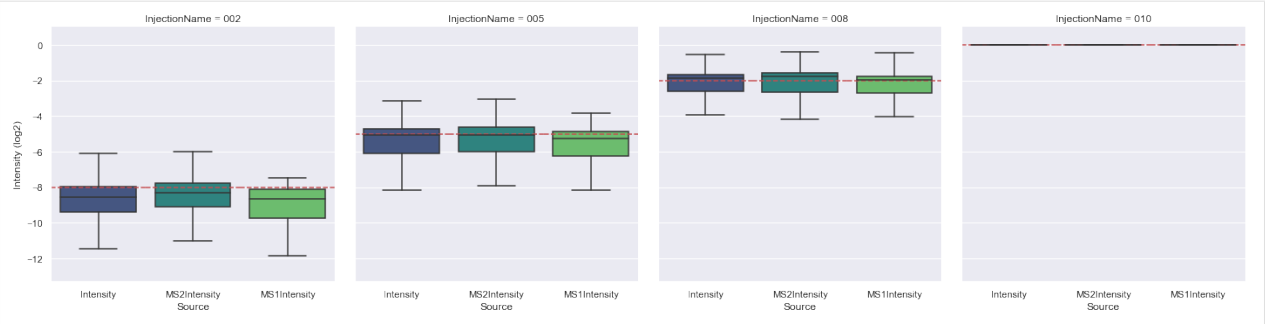
(continues on next page)

(continued from previous page)

```
        if divisor is not None:
            df = df.divide(df[(col, divisor)], axis=0)
        df = df.apply(log_func)
        return df.stack().reset_index(drop=False).rename(columns={col: colname})

    results = []
    for col in cols:
        df = load(requantified, col)
        df['Source'] = col
        results.append(df)
    results = pd.concat(results)

    x = 'InjectionName'
    col = 'Source'
    if expected_intensity is not None:
        x, col = col, x
    g = sns.catplot(
        kind='box',
        data=results,
        y=colname,
        x=x,
        col=col,
        palette='viridis',
        zorder=5,
        sharey=ylim is not None,
        showfliers=False,
    )

    if expected_intensity is not None:
        for ax, e in zip(g.axes.flatten(), expected_intensity):
            for i in range(len(results[x].unique())):
                ax.plot([i - 0.5, i + 0.5], log_func([e, e]), 'r--', zorder=1)
    if ylim is not None:
        ax.set_ylim(log_func(ylim))
    plt.show()
```

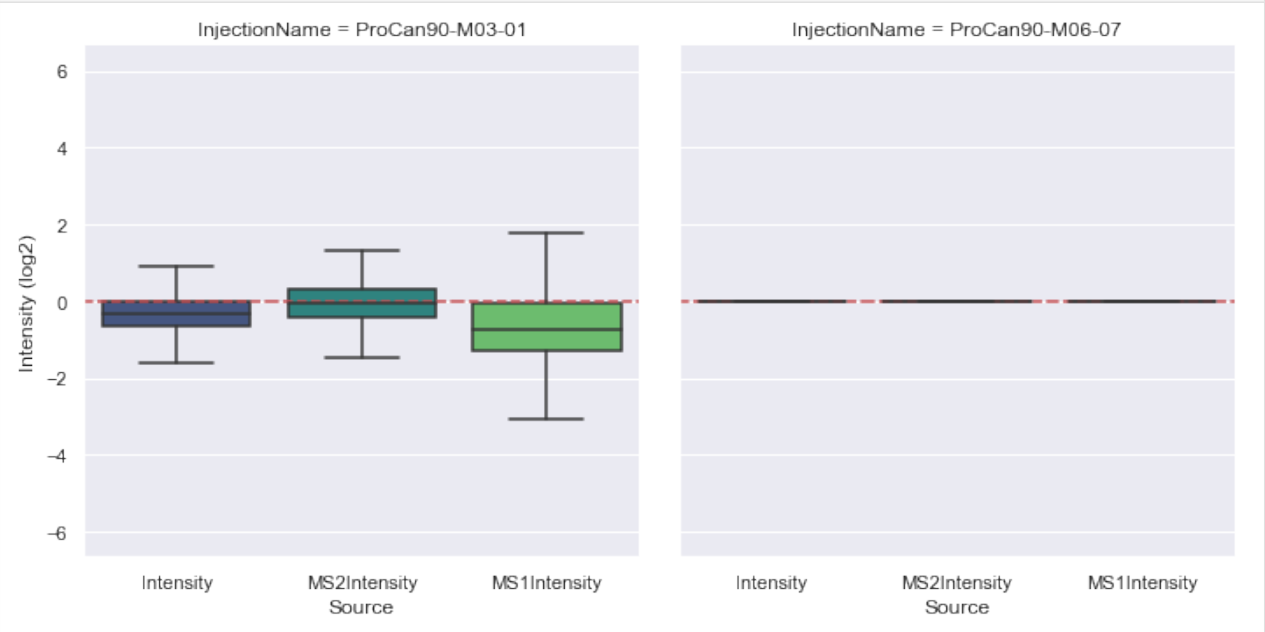
Swath Gold Standard data

[6]:
```
plot_intensities(
    sgs_requantified,
    ['Intensity', 'MS2Intensity', 'MS1Intensity'],
    'Intensity',
    expected_intensity=[2.0 ** (d - 10) for d in sgs_files],
    divisor='010',
    ylim=(1e-4, 2),
)
```



ProCan90 data

[7]:
```
plot_intensities(
    procan90_requantified,
    ['Intensity', 'MS2Intensity', 'MS1Intensity'],
    'Intensity',
    expected_intensity=(1.0, 1.0),
    divisor='ProCan90-M06-07',
    ylim=(0.01, 100),
)
```



1.5.3 Explore the raw data

High-scoring results

[8]:
```
def explore_raw_data(file, quantifiers, peptide_ids=None, plotter=None):
    q = quantifiers[file]
    if plotter is None:
        plotter = ToffeeFragmentsPlotter(q.toffee_filename)
    srl = q._add_ms2_windows(plotter.extractor.swath_run, q.srl)
```

(continues on next page)

(continued from previous page)

```
    if peptide_ids is None:
        peptide_ids = q.scores \
            .sort_values(['m_score', 'Intensity'], ascending=(True, False)) \
            .iloc[100:115] \
            .peptide_id \
            .tolist()

    all_srl = srl.loc[srl.peptide_id.isin(peptide_ids)].copy()
    for ms2_name in sorted(all_srl.MS2Name.unique()):
        print('*' * 80)
        ms2_srl = all_srl.loc[all_srl.MS2Name == ms2_name]
        for peptide_id in ms2_srl.peptide_id.unique():
            print('-' * 50)

            fragments = ms2_srl.loc[ms2_srl.peptide_id == peptide_id]
            pre = fragments['PrecursorMz'].tolist()[:1]
            pro = fragments['ProductMz'].tolist()[:6]

            peptide = q.scores.loc[q.scores.peptide_id == peptide_id, 'FullPeptideName
↪'].iloc[0]
            charge = q.scores.loc[q.scores.peptide_id == peptide_id, 'Charge'].iloc[0]
            rt = q.scores.loc[q.scores.peptide_id == peptide_id, 'RT'].iloc[0]
            rt_zoom = (rt - 300.0, rt + 300.0)

            print(f'ms2_name = \'{ms2_name}\'')
            print(f'peptide_id, peptide, charge, rt, pre = \'{peptide_id}\', \'
↪{peptide}\', {charge}, {rt}, {pre[0]}')
            print(f'pro = {pro}')

            output_fname = '/tmp/fig.png'
            fig = plotter.create_plotly_figure(
                precursor_mz_list=pre,
                product_mz_list=pro,
                psm_name='{} | extraction_width = {} {}'.format(
                    plotter.tof_fname.split('/')[-1].split('.')[0],
                    plotter.extractor.extraction_width,
                    'ppm' if plotter.extractor.use_ppm else 'px',
                ),
                number_of_isotopes=0,
                log_heatmap=True,
                normalise_per_fragment=False,
                retention_time_zoom=rt_zoom,
                aspect_scale=0.2,
                retention_time_line=rt,
                output_fname=output_fname,
            )
            display(Image(output_fname))
```

SGS data

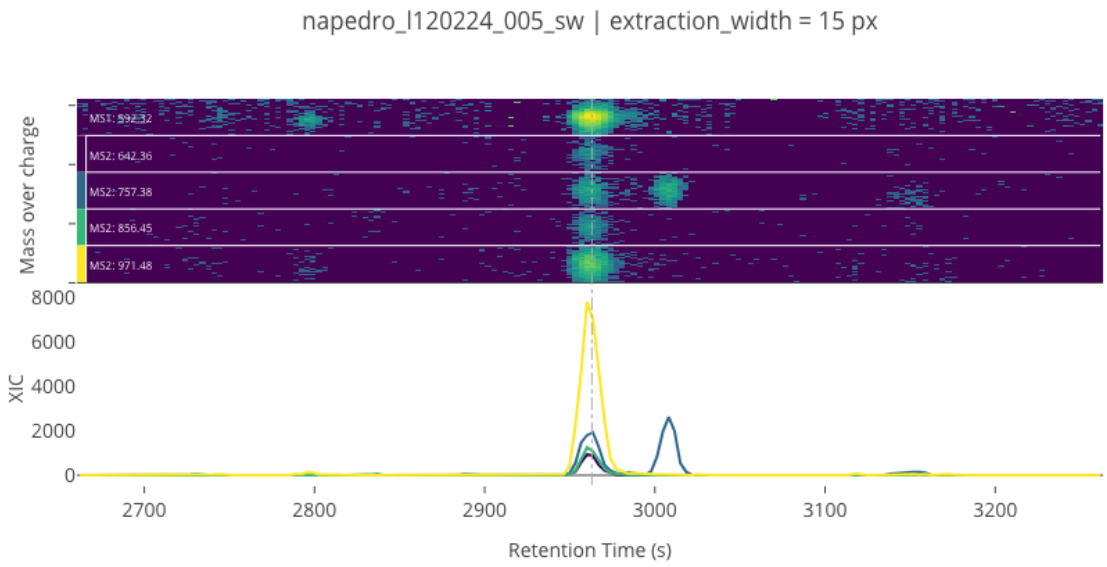
[9]:
```
sgs_file = 5
sgs_peptide_ids = [
    476,
```

(continues on next page)

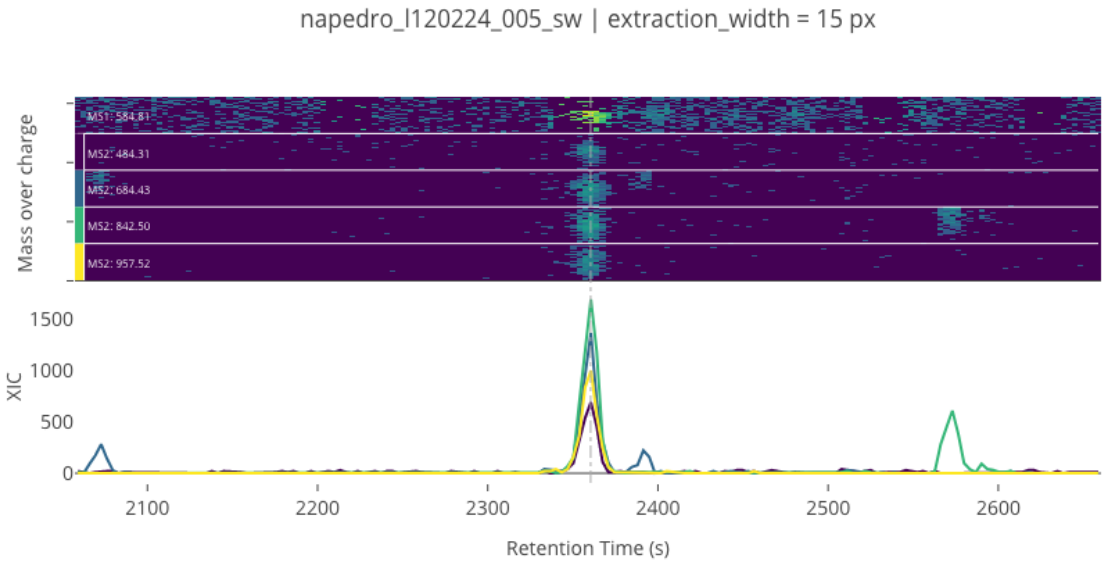
(continued from previous page)

```
    477,
    723,
    920,
]
explore_raw_data(
    file=sgs_file,
    quantifiers=sgs_quantifiers,
    peptide_ids=sgs_peptide_ids,
)
```

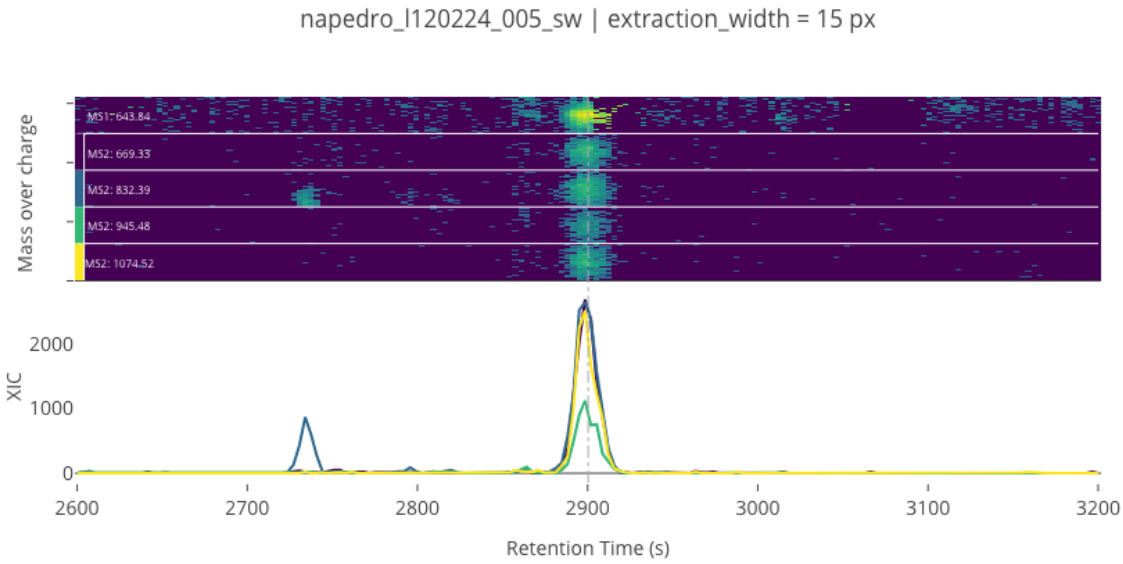
```
********************************************************************************
---------------------------------
ms2_name = 'ms2-008'
peptide_id, peptide, charge, rt, pre = '476', 'IVDVDLTSEGK(UniMod:259)', 2, 2962.99,␣
↪592.318
pro = [642.355, 757.382, 856.45, 971.477]
```



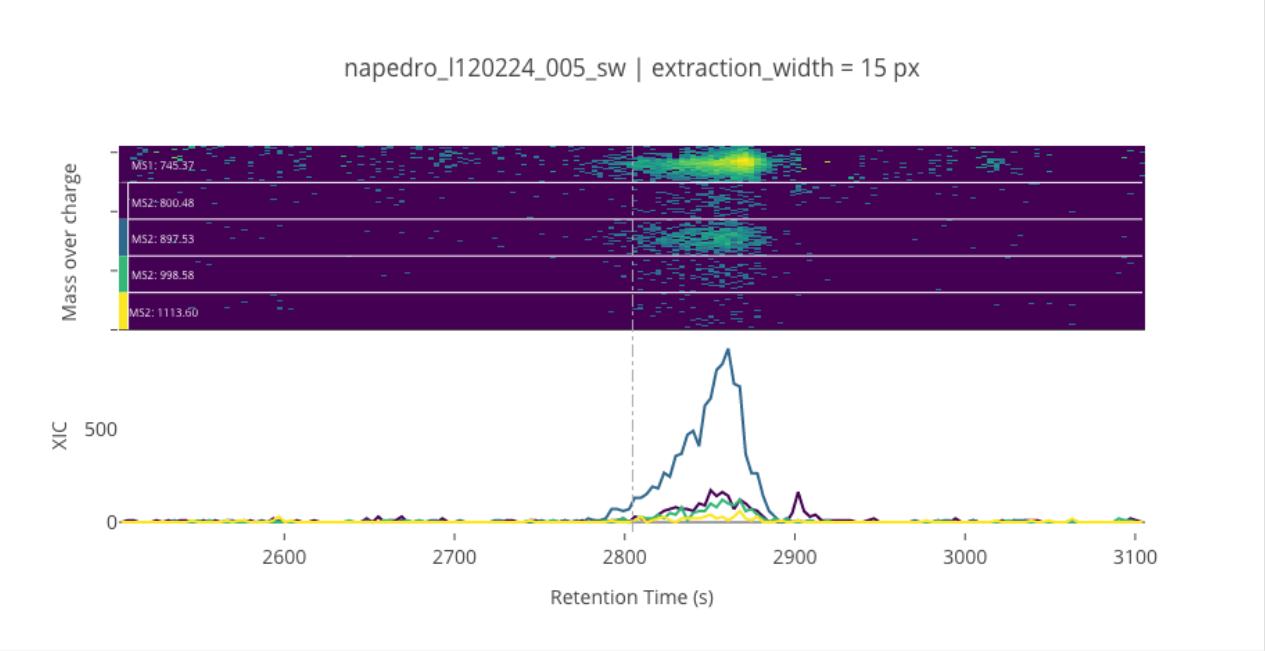

```
---------------------------------
ms2_name = 'ms2-008'
peptide_id, peptide, charge, rt, pre = '723', 'NPDGTVTVISR(UniMod:267)', 2, 2360.46,␣
↪584.813
pro = [484.312, 684.428, 842.497, 957.524]
```




```
********************************************************************************
---------------------------------
ms2_name = 'ms2-010'
peptide_id, peptide, charge, rt, pre = '477', 'IVEIYGPESSGK(UniMod:259)', 2, 2900.32,␣
↪643.84
pro = [669.329, 832.393, 945.477, 1074.52]
```



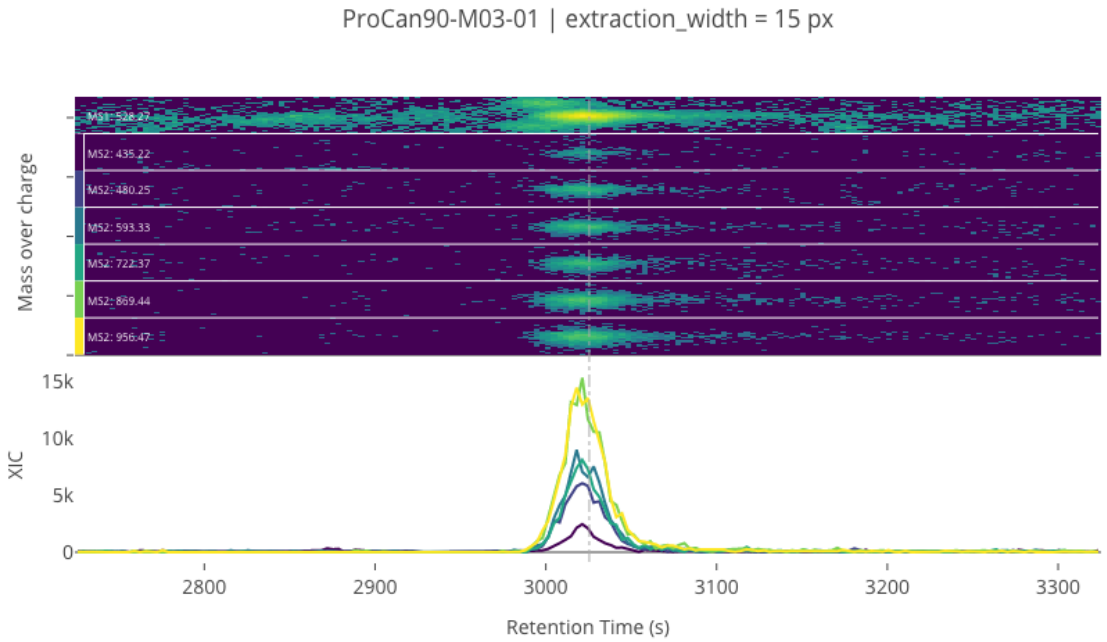

```
********************************************************************************
---------------------------------
ms2_name = 'ms2-014'
peptide_id, peptide, charge, rt, pre = '920', 'SDSSDTPPLPSPPGK(UniMod:259)', 2, 2804.
↪66, 745.367
pro = [800.476, 897.528, 998.576, 1113.6]
```



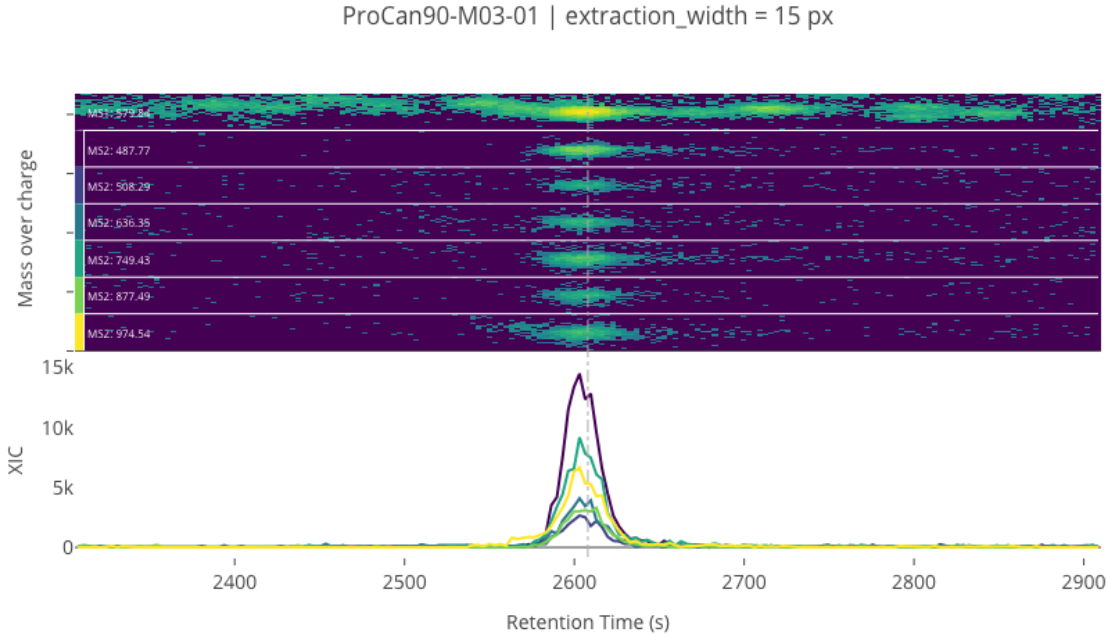
ProCan90 data

[10]:
```
procan90_file = '03-01'
procan90_peptide_ids = [  # these are found by examing the raw data
    197043,
    4114,
    114292,
    193918,
    33530,
#     29585,
#     34385,
#     199044,
#     109281,
#     16535,
#     31437,
#     113336,
#     192320,
]
explore_raw_data(
    file=procan90_file,
    quantifiers=procan90_quantifiers,
    peptide_ids=procan90_peptide_ids,
)
```

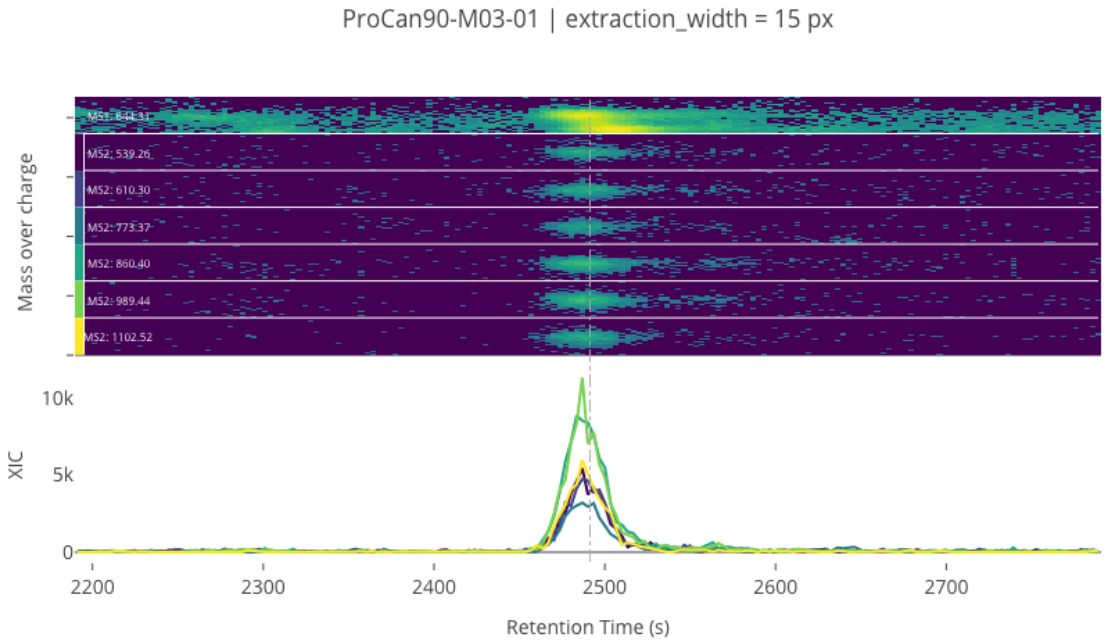
```
********************************************************************************
---------------------------------
ms2_name = 'ms2-024'
peptide_id, peptide, charge, rt, pre = '197043', 'VSFELFADK', 2, 3025.44, 528.27404785
pro = [435.22381592, 480.24526978, 593.3293457, 722.37194824, 869.44036865, 956.
↪47235107]
```



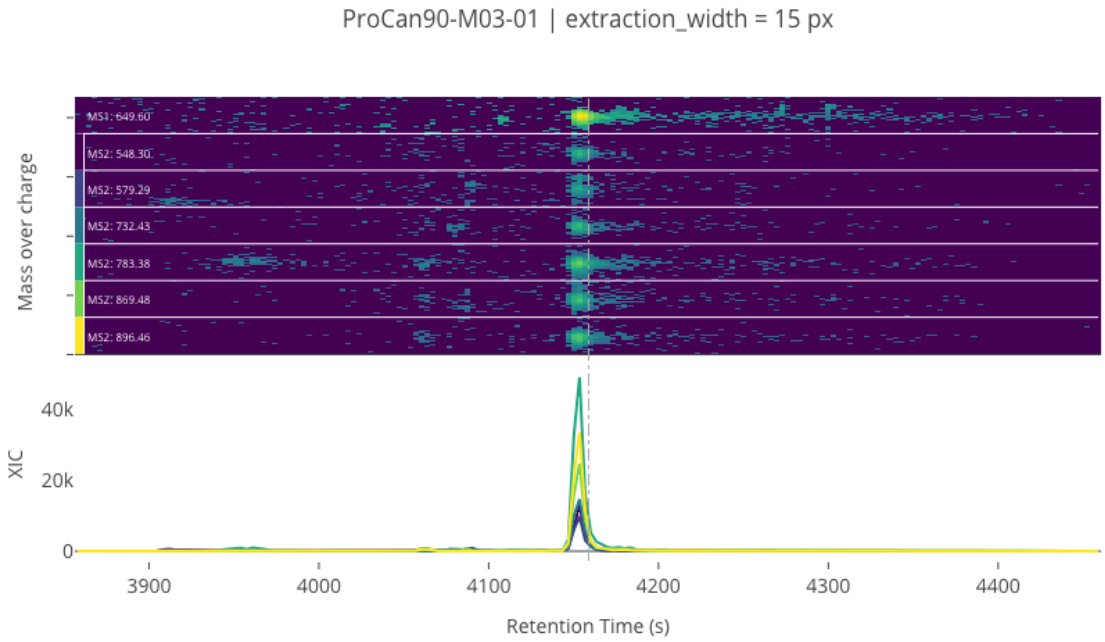

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-035'
peptide_id, peptide, charge, rt, pre = '4114', 'AIPQLQGYLR', 2, 2607.96, 579.83514404
pro = [487.77453613, 508.28781128, 636.34637451, 749.43041992, 877.48901367, 974.
↪54180908]
```



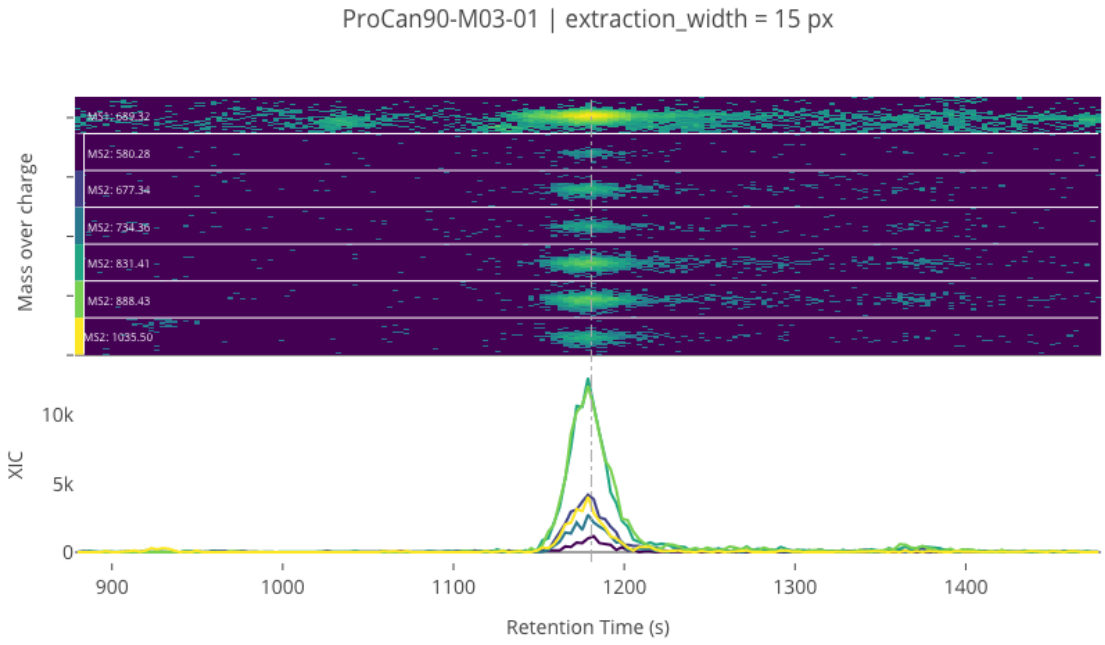
```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-048'
peptide_id, peptide, charge, rt, pre = '114292', 'NALESYAFNMK', 2, 2491.33, 644.
↪30554199
pro = [539.26464844, 610.30175781, 773.36505127, 860.39709473, 989.43969727, 1102.
↪5238037]
```




```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-049'
peptide_id, peptide, charge, rt, pre = '193918', 'VIHDNFGIVEGLMTTVHAITATQK', 4, 4158.
↪86, 649.59545898
pro = [548.30383301, 579.28851318, 732.42504883, 783.37841797, 869.48394775, 896.
↪46246338]
```




```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-055'
peptide_id, peptide, charge, rt, pre = '33530', 'GGGGNFGPGPGSNFR', 2, 1180.7, 689.
↪31835938
pro = [580.28381348, 677.33654785, 734.35803223, 831.4107666, 888.43225098, 1035.
↪5006104]
```



Outliers defined by the C++ library results

In fitting an analytic model to the raw data, we now have a new set of parameters that can be used for exploring *outliers*; where an outlier is defined in a manner similar to plotting the whiskers on a box plot. For certain parameters, if the value from the fit model falls outside a multiplier of the inter-quartile range, then it is classed as an outlier.

[11]:
```
def plot_params(requantified, cols, colname, whis=5.0):
    groups = requantified.groupby('InjectionName')

    n_injections = len(groups)
    colours = [plt.cm.viridis(i / n_injections) for i in range(n_injections)]

    scale = 5
    ncols = len(cols)
    fig, axes = plt.subplots(ncols=ncols, figsize=(scale * ncols, scale))
    for i, (ax, col) in enumerate(zip(axes, cols)):
        xlims = []
        for colour, (injection, df) in zip(colours, groups):

            mp_groups = df[col]
            mp_groups.plot(kind='kde', ax=ax, bw_method=0.3, c=colour,␣
↪label=injection)

            # plot outlier limits
            q1 = mp_groups.quantile(0.25)
            q3 = mp_groups.quantile(0.75)
            iqr = q3 - q1
            outlier_offset = whis * iqr
            lower = q1 - outlier_offset
            upper = q3 + outlier_offset
            ax.axvline(lower, ls='--', c=colour, alpha=0.5)
            ax.axvline(upper, ls='--', c=colour, alpha=0.5)

            xlim = list(ax.get_xlim())
            xlim[0] = max(xlim[0], lower - 2 * outlier_offset)
            xlim[1] = min(xlim[1], upper + 2 * outlier_offset)
            xlims.append(xlim)

        ax.set_xlim((min(x[0] for x in xlims), max(x[1] for x in xlims)))
        ax.set_title(col)
        if i == 0:
            ax.legend()
    plt.show()

def mark_outliers(requantified, col, whis=5.0):
    mp_groups = requantified[col].groupby(requantified.InjectionName)
    mean = mp_groups.mean()

    # calculate outliers as you would in a box plot
    q1 = mp_groups.quantile(0.25)
    q3 = mp_groups.quantile(0.75)
    iqr = q3 - q1
    lower = q1 - whis * iqr
    upper = q3 + whis * iqr

    delta_col = 'Delta' + col
    requantified[delta_col] = pd.np.NaN
```

(continues on next page)

(continued from previous page)

```
    outlier_col = 'Outlier' + col
    requantified[outlier_col] = pd.np.NaN
    for n in lower.index:
        mask = requantified.InjectionName == n

        requantified.loc[mask, delta_col] = (requantified.loc[mask, col] - mean.
↪loc[n]).abs()

        requantified.loc[mask, outlier_col] = (
            (requantified.loc[mask, col] < lower.loc[n]) |
            (requantified.loc[mask, col] > upper.loc[n])
        )

def get_outlier_peptides(requantified, injection_name, n=8, force=False, **kwargs):
    col_mz0 = 'ModelParamMz0MS2'
    col_rt0 = 'ModelParamRT0'
    col_sigma_rt = 'ModelParamSigmaRT'

    for col in [col_mz0, col_rt0, col_sigma_rt]:
        delta_col = 'Delta' + col
        outlier_col = 'Outlier' + col

        if delta_col not in requantified.columns or force:
            mark_outliers(requantified, col, **kwargs)

    outlier_col = 'PotentialOutlier'
    requantified[outlier_col] = (
        requantified['Outlier' + col_mz0] |
        requantified['Outlier' + col_rt0] |
        requantified['Outlier' + col_sigma_rt]
    )
    display(requantified[outlier_col].groupby([
        requantified['InjectionName'],
        requantified[outlier_col],
    ]).count())

    delta_col = 'PotentialOutlierDelta'
    requantified[delta_col] = (
        requantified['Delta' + col_mz0] *
        requantified['Delta' + col_rt0] *
        requantified['Delta' + col_sigma_rt]
    )

    outlier_peptide_ids = requantified.loc[
        (requantified.InjectionName == injection_name) &
        requantified[outlier_col]
    ].sort_values([delta_col, 'Intensity'], ascending=(False, False)).head(n)
    return outlier_peptide_ids.peptide_id.tolist()
```

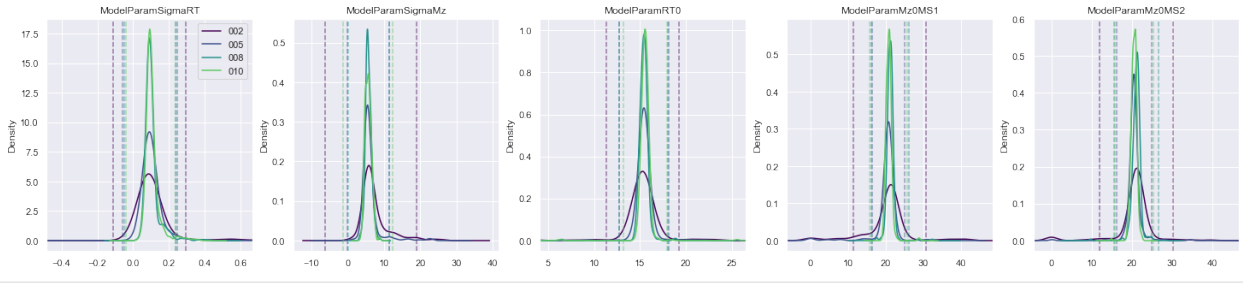
SGS data

[12]:
```
plot_params(
    sgs_requantified,
    ['ModelParamSigmaRT', 'ModelParamSigmaMz', 'ModelParamRT0', 'ModelParamMz0MS1',
↪'ModelParamMz0MS2'],
```

(continues on next page)

(continued from previous page)

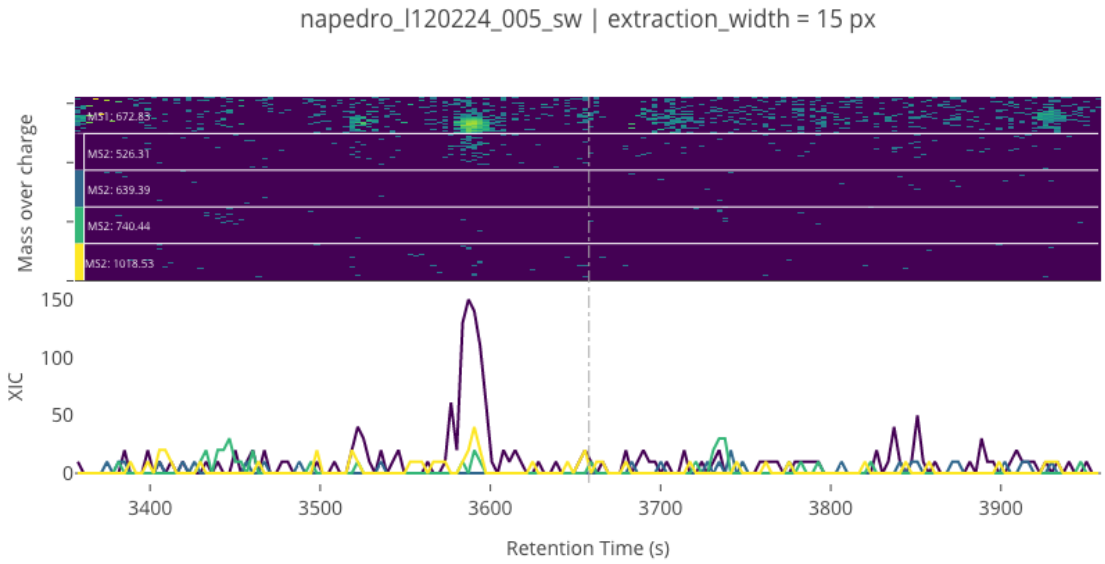
```
    'ModelParam',
)
sgs_outlier_peptide_ids = get_outlier_peptides(
    requantified=sgs_requantified,
    injection_name=f'{sgs_file:03d}',
    force=True,
    whis=5.0,
)
```



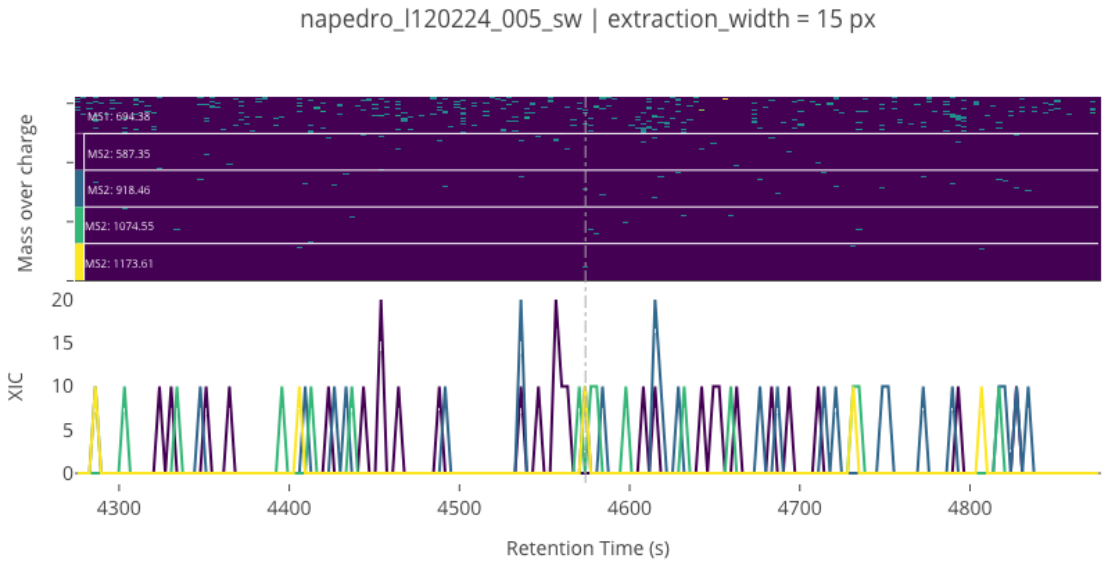

```
InjectionName  PotentialOutlier
002            False               161
               True                 22
005            False               258
               True                 18
008            False               331
               True                  8
010            False               331
               True                 12
Name: PotentialOutlier, dtype: int64
```

[13]:
```
explore_raw_data(
    file=sgs_file,
    quantifiers=sgs_quantifiers,
    peptide_ids=sgs_outlier_peptide_ids,
)
```

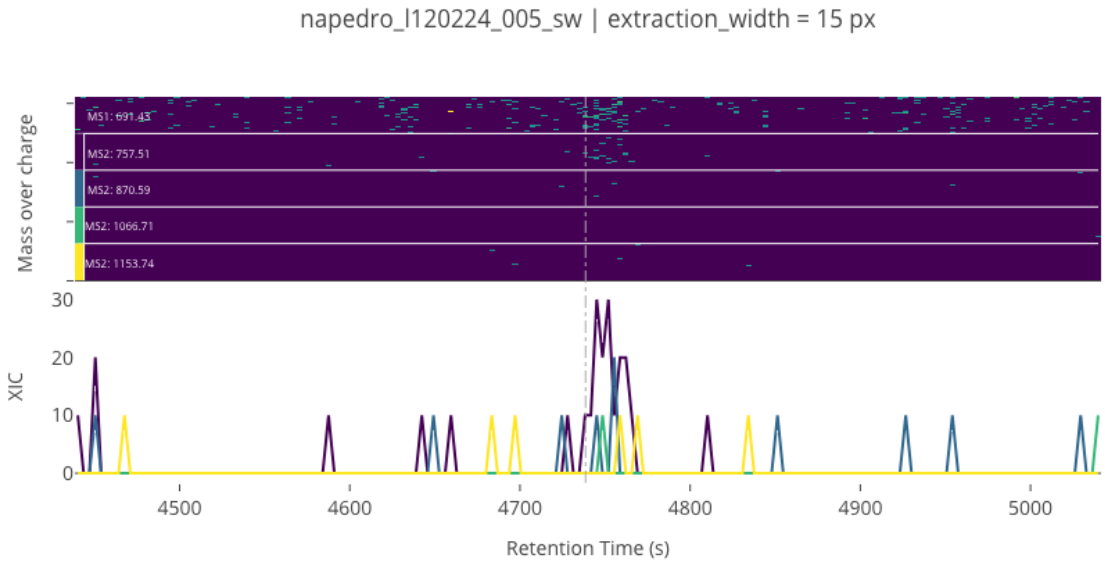
```
********************************************************************************
---------------------------------
ms2_name = 'ms2-011'
peptide_id, peptide, charge, rt, pre = '1158', 'YYDYTLSINGK(UniMod:259)', 2, 3657.93,␣
↪672.832
pro = [526.307, 639.392, 740.439, 1018.53]
```



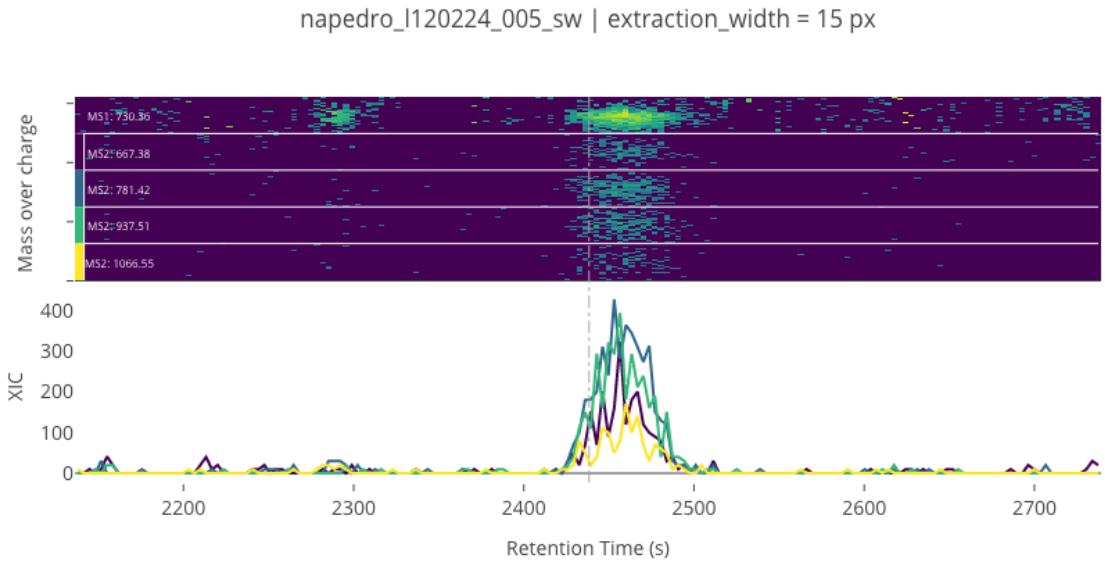

```
**********************************************************************************
----------------------------------
ms2_name = 'ms2-012'
peptide_id, peptide, charge, rt, pre = '985', 'TIVGVSEDFAALR(UniMod:267)', 2, 4574.05,
↪ 694.376
pro = [587.354, 918.455, 1074.55, 1173.61]
```



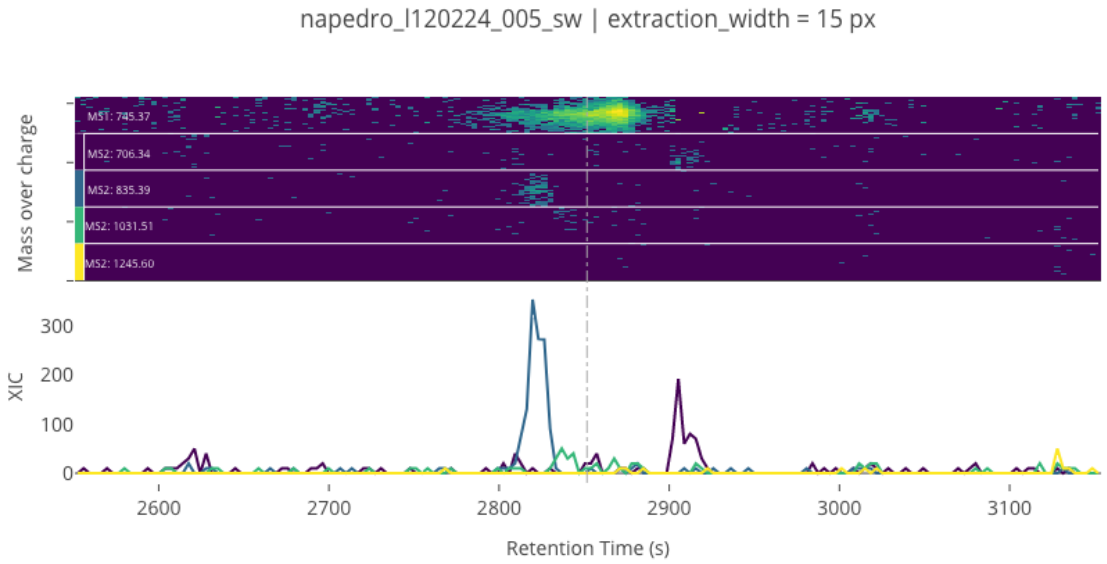

```
---------------------------------
ms2_name = 'ms2-012'
peptide_id, peptide, charge, rt, pre = '1051', 'VESPVLPPVLVPK(UniMod:259)', 2, 4738.
↪54, 691.431
pro = [757.506, 870.59, 1066.71, 1153.74]
```



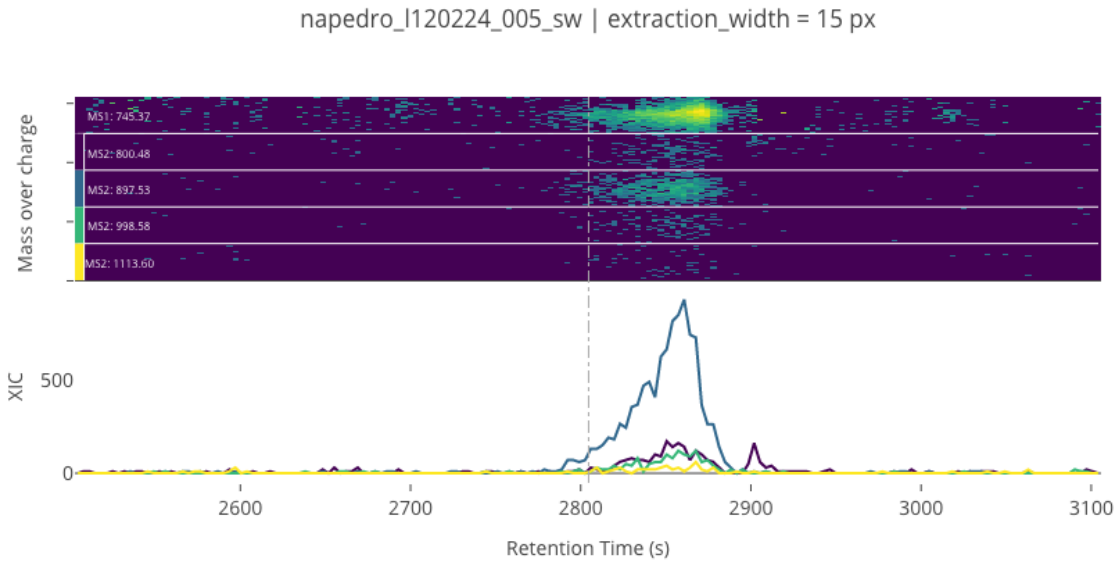

```
********************************************************************************
----------------------------------
ms2_name = 'ms2-014'
peptide_id, peptide, charge, rt, pre = '212', 'EYTEGVNGQPSIR(UniMod:267)', 2, 2438.35,
↪ 730.356
pro = [667.376, 781.419, 937.509, 1066.55]
```



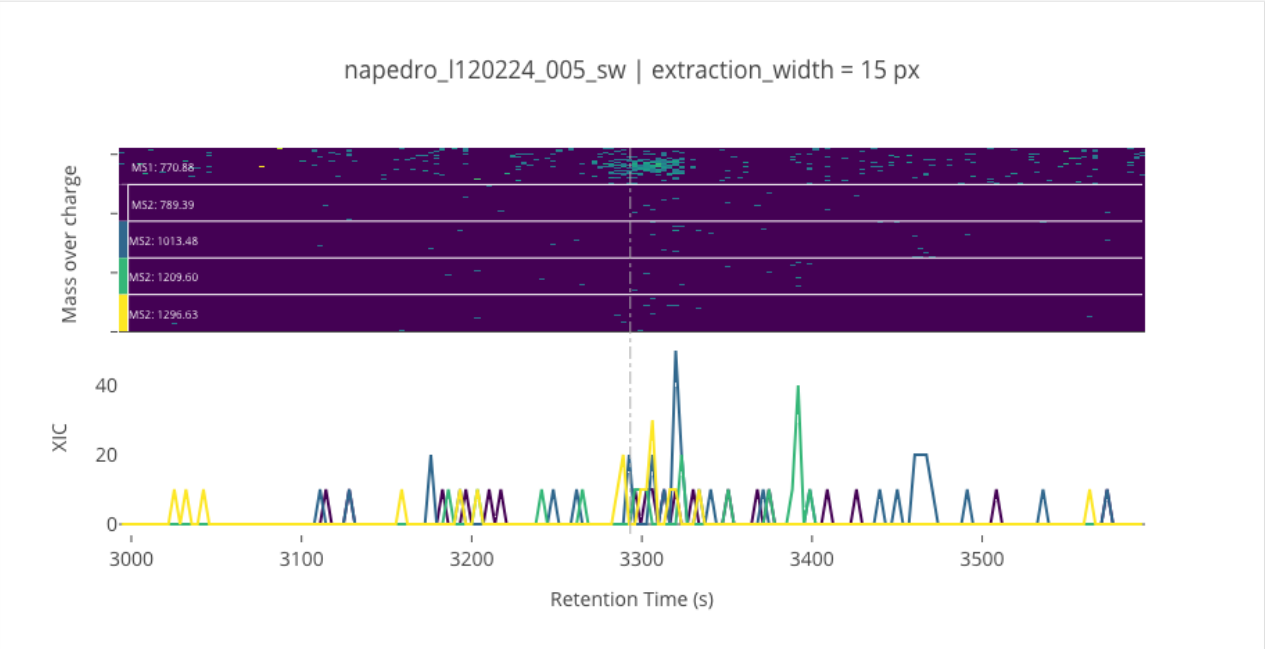

```
---------------------------------
ms2_name = 'ms2-014'
peptide_id, peptide, charge, rt, pre = '695', 'MIVDPVEPHGEMK(UniMod:259)', 2, 2851.66,
↪ 745.367
pro = [706.343, 835.386, 1031.51, 1245.6]
```



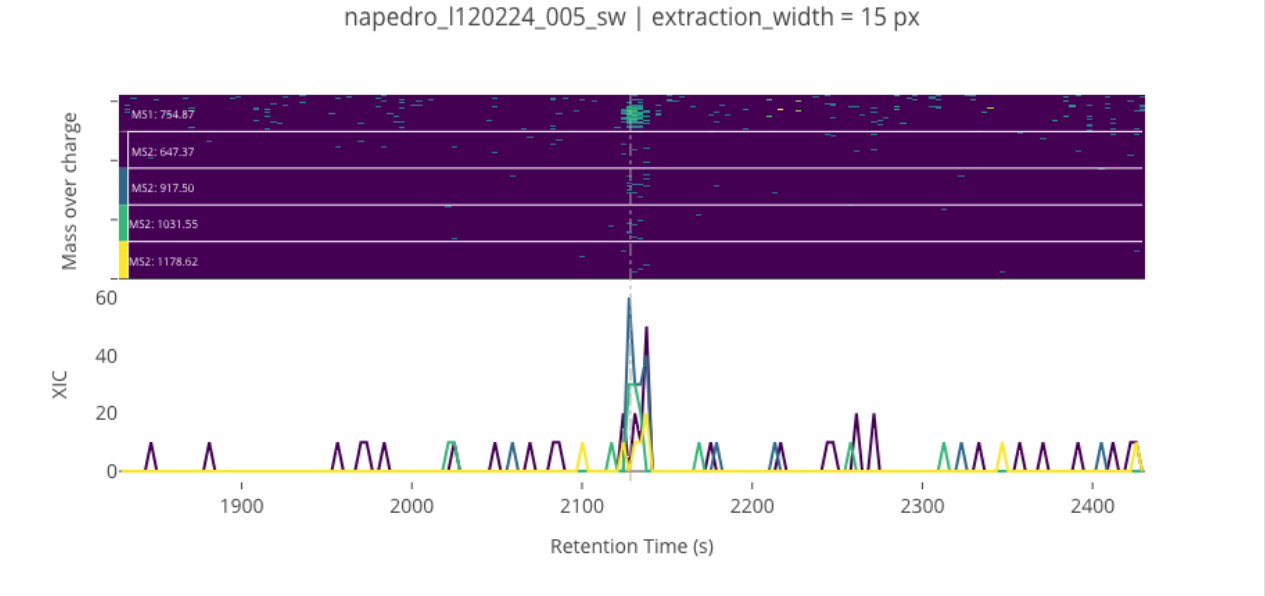

```
----------------------------------
ms2_name = 'ms2-014'
peptide_id, peptide, charge, rt, pre = '920', 'SDSSDTPPLPSPPGK(UniMod:259)', 2, 2804.
↪66, 745.367
pro = [800.476, 897.528, 998.576, 1113.6]
```




```
********************************************************************************
----------------------------------
ms2_name = 'ms2-015'
peptide_id, peptide, charge, rt, pre = '239', 'FPSPVSHADDLYGK(UniMod:259)', 2, 3293.
↪18, 770.88
pro = [789.387, 1013.48, 1209.6, 1296.63]
```




```
---------------------------------
ms2_name = 'ms2-015'
peptide_id, peptide, charge, rt, pre = '729', 'NTDFNGVNNIHQK(UniMod:259)', 2, 2128.47,
↪ 754.87
pro = [647.371, 917.504, 1031.55, 1178.62]
```



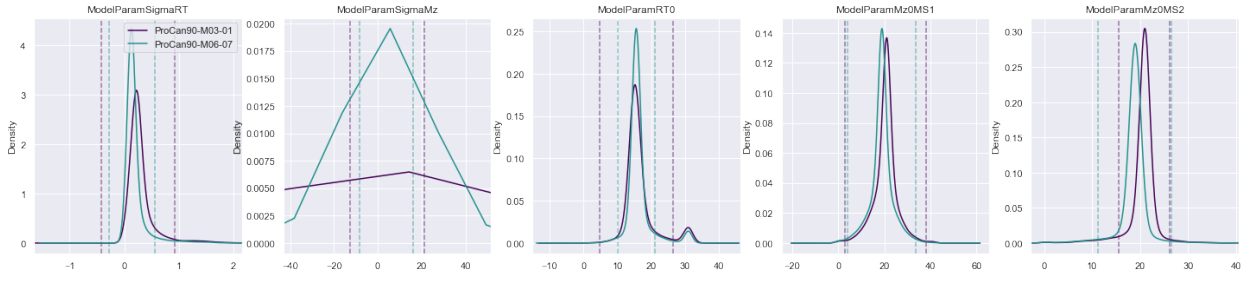
ProCan90 data

[14]:
```
plot_params(
    procan90_requantified,
    ['ModelParamSigmaRT', 'ModelParamSigmaMz', 'ModelParamRT0', 'ModelParamMz0MS1',
↪'ModelParamMz0MS2'],
```

(continues on next page)

(continued from previous page)

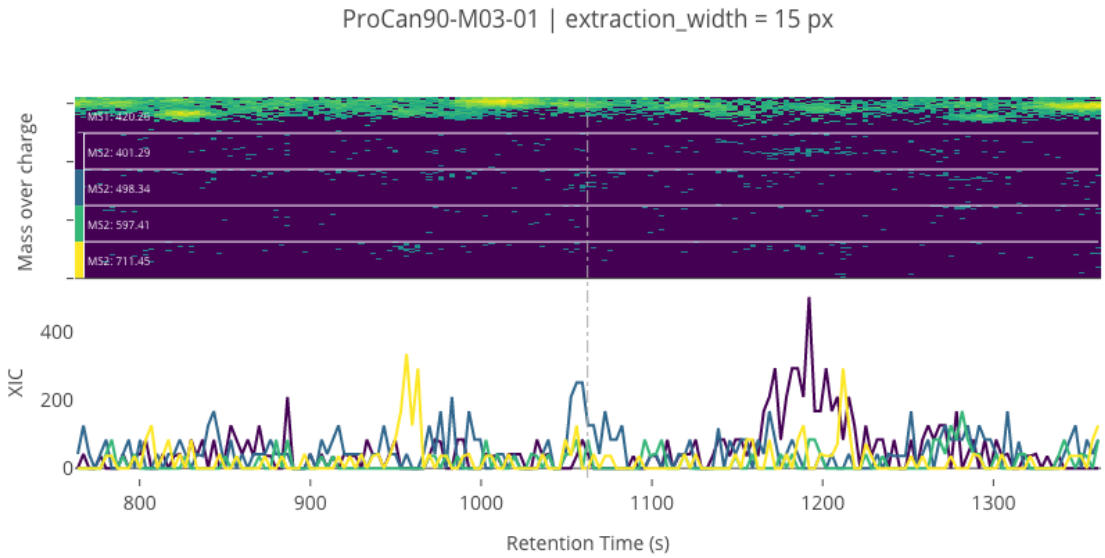
```
    'ModelParam',
)
procan90_outlier_peptide_ids = get_outlier_peptides(
    requantified=procan90_requantified,
    injection_name=f'ProCan90-M{procan90_file}',
    force=True,
)
```



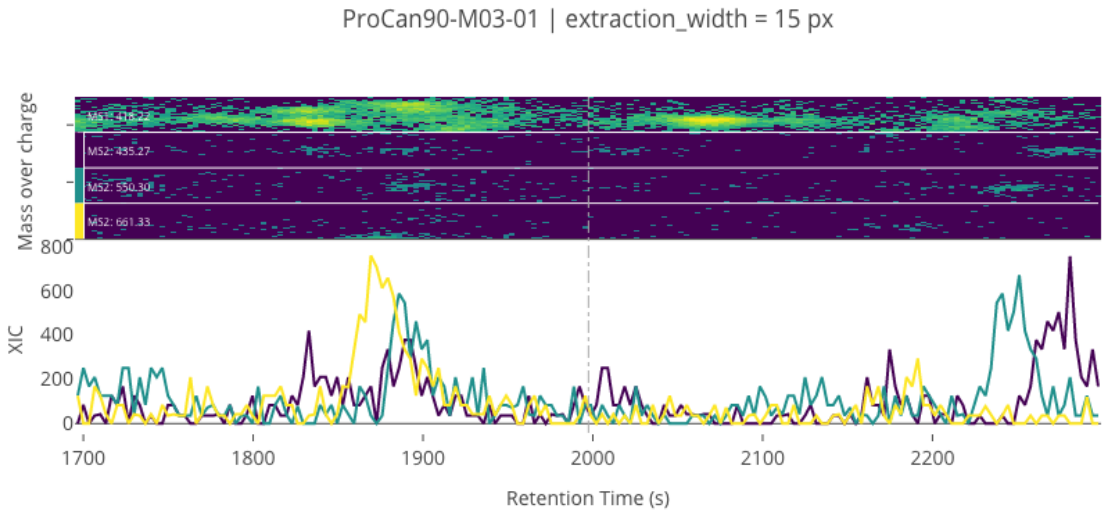

```
InjectionName    PotentialOutlier
ProCan90-M03-01  False               17022
                 True                2377
ProCan90-M06-07  False               23297
                 True                3138
Name: PotentialOutlier, dtype: int64
```

[15]:
```
explore_raw_data(
    file=procan90_file,
    quantifiers=procan90_quantifiers,
    peptide_ids=procan90_outlier_peptide_ids,
)
```

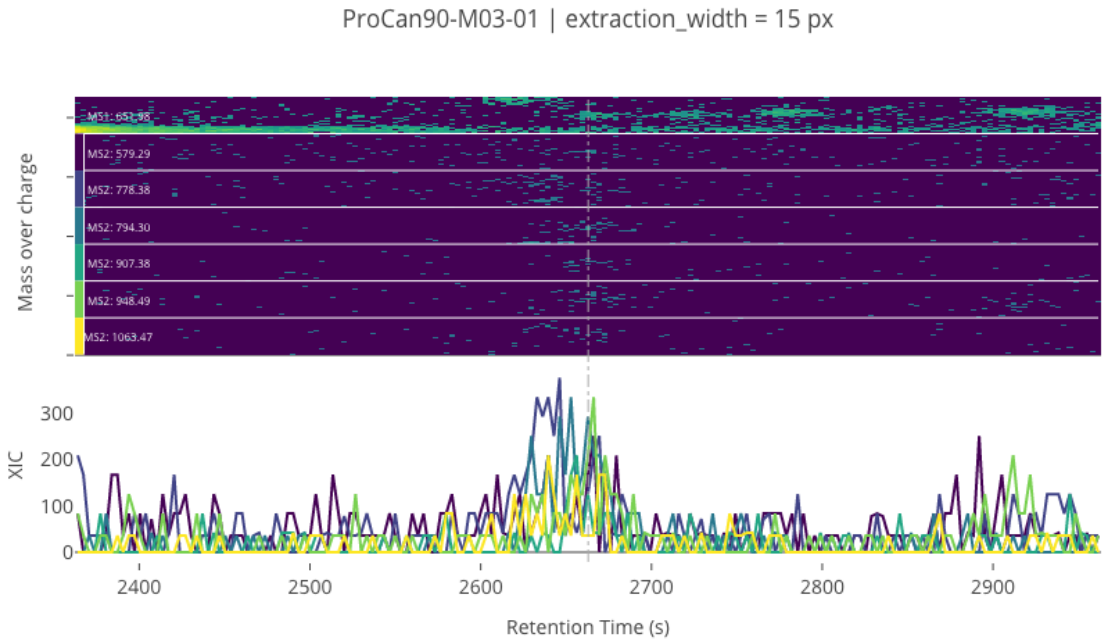
```
********************************************************************************
---------------------------------
ms2_name = 'ms2-004'
peptide_id, peptide, charge, rt, pre = '122878', 'QNVPIIR', 2, 1062.29, 420.2585144
pro = [401.28707886, 498.33984375, 597.40826416, 711.45117188]
```



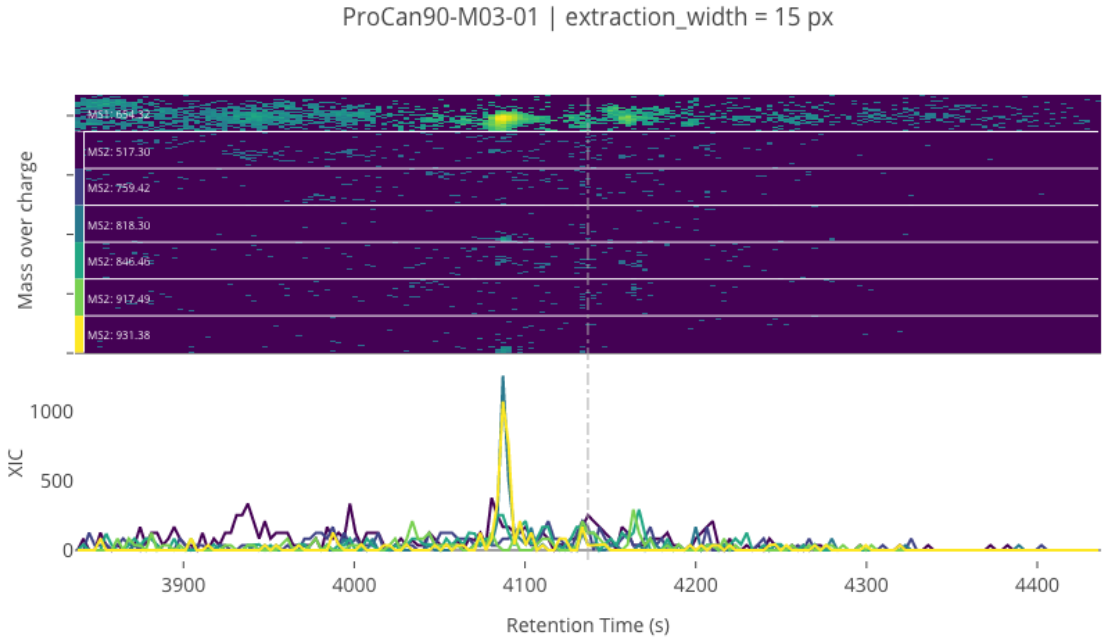

```
---------------------------------
ms2_name = 'ms2-004'
peptide_id, peptide, charge, rt, pre = '132921', 'REDFLR', 2, 1997.44, 418.22467041
pro = [435.27142334, 550.29840088, 661.3303833]
```



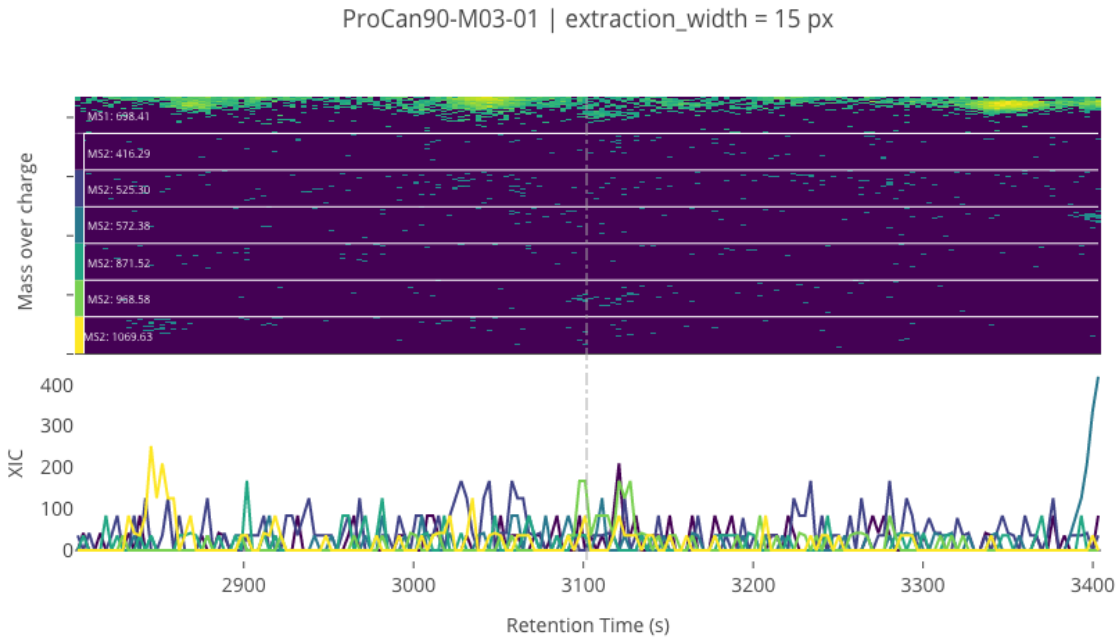

```
********************************************************************************
---------------------------------
ms2_name = 'ms2-049'
peptide_id, peptide, charge, rt, pre = '38727', 'HGEPEEDIVGLQAFQER', 3, 2662.94, 651.
↪98156738
pro = [579.28851318, 778.38421631, 794.29516602, 907.37921143, 948.48974609, 1063.
↪4691162]
```



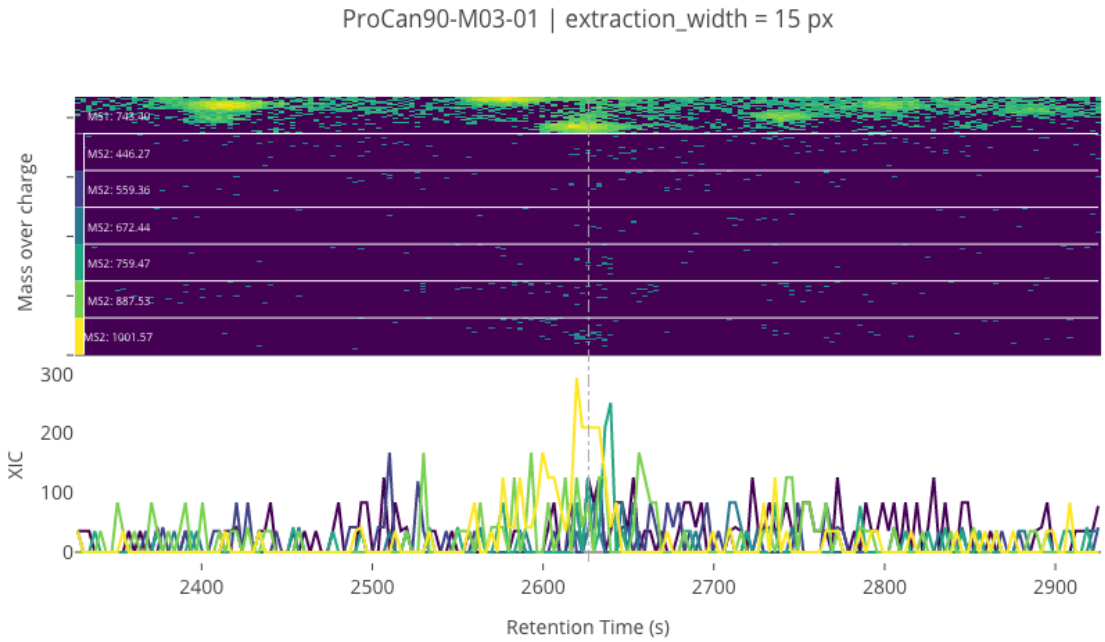

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-050'
peptide_id, peptide, charge, rt, pre = '114165', 'NADMQEELIASLEEQLK', 3, 4136.85, 654.
↪3225708
pro = [517.29803467, 759.42468262, 818.29852295, 846.45672607, 917.49383545, 931.
↪38256836]
```



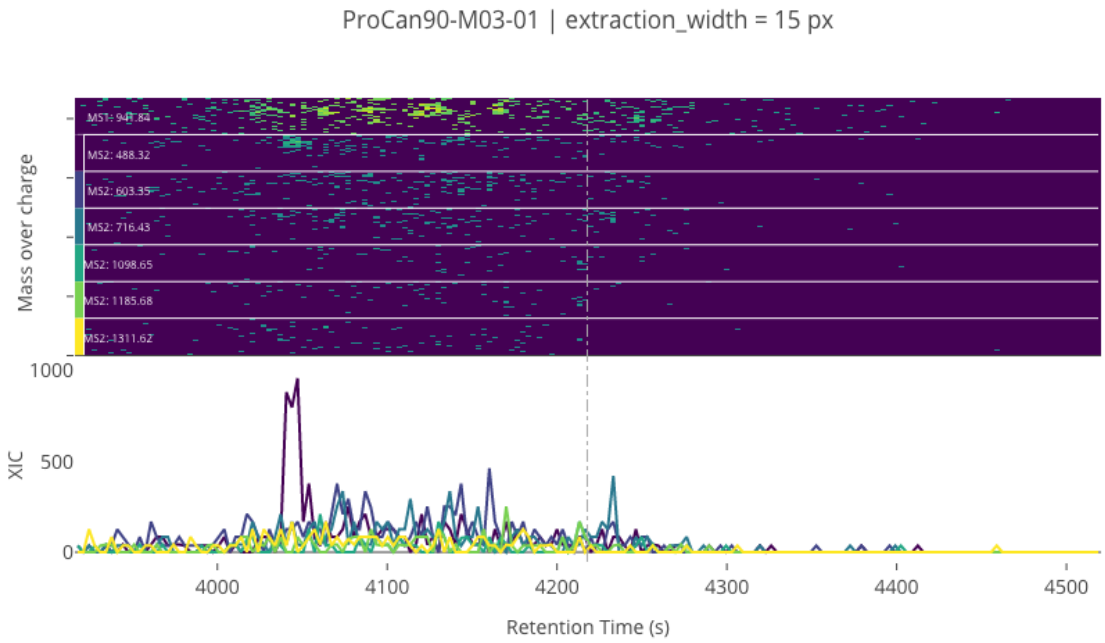
```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-057'
peptide_id, peptide, charge, rt, pre = '195660', 'VNITPAEVGVLVGK', 2, 3101.9, 698.
↪41394043
pro = [416.28674316, 525.30310059, 572.37664795, 871.52471924, 968.57751465, 1069.
↪6252441]
```



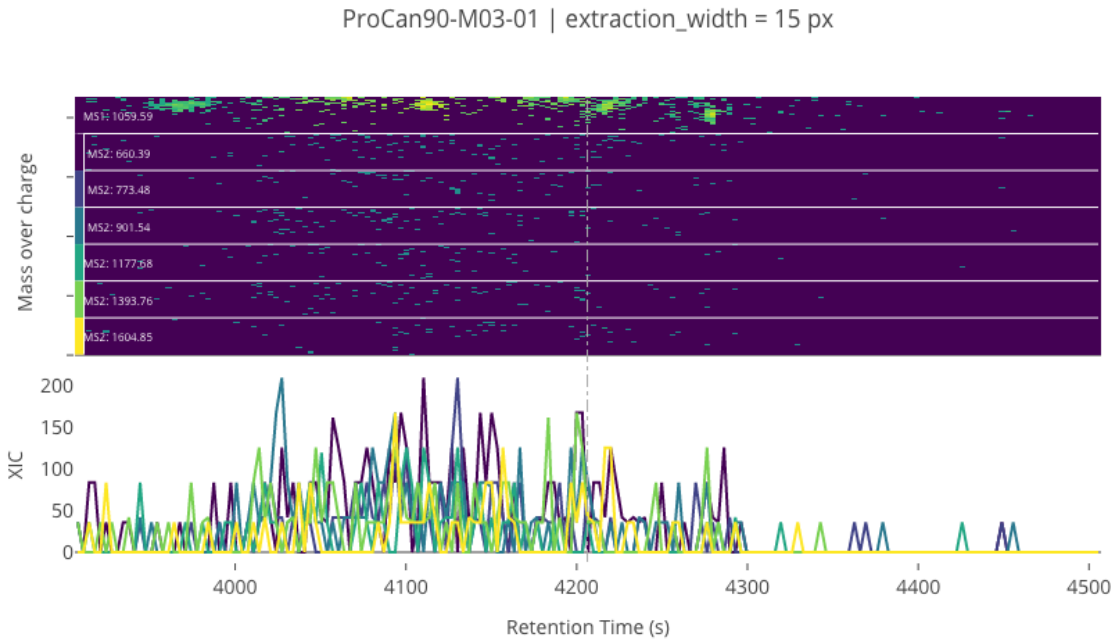

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-064'
peptide_id, peptide, charge, rt, pre = '18270', 'EAGNINQSLLTLGR', 2, 2626.65, 743.
↪40460205
pro = [446.27215576, 559.35620117, 672.44030762, 759.47229004, 887.53088379, 1001.
↪5737915]
```




```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-089'
peptide_id, peptide, charge, rt, pre = '192141', 'VEEASPGRPSSVDTLLSPTALIDSILR', 3,␣
↪4217.8, 941.84143066
pro = [488.31912231, 603.34606934, 716.43011475, 1098.6517334, 1185.6837158, 1311.
↪6175537]
```



```
********************************************************************************
---------------------------------
ms2_name = 'ms2-096'
peptide_id, peptide, charge, rt, pre = '179998', 'SLLSIPNTDYIQLLSEIAK', 2, 4206.23,␣
↪1059.5882568
pro = [660.39263916, 773.47674561, 901.53533936, 1177.6827393, 1393.7573242, 1604.
↪8530273]
```



1.5.4 Building a python model

[16]:
```
import scipy
import scipy.optimize
import scipy.special


class Requant():
    def __init__(
        self,
        tof_fname,
        rt_pixel_half_width=15,
        mz_pixel_half_width=20,
        num_ms1_isotopes=1,
        debug_plots=False,
    ):
        self.swath_run = toffee.SwathRun(tof_fname)
        self.ms1_swath_map = self.swath_run.loadSwathMap(toffee.ToffeeWriter.MS1_NAME)
        self.rt_pixel_half_width = rt_pixel_half_width
        self.mz_pixel_half_width = mz_pixel_half_width
        self.num_ms1_isotopes = num_ms1_isotopes
        self.debug_plots = debug_plots
```

(continues on next page)

(continued from previous page)

```
    def run(self, ms2_name, peak_groups):
        ms2_swath_map = self.swath_run.loadSwathMap(ms2_name)
        results = []
        for peak_group in peak_groups:
            assert isinstance(peak_group, requant.RequantPeakGroup)
            raw_data = self._extract_raw_data(ms2_swath_map, peak_group)
            if raw_data is None:
                continue
            ms1_intensity, ms2_intensity = self._fit_intensities(*raw_data)
            results.append((peak_group.modifiedSequence(), peak_group.charge(), ms1_
↪intensity, ms2_intensity))
        return results

    def _extract_raw_data(self, ms2_swath_map, peak_group):
        rt_range = toffee.RetentionTimeRangeWithPixelHalfWidth(
            peak_group.retentionTime(),
            self.rt_pixel_half_width,
        )
        expected_shape = (2 * self.rt_pixel_half_width + 1, 2 * self.mz_pixel_half_
↪width + 1)
        raw_data = []
        normalisers = []
        delta_rt = []
        delta_mz = []

        def extract(sm, mz):
            mz_range = toffee.MassOverChargeRangeWithPixelHalfWidth(mz, self.mz_pixel_
↪half_width)
            chrom = sm.filteredExtractedIonChromatogram(mz_range, rt_range)
            if chrom.intensities.shape == expected_shape:
                r = chrom.intensities.astype(np.float)
                n = max(1, r.max())
                raw_data.append(r / n)
                normalisers.append(n)
                delta_rt.append(chrom.retentionTime[1] - chrom.retentionTime[0])
                delta_mz.append(np.diff(chrom.massOverCharge).mean())
            else:
                print(mz, chrom.intensities.shape, '!=', expected_shape)

        # MS1 data
        for i in range(self.num_ms1_isotopes + 1):
            isotope_mass_offset = 1.0033548
            mz = peak_group.massOverCharge() + i * isotope_mass_offset / peak_group.
↪charge()
            extract(self.ms1_swath_map, mz)
        if len(raw_data) == 0:
            return None

        # MS2 data
        for fragment_mz in peak_group.fragmentMassOverCharges():
            extract(ms2_swath_map, fragment_mz)

        return delta_rt, delta_mz, raw_data, normalisers

    def _fit_intensities(self, delta_rt, delta_mz, raw_data, normalisers):
        n_ms1 = self.num_ms1_isotopes + 1
```

(continues on next page)

(continued from previous page)

```
        n_frag = len(raw_data)
        n_rt = self.rt_pixel_half_width * 2 + 1
        n_mz = self.mz_pixel_half_width * 2 + 1
        assert n_mz > n_rt
        n_slice = n_rt * n_mz

        rt = np.arange(n_rt) + 0.5
        mz = np.arange(n_mz) + 0.5
        rt, mz = np.meshgrid(rt, mz, indexing='ij')

        raw_data_vec = np.zeros((n_frag * n_rt * n_mz,))
        rt_vec = np.zeros_like(raw_data_vec)
        mz_vec = np.zeros_like(raw_data_vec)
        for i_frag, raw in enumerate(raw_data):
            idx = slice(i_frag * n_slice, (i_frag + 1) * n_slice)
            raw_data_vec[idx] = raw.ravel()
            rt_vec[idx] = rt.ravel()
            mz_vec[idx] = mz.ravel()
            for i_rt in range(n_rt):
                for i_mz in range(n_mz):
                    idx = i_frag * n_slice + i_rt * n_mz + i_mz
                    assert raw_data_vec[idx] == raw[i_rt, i_mz]
                    assert rt_vec[idx] == rt[i_rt, i_mz]
                    assert mz_vec[idx] == mz[i_rt, i_mz]
        # pre-allocate for fast calculations
        fit_data_vec = np.zeros_like(raw_data_vec)
        sigma_rt_vec = np.zeros_like(raw_data_vec)
        sigma_mz_vec = np.zeros_like(raw_data_vec)
        rt0_vec = np.zeros_like(raw_data_vec)
        mz0_vec = np.zeros_like(raw_data_vec)
        amplitude_vec = np.zeros_like(raw_data_vec)
        chem_noise_amplitude_vec = np.zeros_like(raw_data_vec)

        radius_vec = np.zeros_like(raw_data_vec)

        def sigmoid(n, x):
            # n * scipy.special.expit(x)
            return 0.5 * n * (1 + np.tanh(x))

        def guassian_2d(sigma_rt, sigma_mz, rt0, mz0_ms1, mz0_ms2, amplitude, chem_
↪noise_amplitude):

            # set up vectors, enforce positivity and ranges
            rt0_vec[:] = sigmoid(n_rt, rt0)
            sigma_rt_vec[:] = sigma_rt ** 2
            sigma_mz_vec[:] = sigma_mz ** 2
            for i_frag in range(n_frag):
                idx = slice(i_frag * n_slice, (i_frag + 1) * n_slice)
                amplitude_vec[idx] = amplitude[i_frag] ** 2
                if i_frag < n_ms1:
                    mz0_vec[idx] = sigmoid(n_mz, mz0_ms1)
                    chem_noise_amplitude_vec[idx] = chem_noise_amplitude[i_frag] ** 2
                else:
                    mz0_vec[idx] = sigmoid(n_mz, mz0_ms2)

            # calculate the fit model
            exp_rt = -0.5 * ((np.log(rt_vec) - np.log(rt0_vec)) / sigma_rt_vec) ** 2
```

(continues on next page)

(continued from previous page)

```
            denom_rt = np.sqrt(2 * np.pi) * rt_vec * sigma_rt_vec

            exp_mz = -0.5 * ((mz_vec - mz0_vec) / sigma_mz_vec) ** 2

            signal_vec = amplitude_vec * np.exp(exp_rt + exp_mz) / denom_rt
            chem_noise_vec = chem_noise_amplitude_vec * np.exp(exp_mz)

            return signal_vec + chem_noise_vec

        def all_guassian_2d(sigma_rt, sigma_mz, rt0, mz0_ms1, mz0_ms2, *amplitudes_
↪and_noise):
            assert len(amplitudes_and_noise) == n_frag + n_ms1
            fit_data_vec[:] = guassian_2d(
                sigma_rt,
                sigma_mz,
                rt0,
                mz0_ms1,
                mz0_ms2,
                amplitudes_and_noise[:n_frag],
                amplitudes_and_noise[n_frag:],
            )

        def residual(params):
            all_guassian_2d(*params)
            return fit_data_vec - raw_data_vec

        # sigma_x, sigma_y, x0, y0_ms1, y0_ms2
        guess = [0.5, 2.0, 0.0, 0.0, 0.0]
        n = len(guess)
        # add amplitude vector
        guess = guess + [1.0] * n_frag
        # add chemical noise vector
        guess = guess + [0.1] * n_ms1

        result = scipy.optimize.least_squares(residual, guess, method='lm')

        ms1_intensity, ms2_intensity = np.NaN, np.NaN
        if result.success:
            # print(result)
            # print('')
            # print('n_frag', n_frag)
            # print('sigma_x, sigma_y, x0', result.x[:n])
            # print('y0', result.x[n:n + n_frag])
            # print('amplitude', result.x[n + n_frag:n + 2 * n_frag])
            # print('noise', result.x[n + 2 * n_frag:])

            if self.debug_plots:
                self._make_debug_plots(raw_data_vec, fit_data_vec)

            # calculate final fit
            params = result.x.copy()
            params[-n_ms1:] = 0.0  # set checmical noise to zero
            all_guassian_2d(*params)

            area_under_curve = []
            raw_area_under_curve = []
            for i_frag in range(n_frag):
```

(continues on next page)

(continued from previous page)

```
                idx = slice(i_frag * n_slice, (i_frag + 1) * n_slice)
                a = fit_data_vec[idx].sum()
                a *= normalisers[i_frag]
                a *= delta_rt[i_frag]
                a *= delta_mz[i_frag]
                area_under_curve.append(a)
                a = raw_data_vec[idx].sum()
                a *= normalisers[i_frag]
                a *= delta_rt[i_frag]
                a *= delta_mz[i_frag]
                raw_area_under_curve.append(a)

            # integrate intensity -- subtracting noise
            ms1_intensity = sum(area_under_curve[:n_ms1])
            ms2_intensity = sum(area_under_curve[n_ms1:])
            raw_ms1_intensity = sum(raw_area_under_curve[:n_ms1])
            raw_ms2_intensity = sum(raw_area_under_curve[n_ms1:])
            print('Raw MS1 & MS2:', raw_ms1_intensity, raw_ms2_intensity)
            print('Fit MS1 & MS2:', ms1_intensity, ms2_intensity)

        return ms1_intensity, ms2_intensity

    def _make_debug_plots(self, raw_data_vec, fit_data_vec):
        import matplotlib.pyplot as plt

        n_rt = self.rt_pixel_half_width * 2 + 1
        n_mz = self.mz_pixel_half_width * 2 + 1
        assert n_mz > n_rt
        n_slice = n_rt * n_mz
        n_frag = raw_data_vec.shape[0] // n_slice

        raw_data = []
        fit_data = []
        for i_frag in range(n_frag):
            idx = slice(i_frag * n_slice, (i_frag + 1) * n_slice)
            raw_data.append(raw_data_vec[idx].reshape(n_rt, n_mz))
            fit_data.append(fit_data_vec[idx].reshape(n_rt, n_mz))

        ncols = len(raw_data)
        nrows = 4
        scale = 2
        figsize = (scale * ncols * 1.5, scale * nrows)
        cmap = plt.cm.viridis
        for take_transpose in [False, True]:
            fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=figsize)
            for raw, fit, (ax1, ax2, ax3, ax4) in zip(raw_data, fit_data, axes.T):
                x = np.arange(n_rt)
                y = np.arange(n_mz)
                if take_transpose:
                    raw = raw.T
                    fit = fit.T
                    x, y = y, x
                diff = raw - fit
                vmin, vmax = 0, max(raw.max(), fit.max())
                levels = np.linspace(vmin, vmax, num=30)

                ax1.contourf(x, y, raw.T, vmin=vmin, vmax=vmax, levels=levels,␣
↪cmap=cmap)
```

(continues on next page)

(continued from previous page)

```
                ax2.contourf(x, y, fit.T, vmin=vmin, vmax=vmax, levels=levels,␣
↪cmap=cmap)
                ax3.contourf(x, y, np.abs(diff).T, vmin=vmin, vmax=vmax,␣
↪levels=levels, cmap=cmap)
                for ax in (ax1, ax2, ax3):
                    ax.get_xaxis().set_visible(False)
                    ax.get_yaxis().set_visible(False)

                ax4.plot(x, np.sum(raw, axis=1))
                ax4.plot(x, np.sum(fit, axis=1))
                ax4.plot(x, np.sum(diff, axis=1))

            x_label = 'RT' if not take_transpose else 'm/z'
            y_label = 'RT' if take_transpose else 'm/z'
            axes[0, 0].set_ylabel(f'Raw data ({y_label})')
            axes[1, 0].set_ylabel(f'Fit data ({y_label})')
            axes[2, 0].set_ylabel(f'Difference ({y_label})')
            axes[3, 0].set_ylabel(f'Integrated over {y_label}')
            for ax in axes[-1, :]:
                ax.set_xlabel(x_label)

            fig.tight_layout()
            plt.show()

def run_python_requant(file, quantifiers, peptide_ids=None):
    q = quantifiers[file]
    swath_run = toffee.SwathRun(q.toffee_filename)
    srl = q._add_ms2_windows(swath_run, q.srl)

    if peptide_ids is None:
        peptide_ids = q.scores \
            .sort_values(['m_score', 'Intensity'], ascending=(True, False)) \
            .iloc[100:115] \
            .peptide_id \
            .tolist()

    all_srl = srl.loc[srl.peptide_id.isin(peptide_ids)].copy()
    operator = Requant(q.toffee_filename, debug_plots=True)

    for ms2_name in sorted(all_srl.MS2Name.unique()):
        print('*' * 80)
        ms2_srl = all_srl.loc[all_srl.MS2Name == ms2_name]
        for peptide_id in ms2_srl.peptide_id.unique():
            print('-' * 50)

            fragments = ms2_srl.loc[ms2_srl.peptide_id == peptide_id]
            pre = fragments['PrecursorMz'].tolist()[:1]
            pro = fragments['ProductMz'].tolist()[:6]

            peptide = q.scores.loc[q.scores.peptide_id == peptide_id, 'FullPeptideName
↪'].iloc[0]
            charge = q.scores.loc[q.scores.peptide_id == peptide_id, 'Charge'].iloc[0]
            rank = q.scores.loc[q.scores.peptide_id == peptide_id, 'peak_group_rank'].
↪iloc[0]
            rt = q.scores.loc[q.scores.peptide_id == peptide_id, 'RT'].iloc[0]

            print(f'ms2_name = \'{ms2_name}\'')
```

(continues on next page)

(continued from previous page)

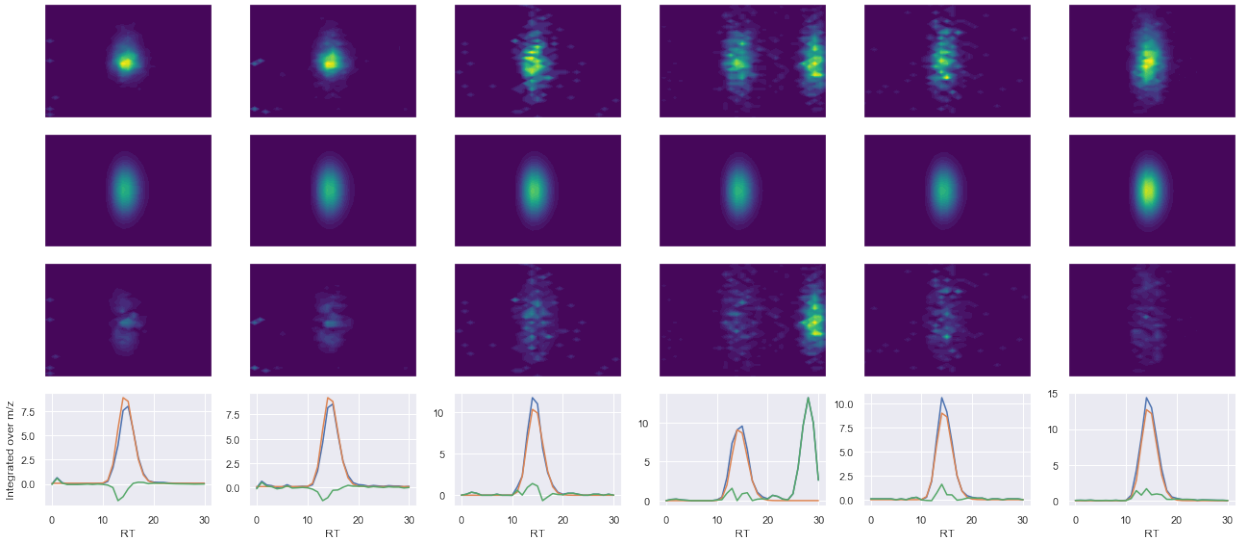
```
            print(f'peptide, charge, rank, rt, pre = \'{peptide}\', {charge}, {rank},
↪{rt}, {pre[0]}')
            print(f'pro = {pro}')
            assert len(pre) > 0
            assert len(pro) > 0

            peak_group = requant.RequantPeakGroup(peptide, charge, rank, rt, pre[0],␣
↪pro)
            print(operator.run(ms2_name, [peak_group]))
```

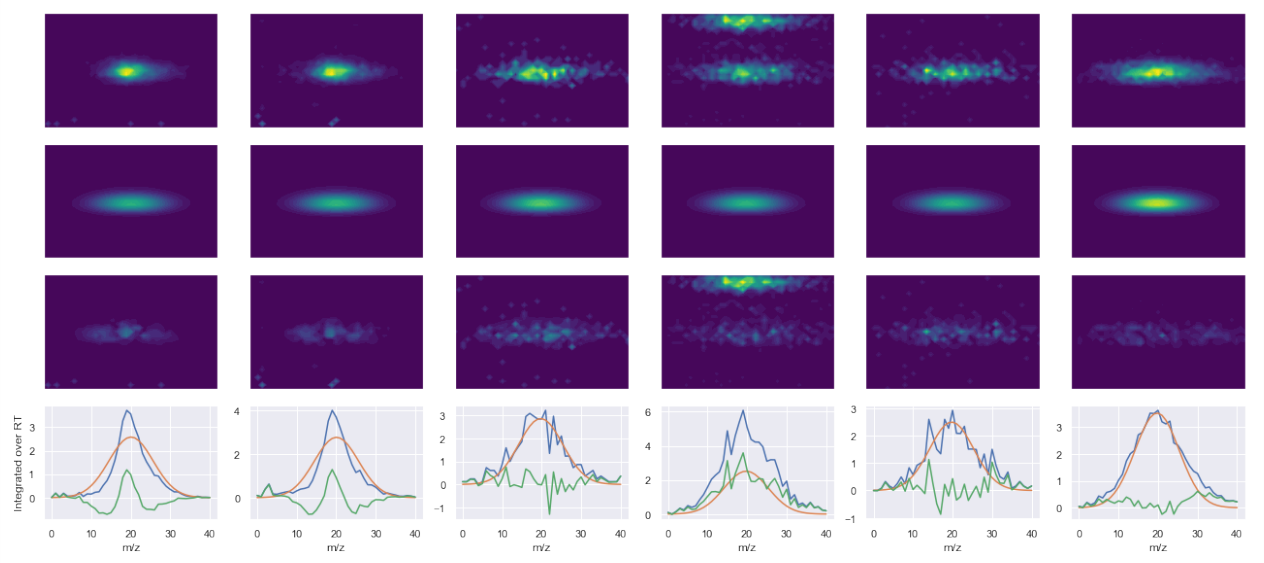
Quantify & visualise the high scoring peptides

SGS data

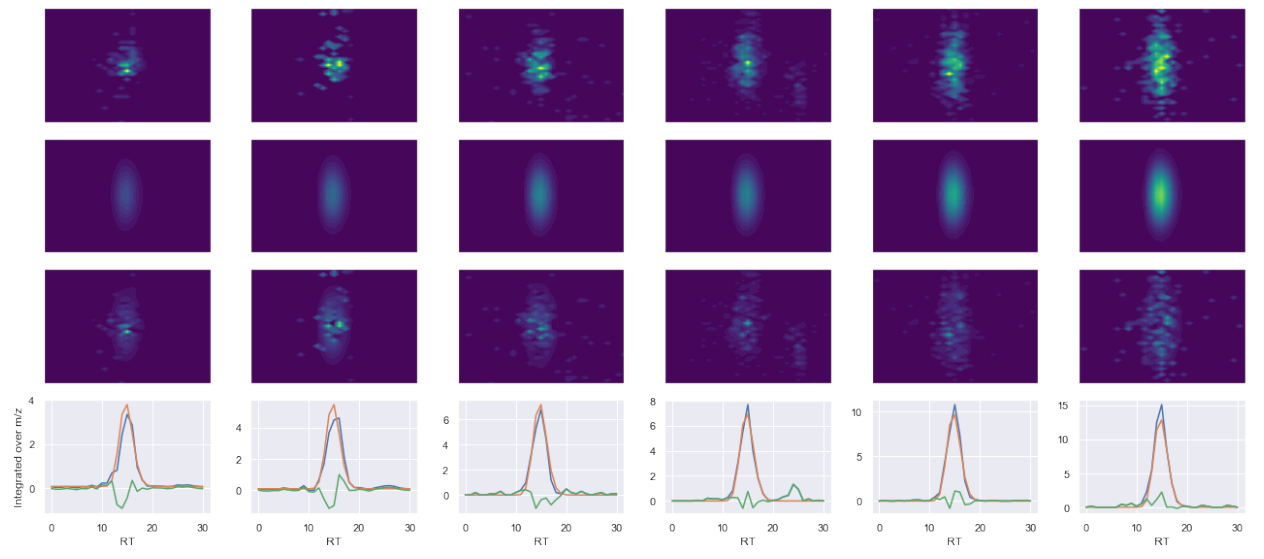
[17]:
```
run_python_requant(
    file=sgs_file,
    quantifiers=sgs_quantifiers,
    peptide_ids=sgs_peptide_ids,
)
```

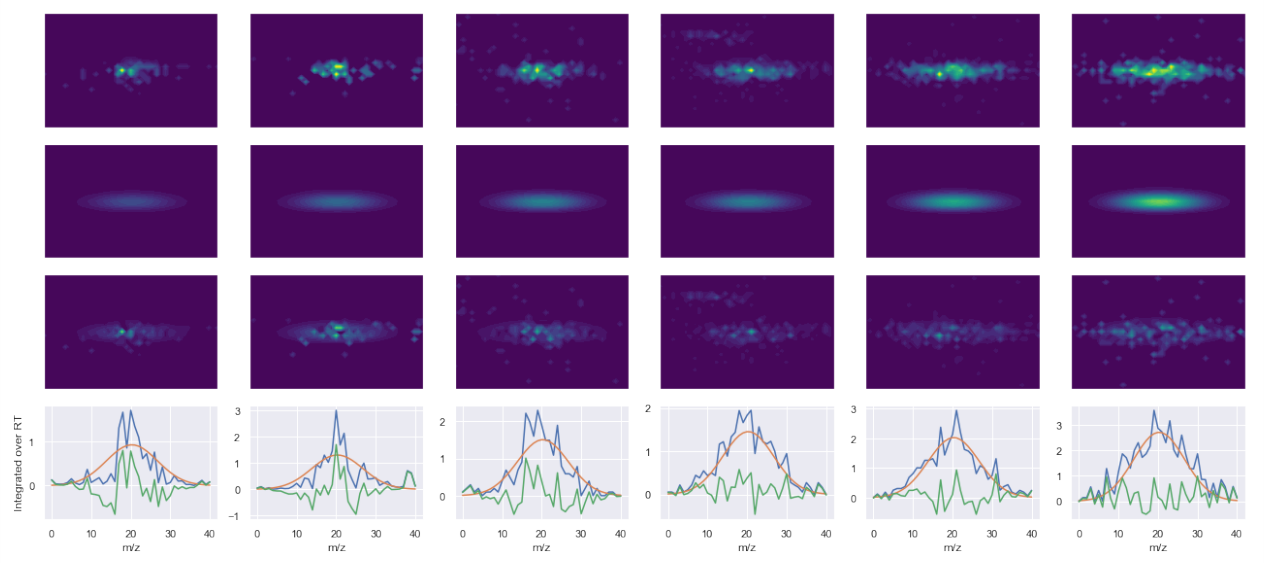
```
********************************************************************************
--------------------------------
ms2_name = 'ms2-008'
peptide, charge, rank, rt, pre = 'IVDVDLTSEGK(UniMod:259)', 2, 1, 2962.99, 592.318
pro = [642.355, 757.382, 856.45, 971.477]
```



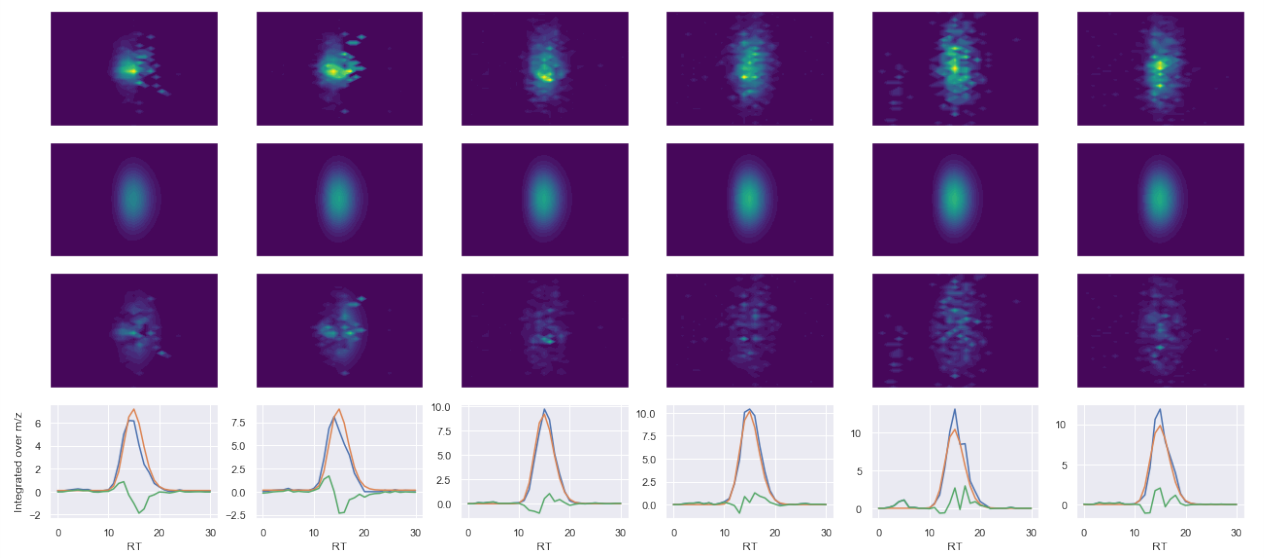


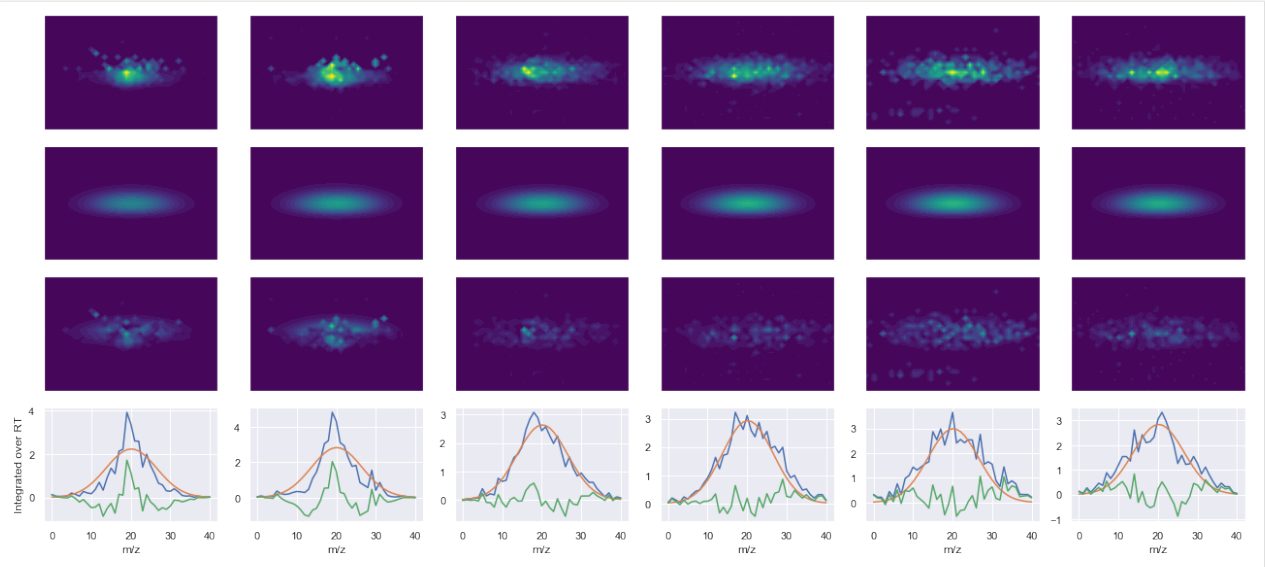

```
Raw MS1 & MS2: 1901.0151427865394 832.5617156912243
Fit MS1 & MS2: 1913.1799130745012 614.3915440003645
[('IVDVDLTSEGK(UniMod:259)', 2, 1913.1799130745012, 614.3915440003645)]
---------------------------------
ms2_name = 'ms2-008'
peptide, charge, rank, rt, pre = 'NPDGTVTVISR(UniMod:267)', 2, 1, 2360.46, 584.813
pro = [484.312, 684.428, 842.497, 957.524]
```



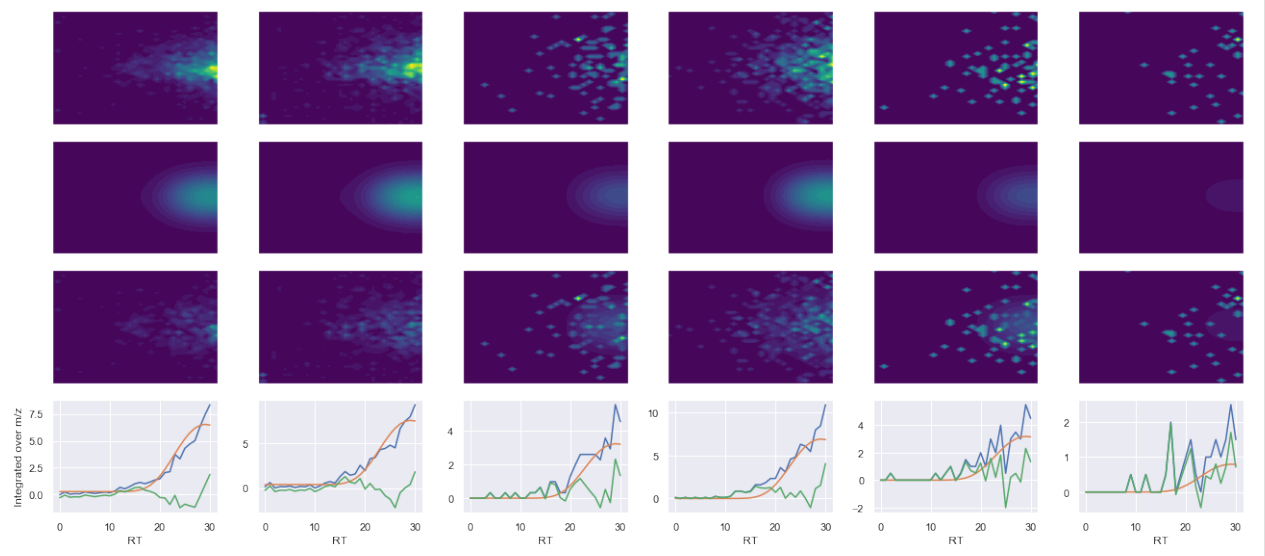


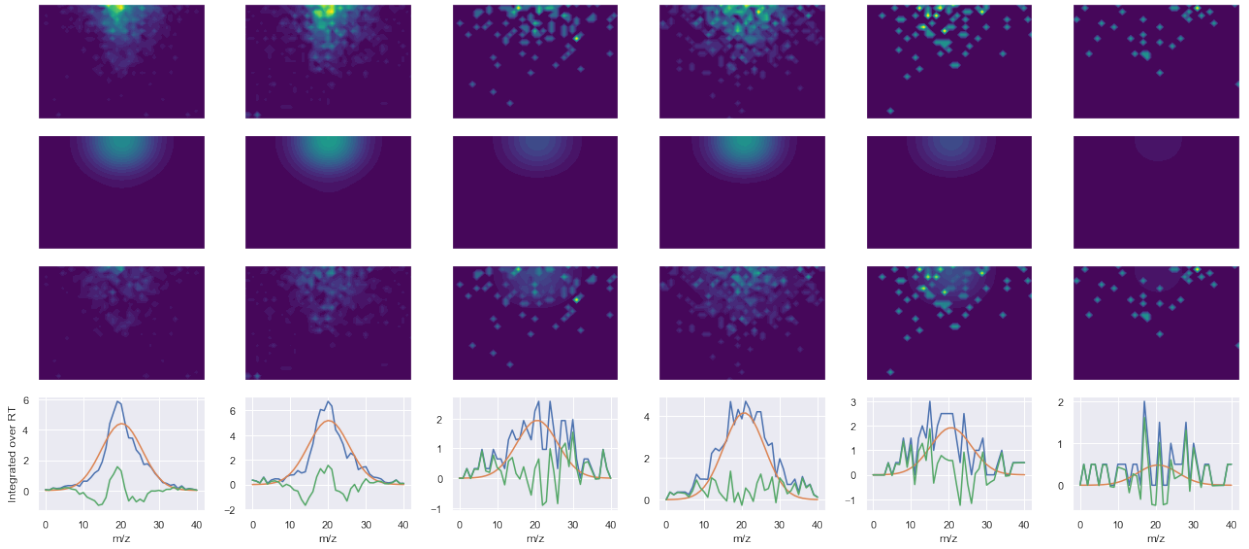

```
Raw MS1 & MS2: 1068.797569083983 226.6386387566327
Fit MS1 & MS2: 964.1610009504825 201.86531477130598
[('NPDGTVTVISR(UniMod:267)', 2, 964.1610009504825, 201.86531477130598)]
********************************************************************************
----------------------------------
ms2_name = 'ms2-010'
peptide, charge, rank, rt, pre = 'IVEIYGPESSGK(UniMod:259)', 2, 1, 2900.32, 643.84
pro = [669.329, 832.393, 945.477, 1074.52]
```






```
Raw MS1 & MS2: 1343.0514804124023 587.4011974819572
Fit MS1 & MS2: 1361.4908623989802 540.0065696055367
[('IVEIYGPESSGK(UniMod:259)', 2, 1361.4908623989802, 540.0065696055367)]
**********************************************************************************
----------------------------------
ms2_name = 'ms2-014'
peptide, charge, rank, rt, pre = 'SDSSDTPPLPSPPGK(UniMod:259)', 2, 1, 2804.66, 745.367
pro = [800.476, 897.528, 998.576, 1113.6]
```



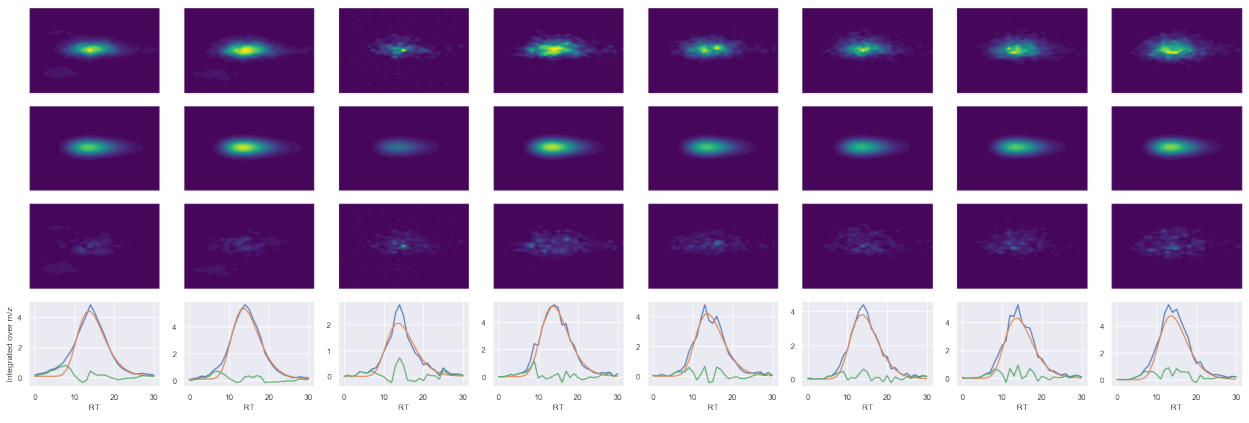


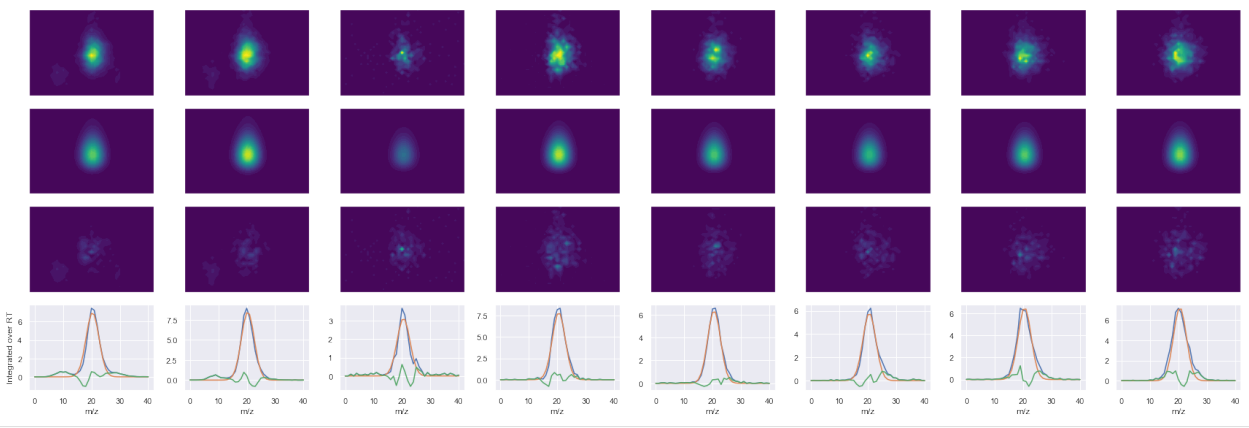

```
Raw MS1 & MS2: 592.4653454073219 122.54244489367036
Fit MS1 & MS2: 505.67140106941656 88.20516067864703
[('SDSSDTPPLPSPPGK(UniMod:259)', 2, 505.67140106941656, 88.20516067864703)]
```

ProCan90 data

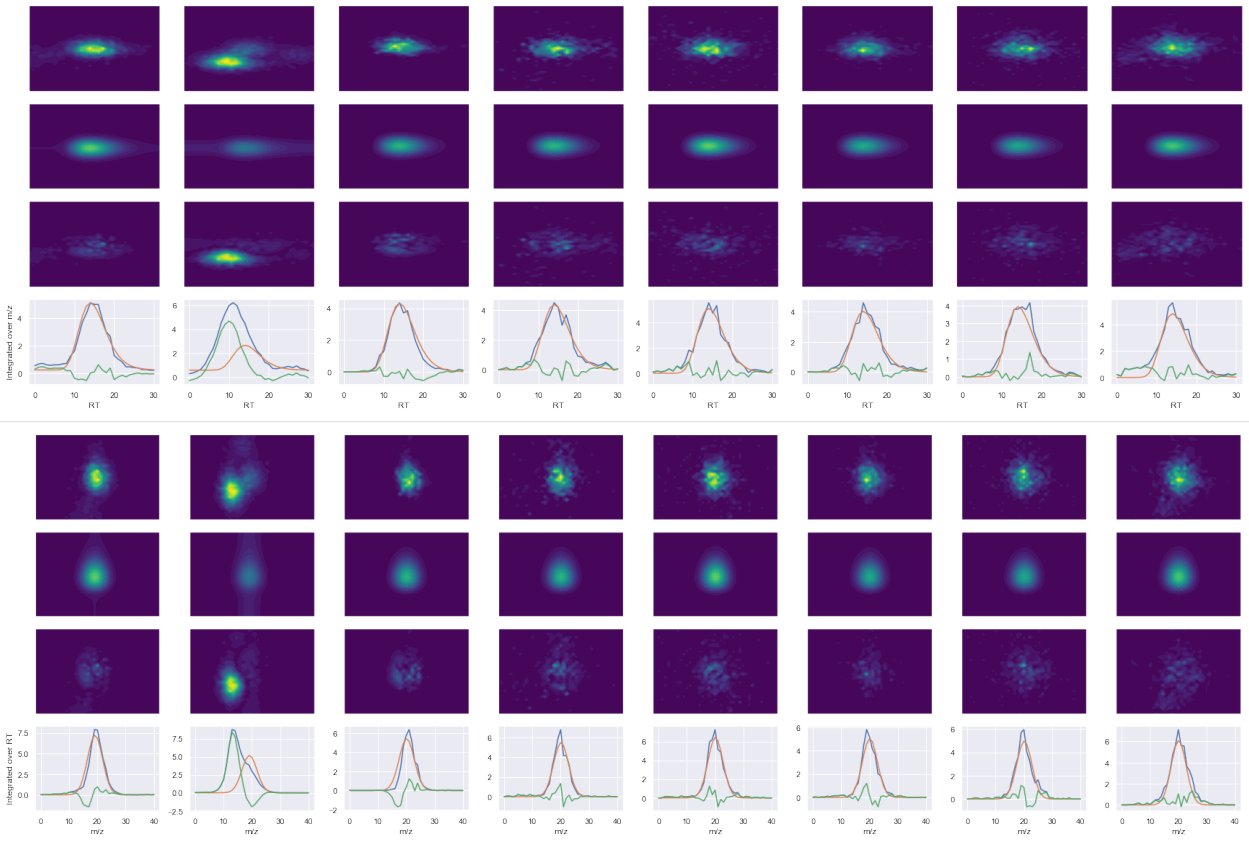
[18]:
```
run_python_requant(
    file=procan90_file,
    quantifiers=procan90_quantifiers,
    peptide_ids=procan90_peptide_ids,
)
```

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-024'
peptide, charge, rank, rt, pre = 'VSFELFADK', 2, 1, 3025.44, 528.27404785
pro = [435.22381592, 480.24526978, 593.3293457, 722.37194824, 869.44036865, 956.
↪47235107]
```





```
Raw MS1 & MS2: 28778.380265487547 6040.770575850728
Fit MS1 & MS2: 24396.972393554213 5284.702891876719
[('VSFELFADK', 2, 24396.972393554213, 5284.702891876719)]
**********************************************************************
--------------------------------
ms2_name = 'ms2-035'
peptide, charge, rank, rt, pre = 'AIPQLQGYLR', 2, 1, 2607.96, 579.83514404
pro = [487.77453613, 508.28781128, 636.34637451, 749.43041992, 877.48901367, 974.
↪54180908]
```

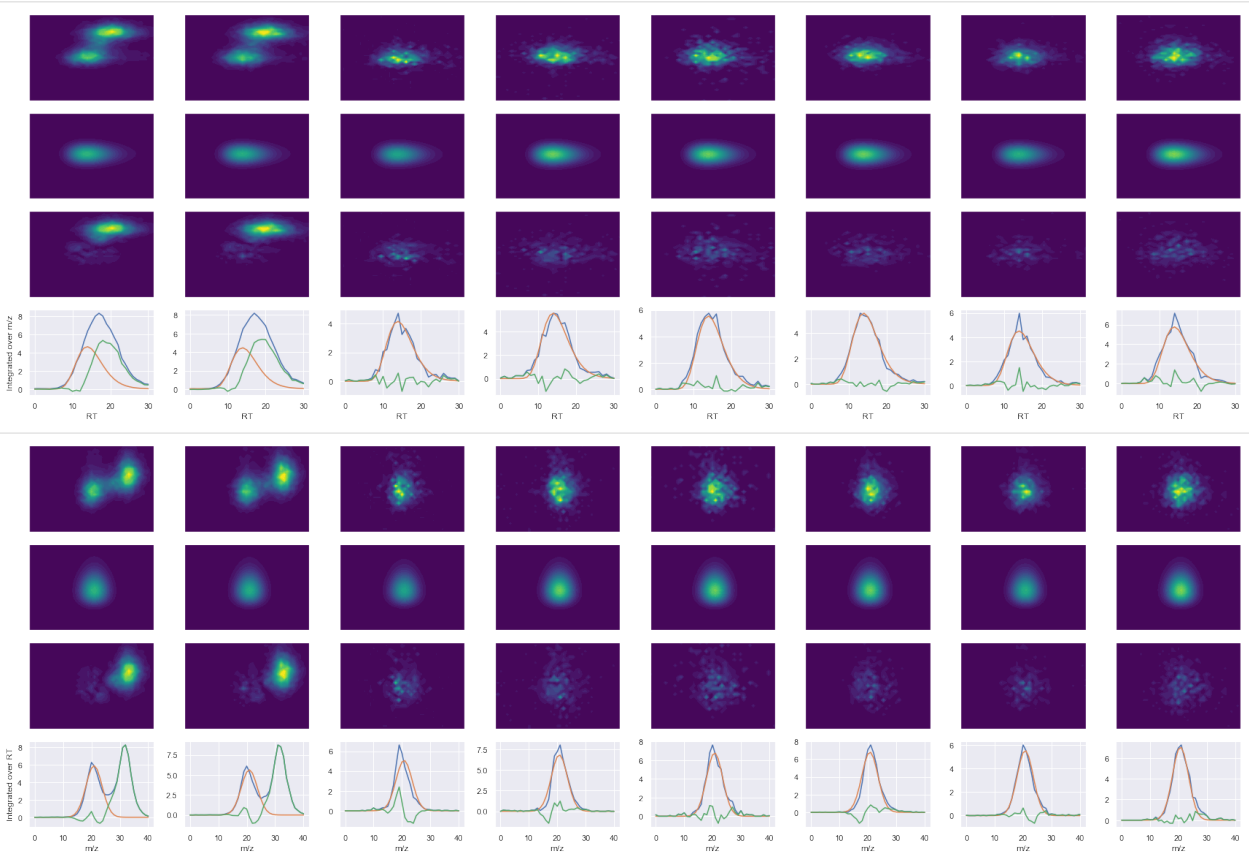


```
Raw MS1 & MS2: 30576.567646871867 4026.2126532143407
Fit MS1 & MS2: 12613.59797450274 3776.8008016967888
[('AIPQLQGYLR', 2, 12613.59797450274, 3776.8008016967888)]
**********************************************************************
```

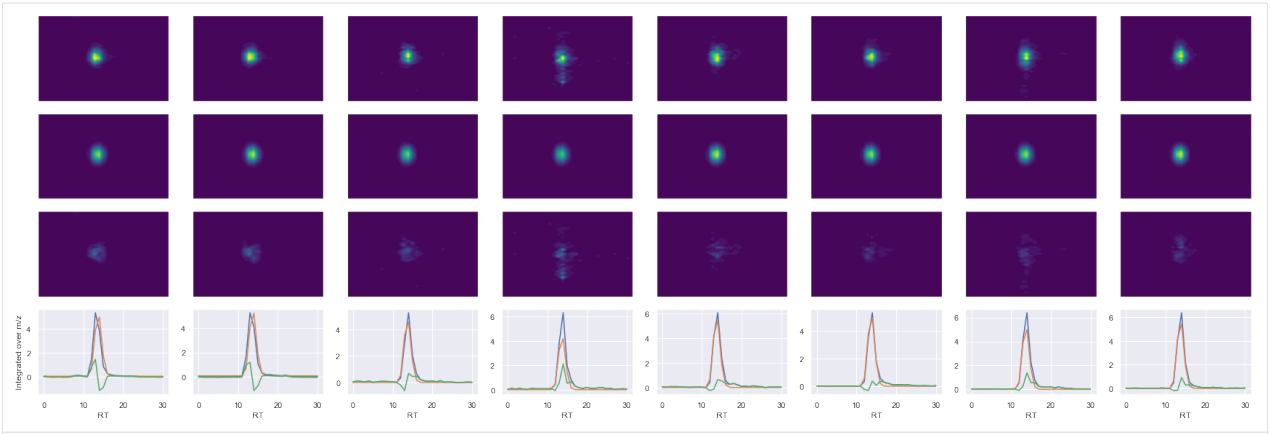
(continues on next page)

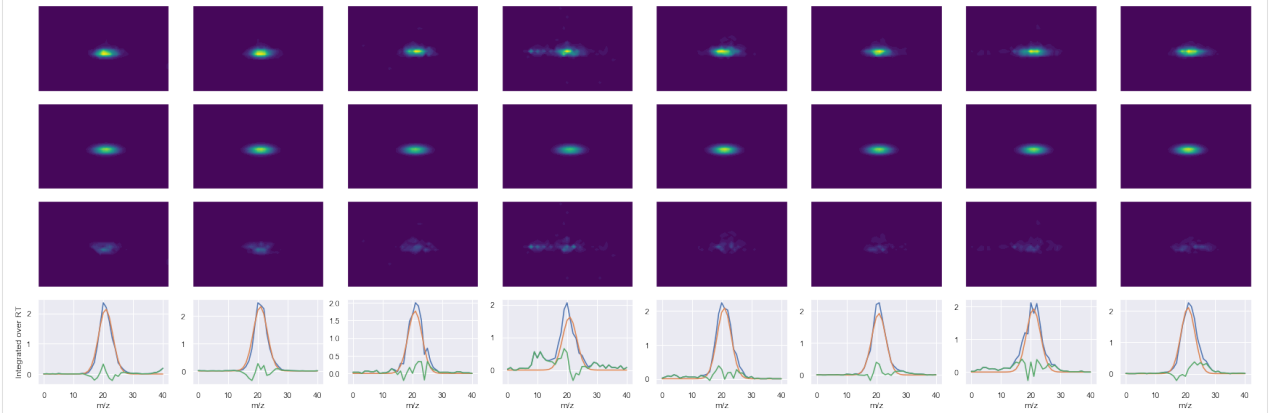
(continued from previous page)

```
---------------------------------
ms2_name = 'ms2-048'
peptide, charge, rank, rt, pre = 'NALESYAFNMK', 2, 1, 2491.33, 644.30554199
pro = [539.26464844, 610.30175781, 773.36505127, 860.39709473, 989.43969727, 1102.
↪5238037]
```

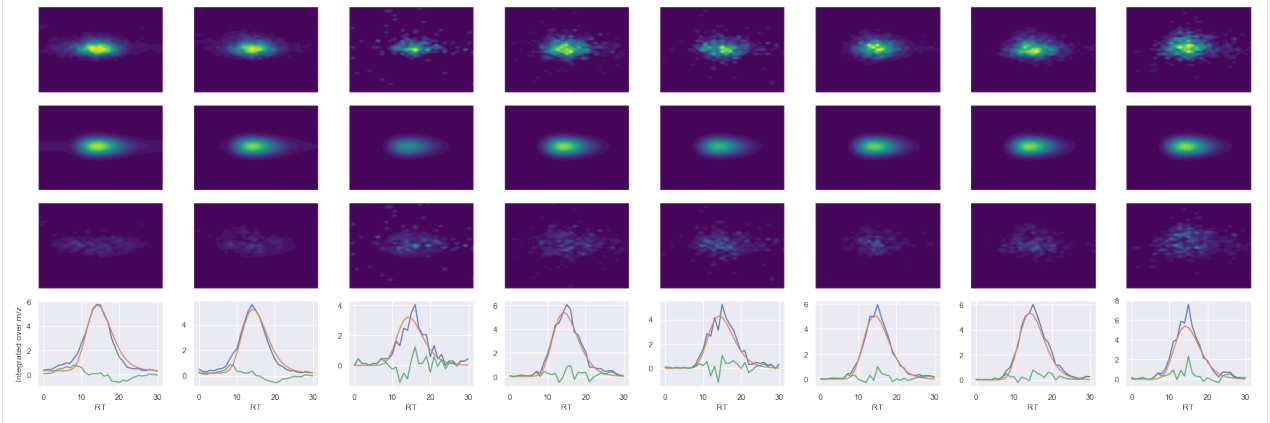


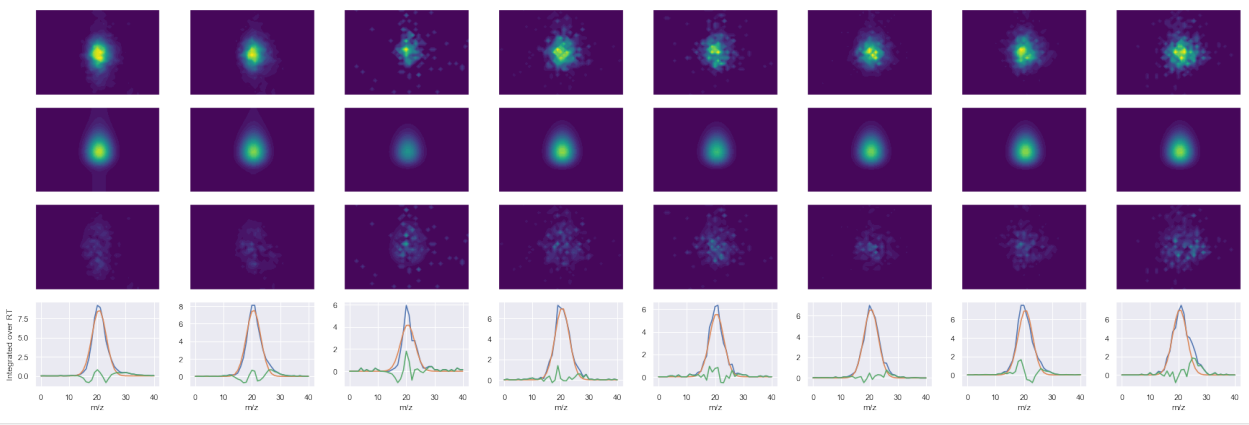

```
Raw MS1 & MS2: 69309.96573677666 4298.2830446508515
Fit MS1 & MS2: 29909.713574711433 4079.6536675590323
[('NALESYAFNMK', 2, 29909.713574711433, 4079.6536675590323)]
**********************************************************************************
---------------------------------
ms2_name = 'ms2-049'
peptide, charge, rank, rt, pre = 'VIHDNFGIVEGLMTTVHAITATQK', 4, 1, 4158.86, 649.
↪59545898
pro = [548.30383301, 579.28851318, 732.42504883, 783.37841797, 869.48394775, 896.
↪46246338]
```






```
Raw MS1 & MS2: 36075.46972826726 4581.8361977096065
Fit MS1 & MS2: 31349.392247620683 3715.5025255865135
[('VIHDNFGIVEGLMTTVHAITATQK', 4, 31349.392247620683, 3715.5025255865135)]
********************************************************************************
--------------------------------
ms2_name = 'ms2-055'
peptide, charge, rank, rt, pre = 'GGGGNFGPGPGSNFR', 2, 1, 1180.7, 689.31835938
pro = [580.28381348, 677.33654785, 734.35803223, 831.4107666, 888.43225098, 1035.
↪5006104]
```





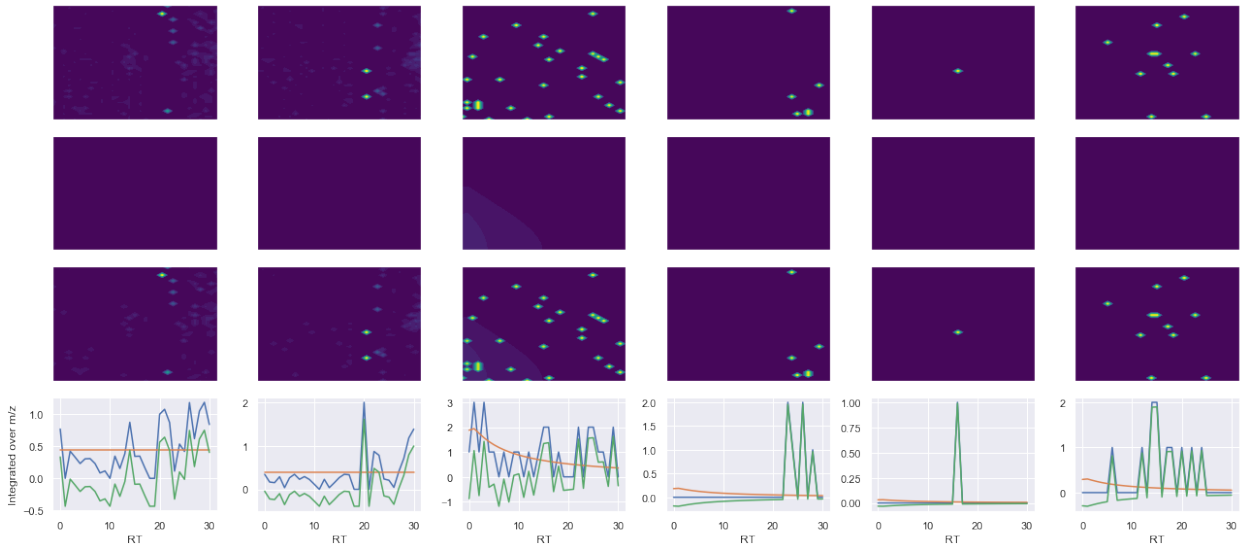
```
Raw MS1 & MS2: 18984.09727504625 3965.848553410137
Fit MS1 & MS2: 15462.107708268175 3531.5971517279813
[('GGGGNFGPGPGSNFR', 2, 15462.107708268175, 3531.5971517279813)]
```

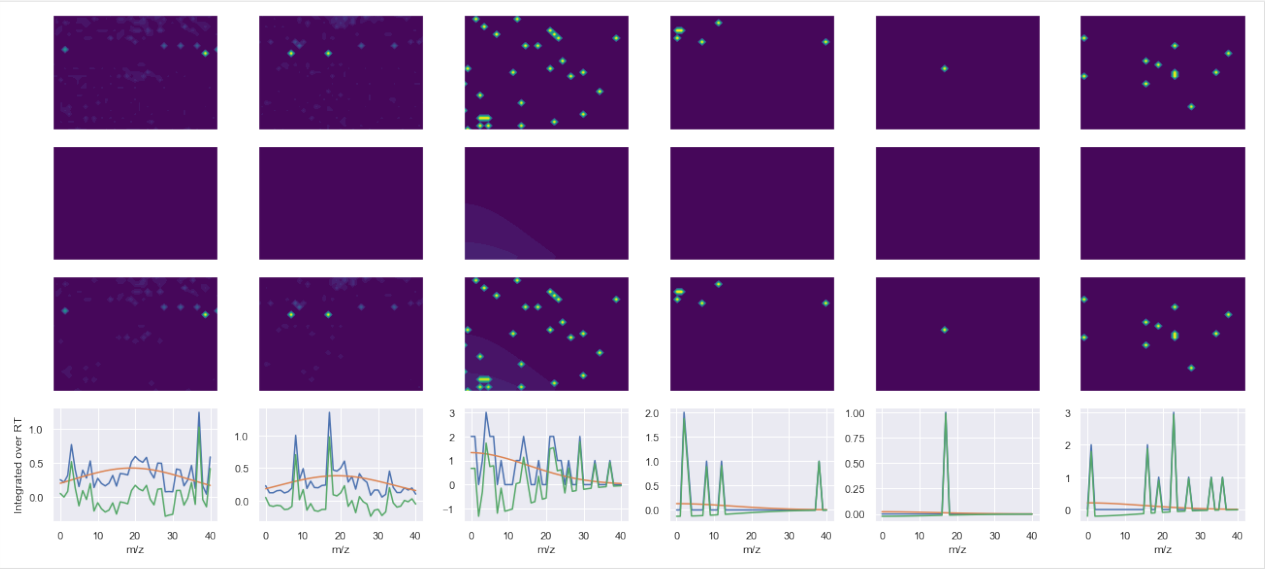
Quantify & visualise the outlier peptides

SGS data

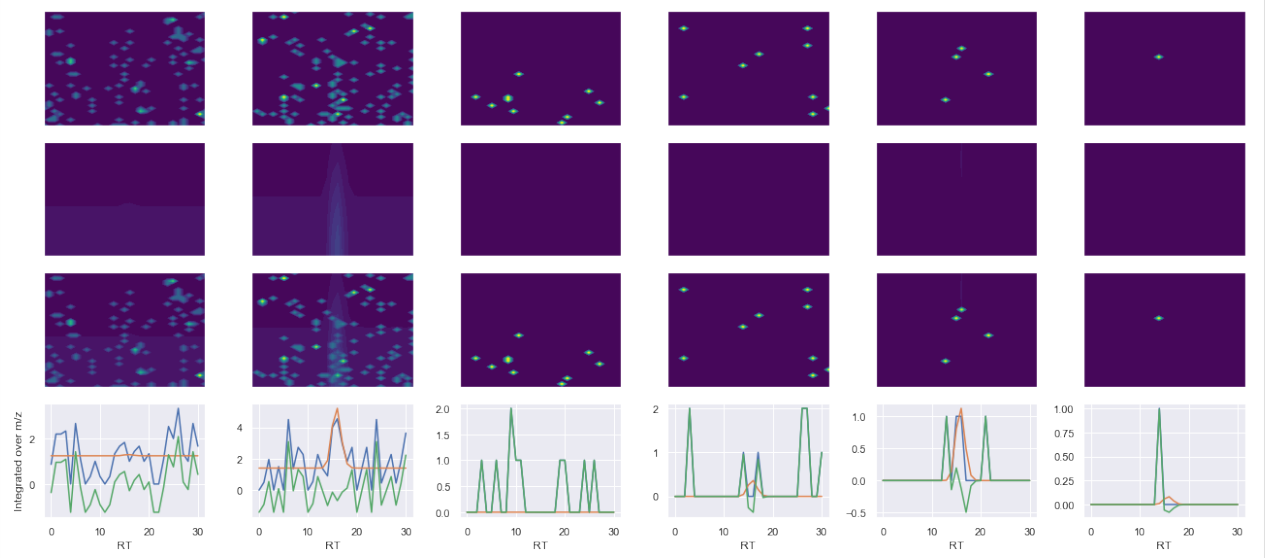
[19]:
```
run_python_requant(
    file=sgs_file,
    quantifiers=sgs_quantifiers,
    peptide_ids=sgs_outlier_peptide_ids,
)
```

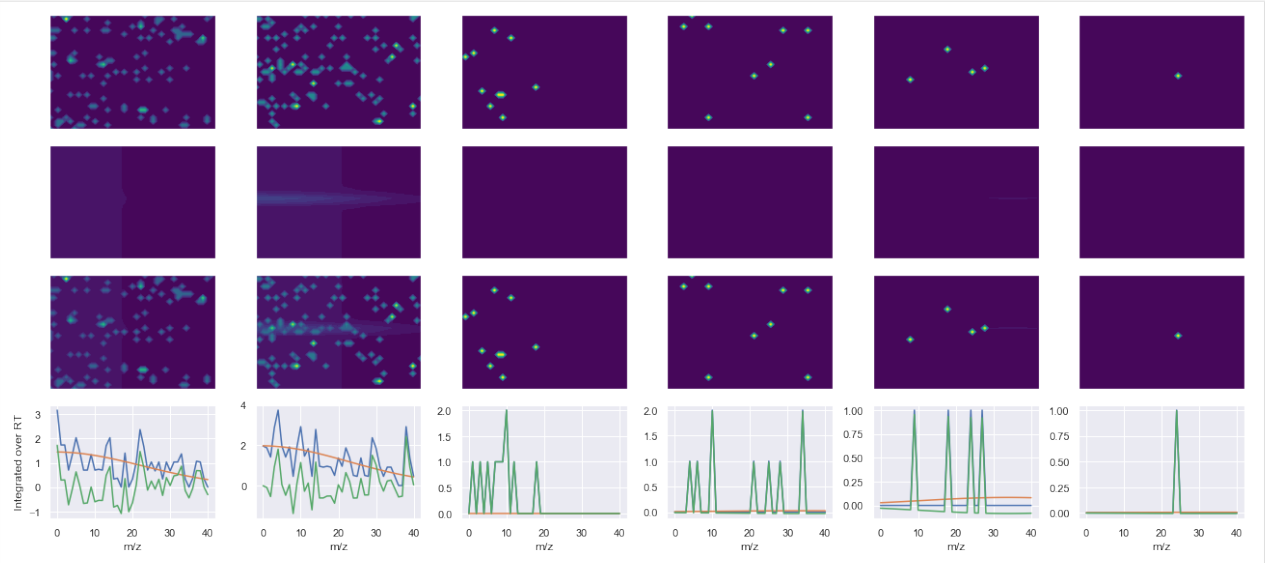
```
********************************************************************************
--------------------------------
ms2_name = 'ms2-011'
peptide, charge, rank, rt, pre = 'YYDYTLSINGK(UniMod:259)', 2, 1, 3657.93, 672.832
pro = [526.307, 639.392, 740.439, 1018.53]
```



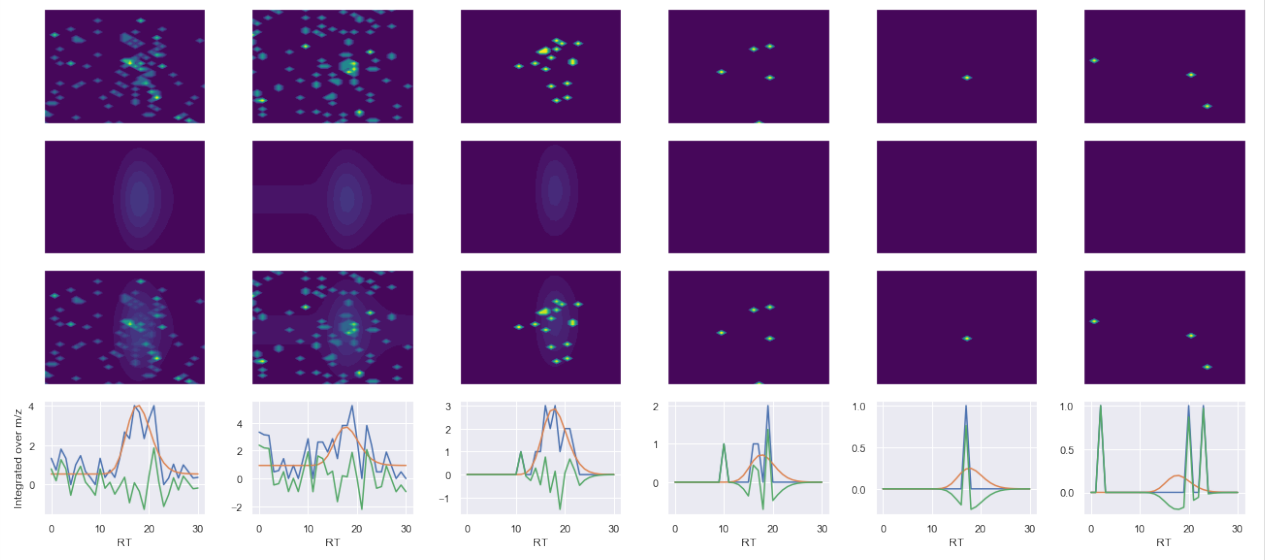


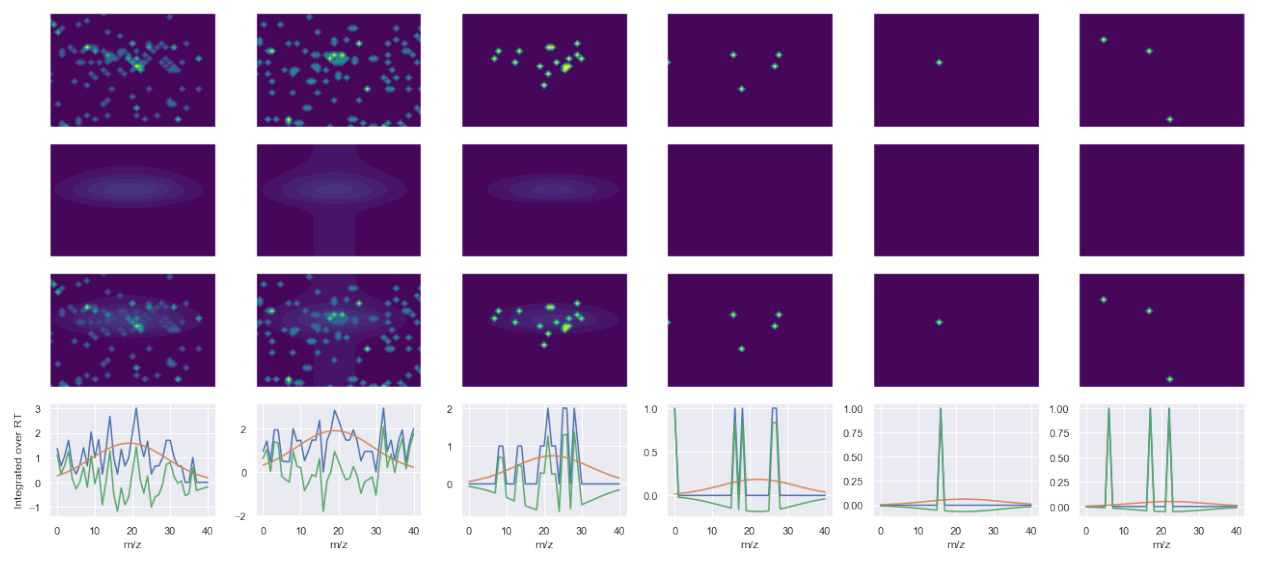

```
Raw MS1 & MS2: 93.2215802795607 5.8908822504070155
Fit MS1 & MS2: 6.277257202415132e-06 3.8327705199061106
[('YYDYTLSINGK(UniMod:259)', 2, 6.277257202415132e-06, 3.8327705199061106)]
********************************************************************************
----------------------------------
ms2_name = 'ms2-012'
peptide, charge, rank, rt, pre = 'TIVGVSEDFAALR(UniMod:267)', 2, 1, 4574.05, 694.376
pro = [587.354, 918.455, 1074.55, 1173.61]
```



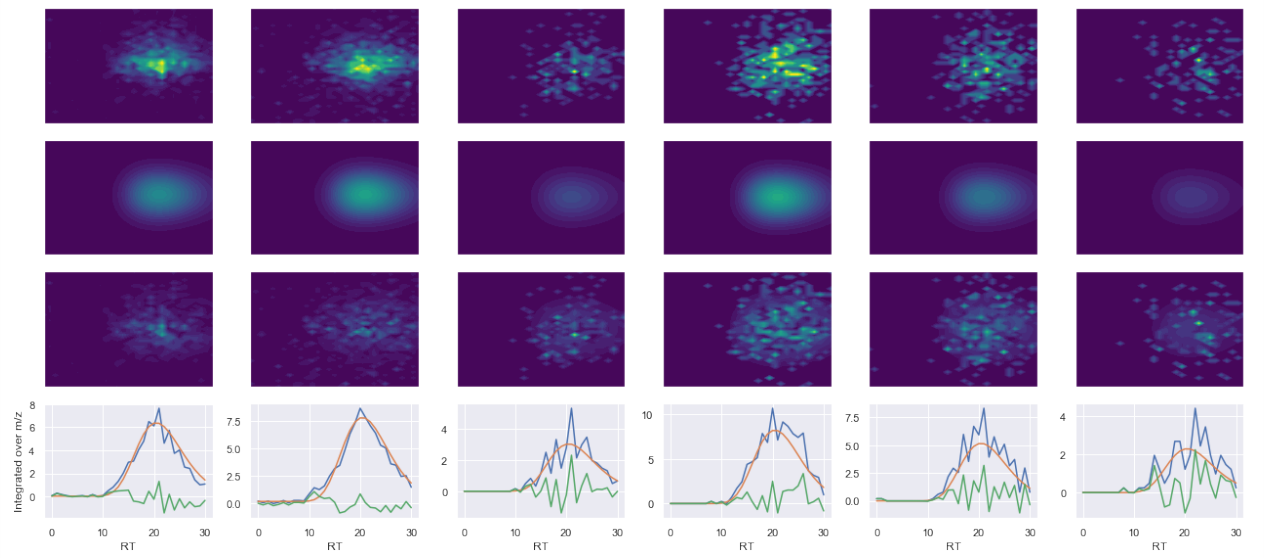


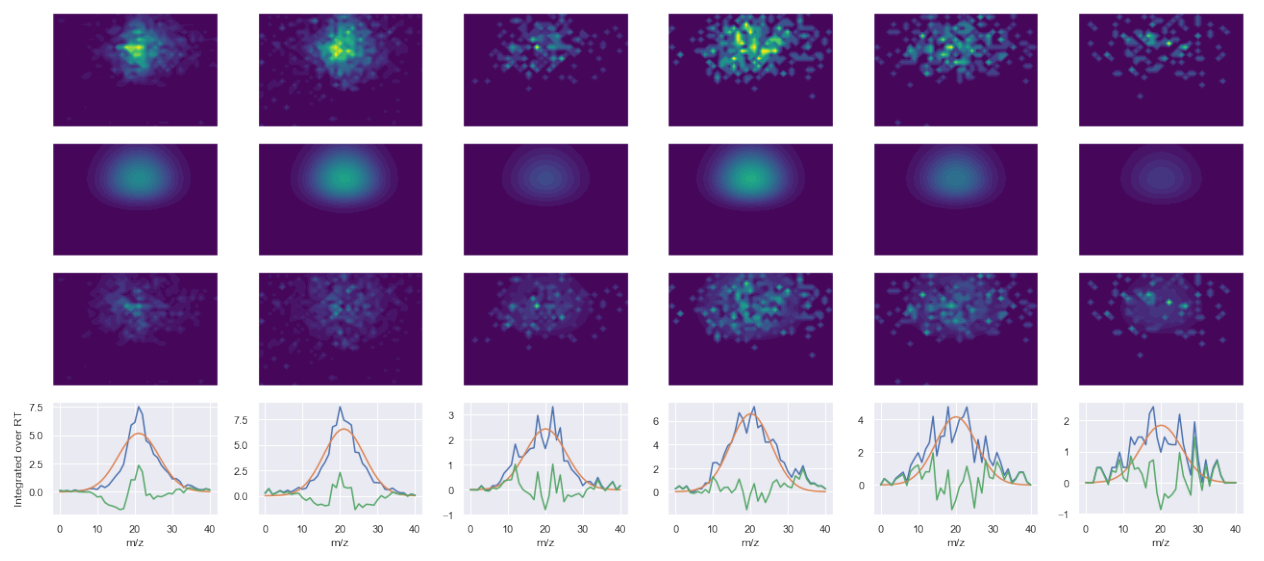

```
Raw MS1 & MS2: 30.68848719714169 3.290575667970753
Fit MS1 & MS2: 2.5708773559614384 0.5779817408456864
[('TIVGVSEDFAALR(UniMod:267)', 2, 2.5708773559614384, 0.5779817408456864)]
----------------------------------
ms2_name = 'ms2-012'
peptide, charge, rank, rt, pre = 'VESPVLPPVLVPK(UniMod:259)', 2, 1, 4738.54, 691.431
pro = [757.506, 870.59, 1066.71, 1153.74]
```



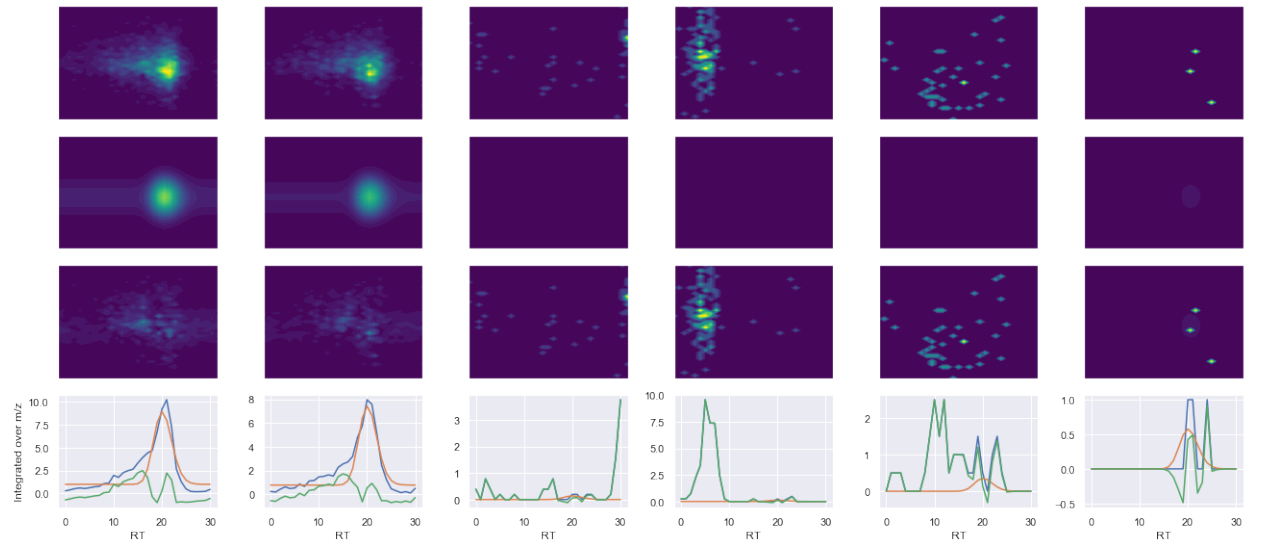


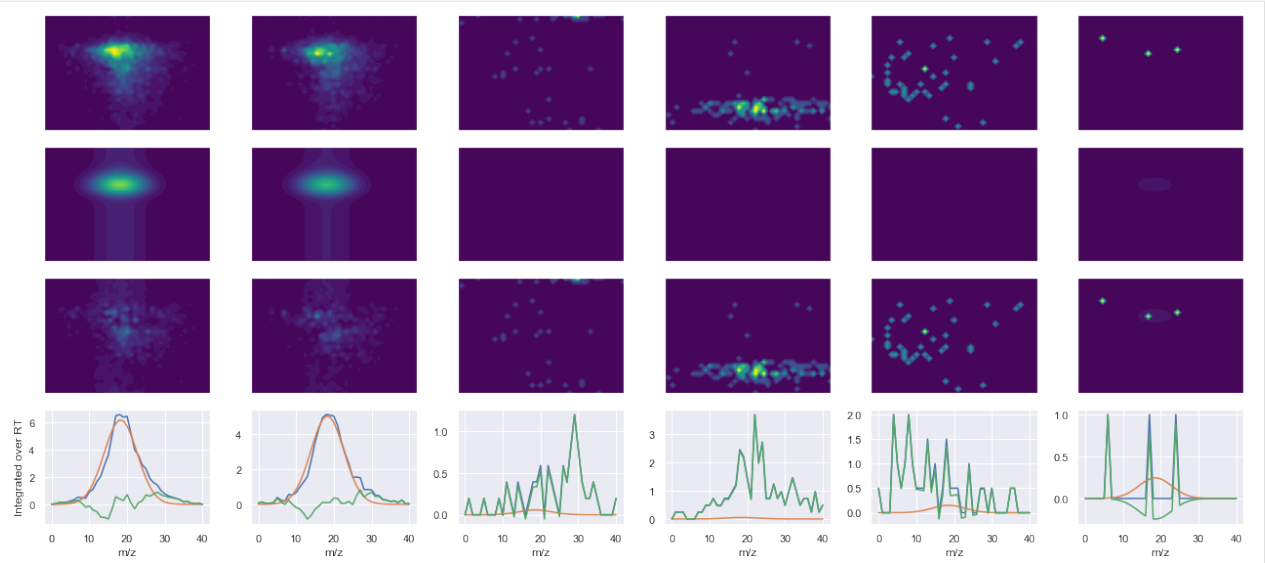

```
Raw MS1 & MS2: 30.267127515105777 3.626600767829161
Fit MS1 & MS2: 12.886623304127127 3.457505682641535
[('VESPVLPPVLVPK(UniMod:259)', 2, 12.886623304127127, 3.457505682641535)]
**********************************************************************************
----------------------------------
ms2_name = 'ms2-014'
peptide, charge, rank, rt, pre = 'EYTEGVNGQPSIR(UniMod:267)', 2, 1, 2438.35, 730.356
pro = [667.376, 781.419, 937.509, 1066.55]
```



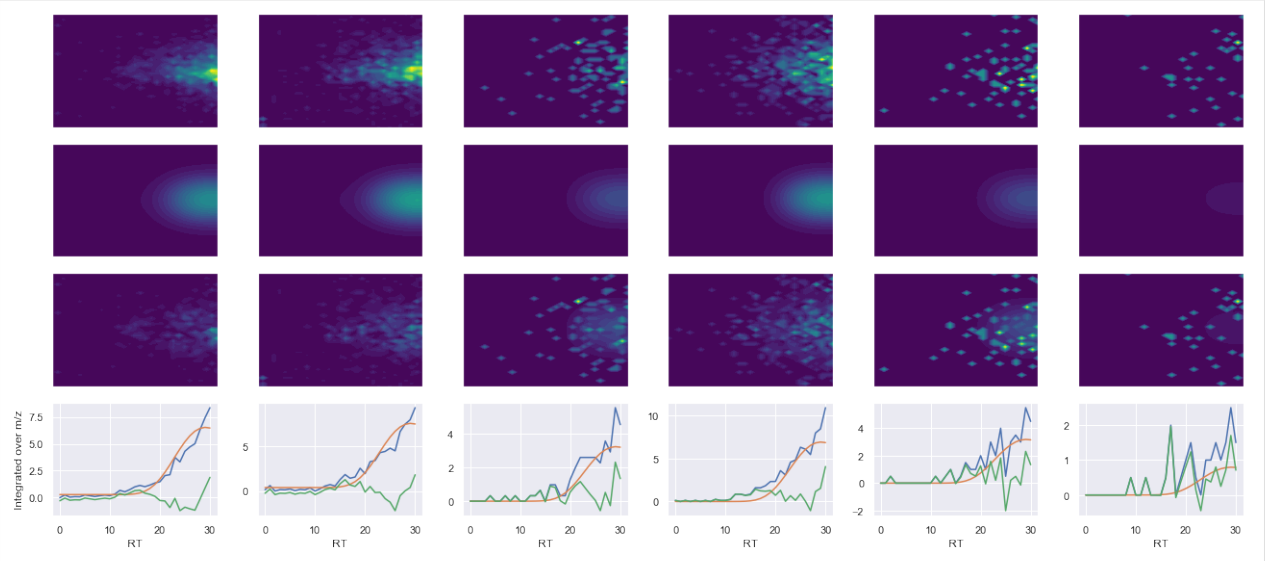


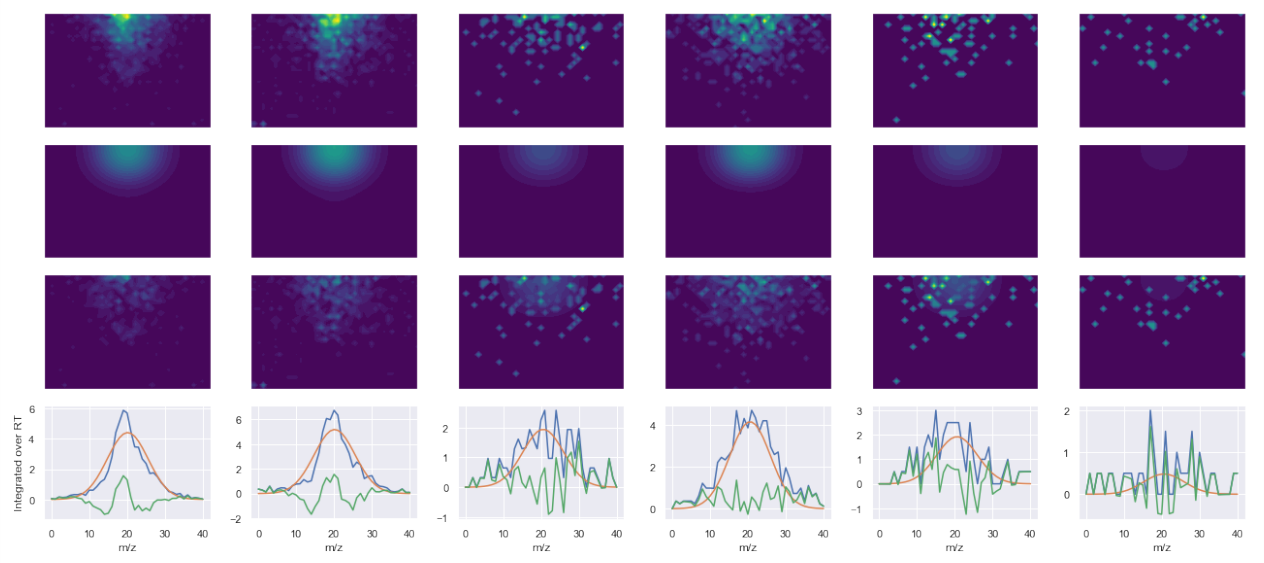

```
Raw MS1 & MS2: 571.6679698524539 159.05646570609215
Fit MS1 & MS2: 569.5681733014899 134.3360836457496
[('EYTEGVNGQPSIR(UniMod:267)', 2, 569.5681733014899, 134.3360836457496)]
----------------------------------
ms2_name = 'ms2-014'
peptide, charge, rank, rt, pre = 'MIVDPVEPHGEMK(UniMod:259)', 2, 1, 2851.66, 745.367
pro = [706.343, 835.386, 1031.51, 1245.6]
```



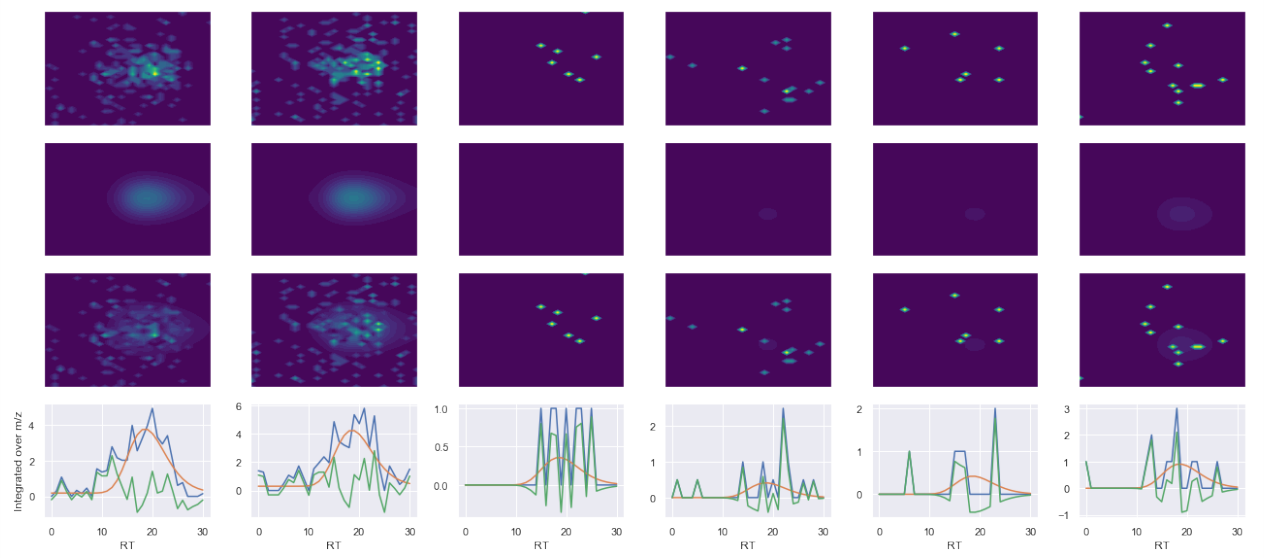


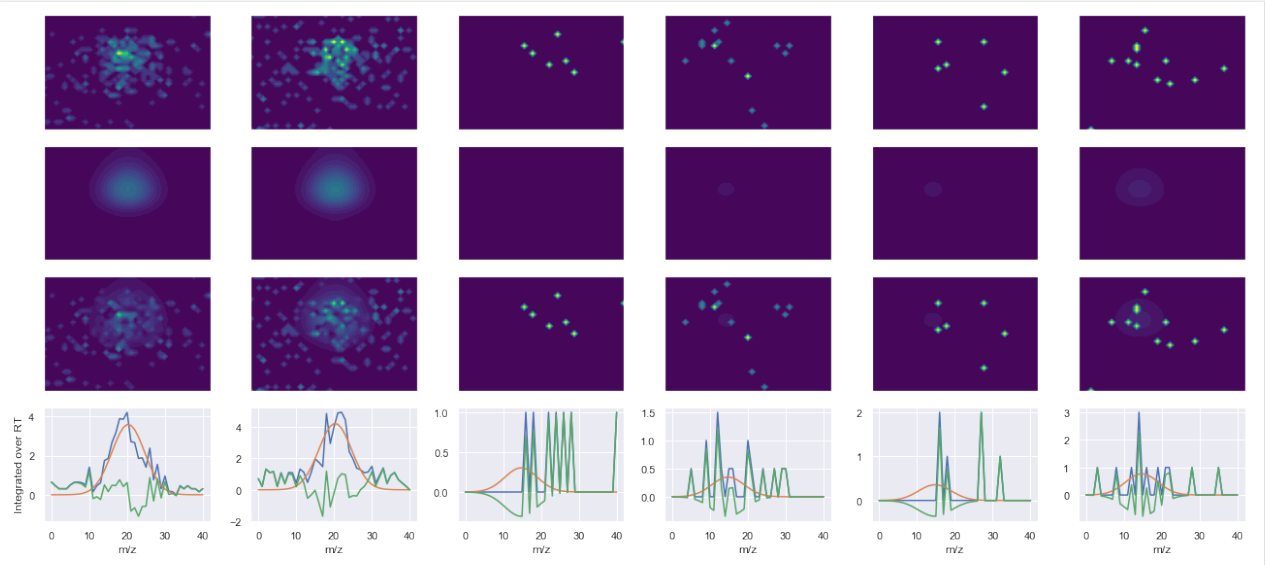

```
Raw MS1 & MS2: 1431.3119159291473 33.13252129485163
Fit MS1 & MS2: 732.8624744794847 1.724481319894012
[('MIVDPVEPHGEMK(UniMod:259)', 2, 732.8624744794847, 1.724481319894012)]
---------------------------------
ms2_name = 'ms2-014'
peptide, charge, rank, rt, pre = 'SDSSDTPPLPSPPGK(UniMod:259)', 2, 1, 2804.66, 745.367
pro = [800.476, 897.528, 998.576, 1113.6]
```



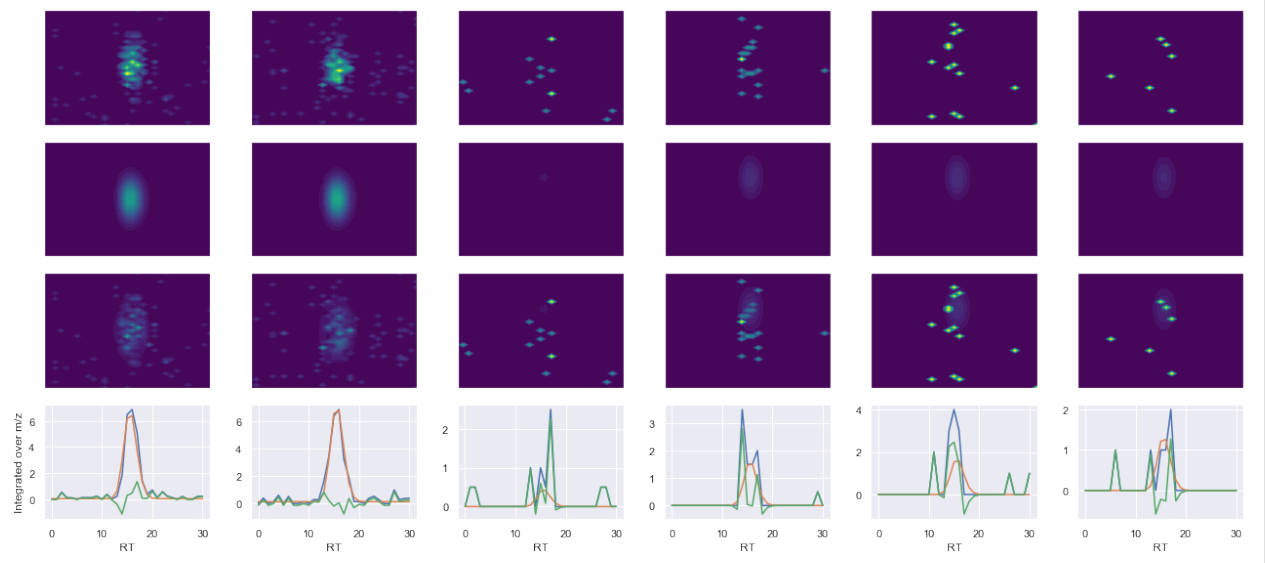


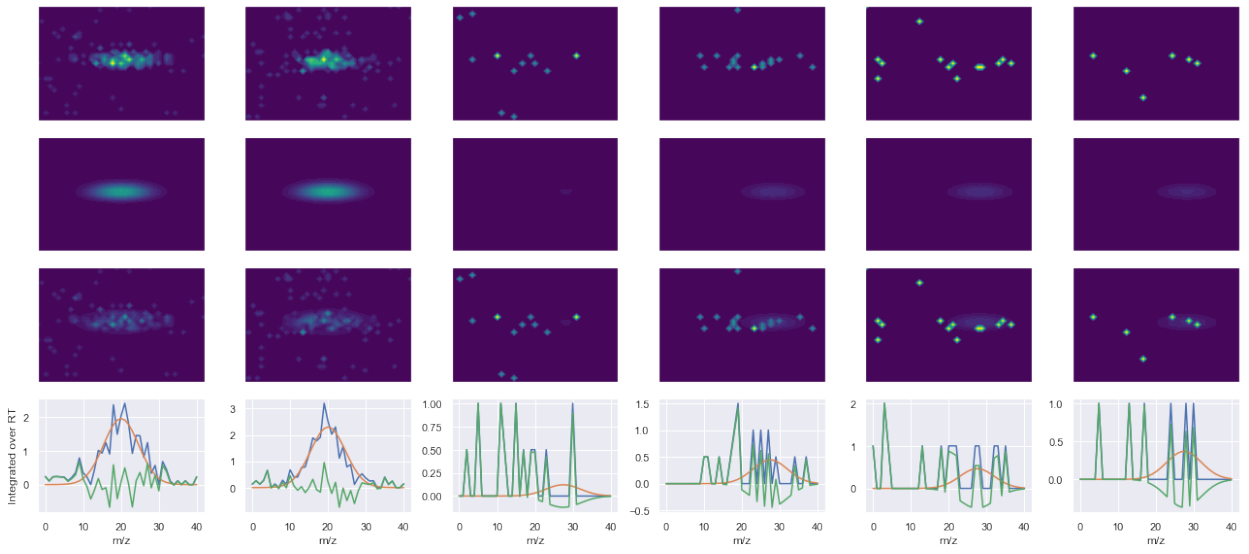

```
Raw MS1 & MS2: 592.4653454073219 122.54244489367036
Fit MS1 & MS2: 505.67140106941656 88.20516067864703
[('SDSSDTPPLPSPPGK(UniMod:259)', 2, 505.67140106941656, 88.20516067864703)]
********************************************************************************
----------------------------------
ms2_name = 'ms2-015'
peptide, charge, rank, rt, pre = 'FPSPVSHADDLYGK(UniMod:259)', 2, 1, 3293.18, 770.88
pro = [789.387, 1013.48, 1209.6, 1296.63]
```






```
Raw MS1 & MS2: 87.85798084824421 6.512684995738631
Fit MS1 & MS2: 56.37263352773063 3.721348294688908
[('FPSPVSHADDLYGK(UniMod:259)', 2, 56.37263352773063, 3.721348294688908)]
---------------------------------
ms2_name = 'ms2-015'
peptide, charge, rank, rt, pre = 'NTDFNGVNNIHQK(UniMod:259)', 2, 1, 2128.47, 754.87
pro = [647.371, 917.504, 1031.55, 1178.62]
```



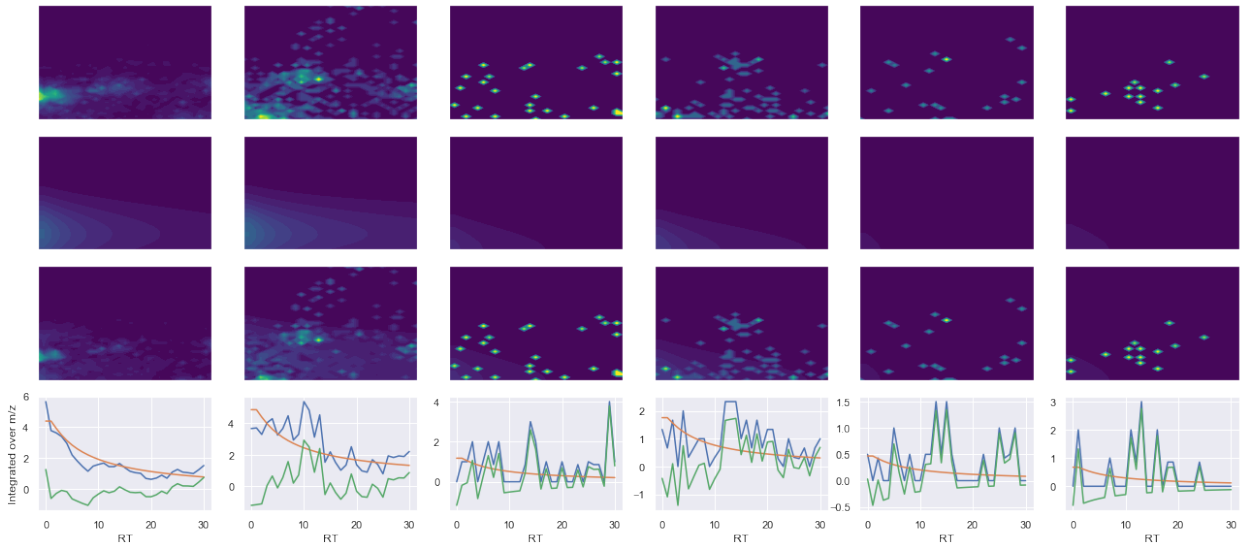


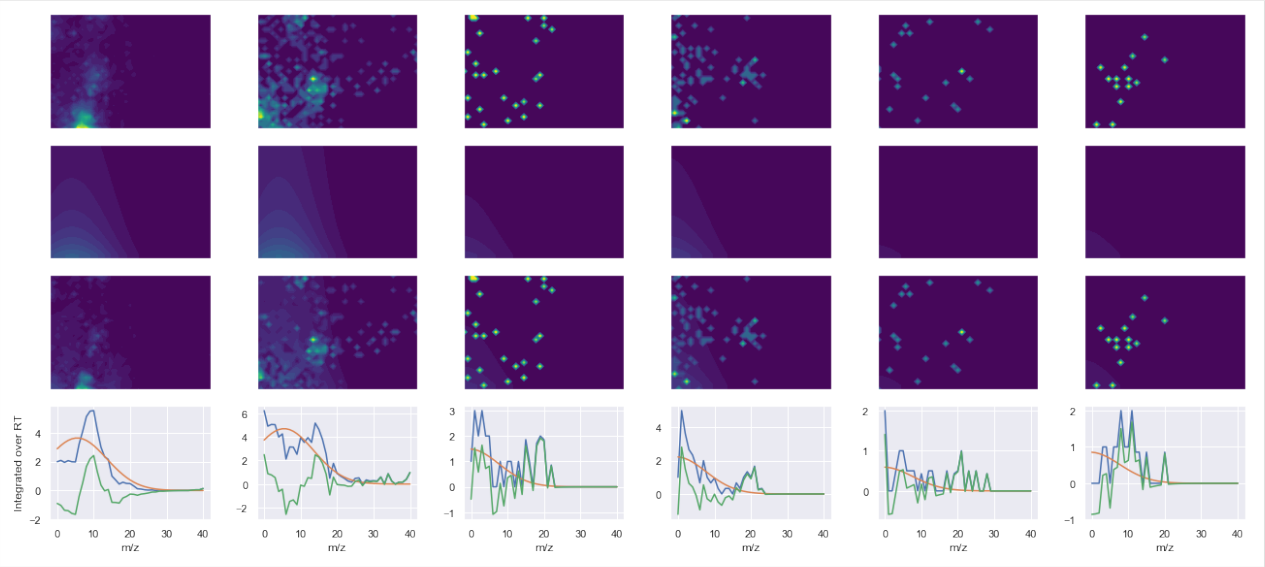

```
Raw MS1 & MS2: 77.90595348022188 7.52938575325957
Fit MS1 & MS2: 61.73288675398399 3.375192854570683
[('NTDFNGVNNIHQK(UniMod:259)', 2, 61.73288675398399, 3.375192854570683)]
```

ProCan90 data

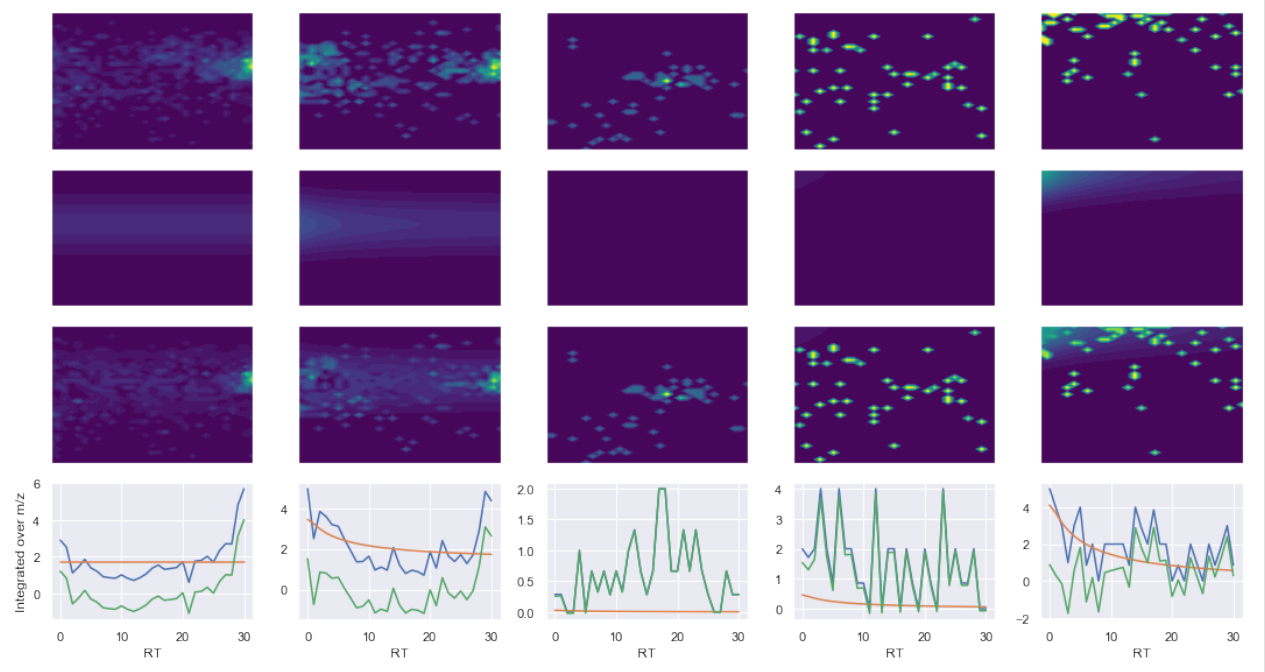
[20]:
```
run_python_requant(
    file=procan90_file,
    quantifiers=procan90_quantifiers,
    peptide_ids=procan90_outlier_peptide_ids,
)
```

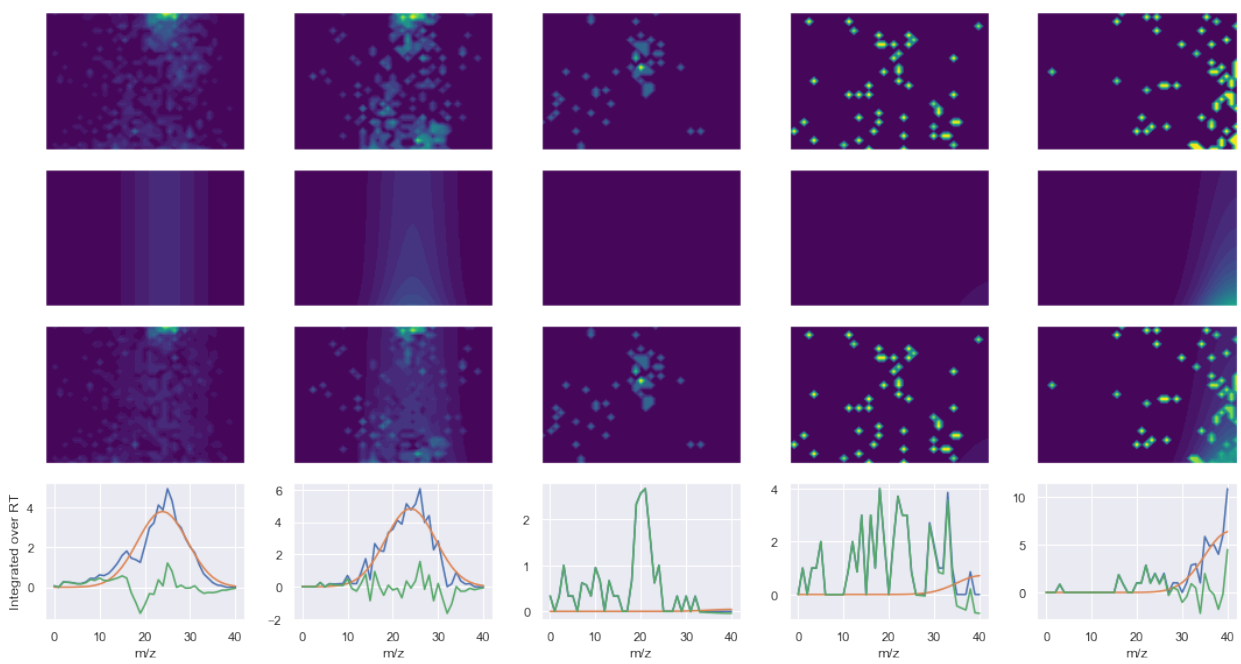
```
********************************************************************************
---------------------------------
ms2_name = 'ms2-004'
peptide, charge, rank, rt, pre = 'QNVPIIR', 2, 1, 1062.29, 420.2585144
pro = [401.28707886, 498.33984375, 597.40826416, 711.45117188]
```



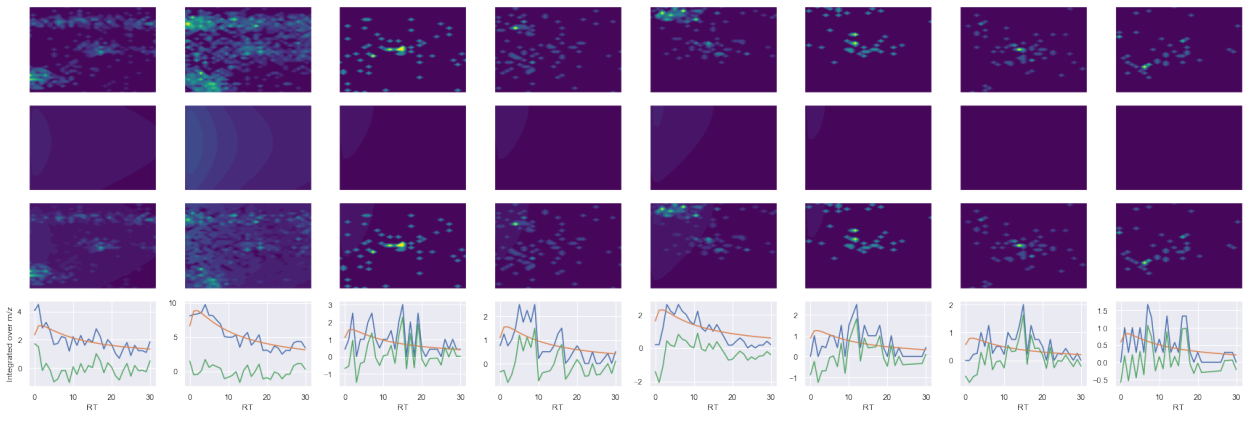


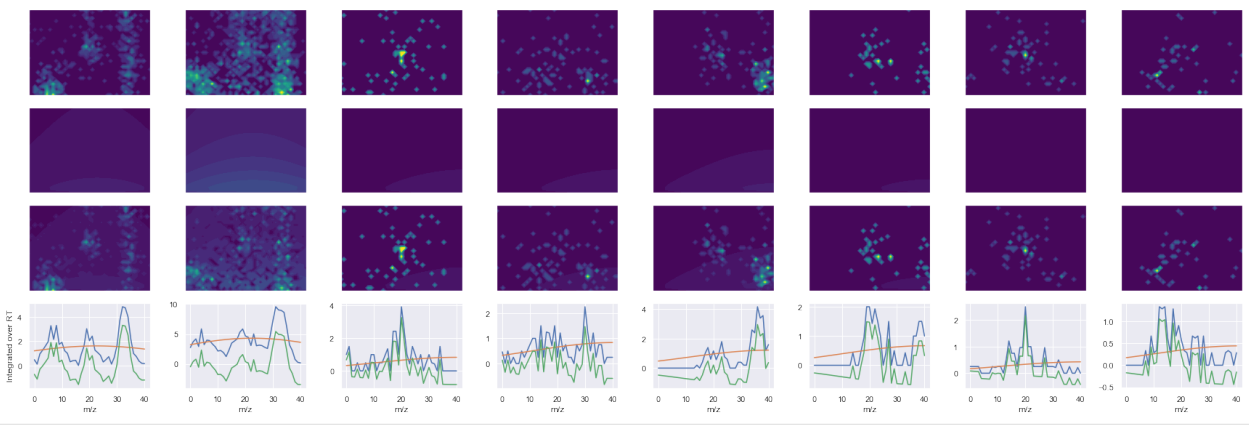

```
Raw MS1 & MS2: 2748.484368356817 66.35065408147446
Fit MS1 & MS2: 2799.4163691023036 45.93175784643054
[('QNVPIIR', 2, 2799.4163691023036, 45.93175784643054)]
---------------------------------
ms2_name = 'ms2-004'
peptide, charge, rank, rt, pre = 'REDFLR', 2, 1, 1997.44, 418.22467041
pro = [435.27142334, 550.29840088, 661.3303833]
```



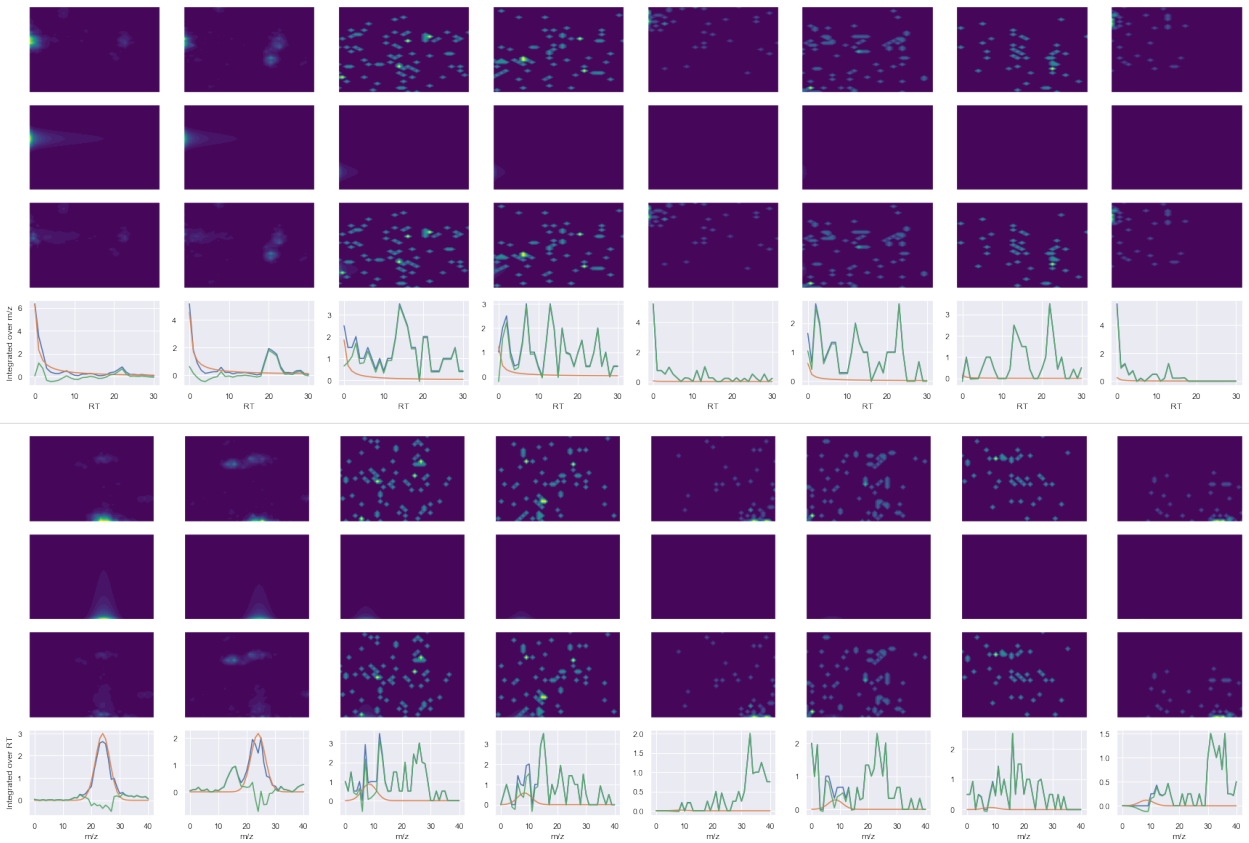



```
Raw MS1 & MS2: 1371.4563234616016 75.66820364750609
Fit MS1 & MS2: 148.42967820914942 24.819024099327837
[('REDFLR', 2, 148.42967820914942, 24.819024099327837)]
********************************************************************************
--------------------------------
ms2_name = 'ms2-049'
peptide, charge, rank, rt, pre = 'HGEPEEDIVGLQAFQER', 3, 1, 2662.94, 651.98156738
pro = [579.28851318, 778.38421631, 794.29516602, 907.37921143, 948.48974609, 1063.
↪4691162]
```





```
Raw MS1 & MS2: 2602.4031337189504 258.6497463384277
Fit MS1 & MS2: 1916.315200109147 267.09342144412926
[('HGEPEEDIVGLQAFQER', 3, 1916.315200109147, 267.09342144412926)]
**********************************************************************************
----------------------------------
ms2_name = 'ms2-050'
peptide, charge, rank, rt, pre = 'NADMQEELIASLEEQLK', 3, 1, 4136.85, 654.3225708
pro = [517.29803467, 759.42468262, 818.29852295, 846.45672607, 917.49383545, 931.
↪38256836]
```

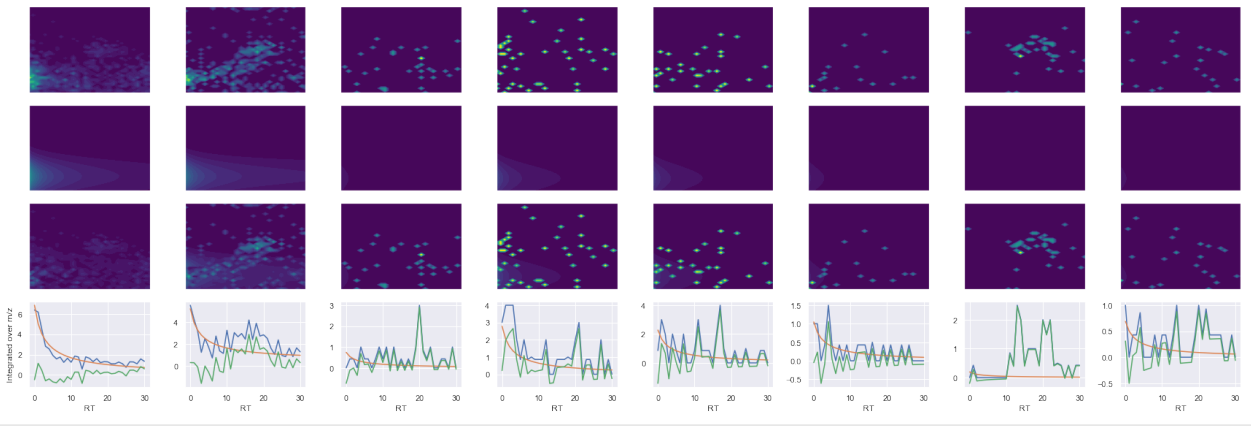


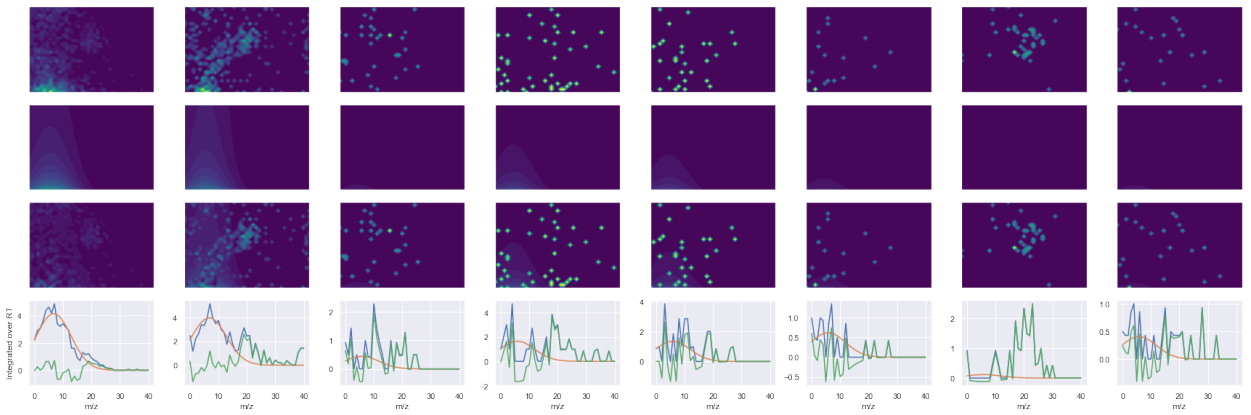
```
Raw MS1 & MS2: 11887.751376622578 206.13587926308224
Fit MS1 & MS2: 10432.259282198298 14.468248025629837
[('NADMQEELIASLEEQLK', 3, 10432.259282198298, 14.468248025629837)]
**********************************************************************************
```

(continues on next page)

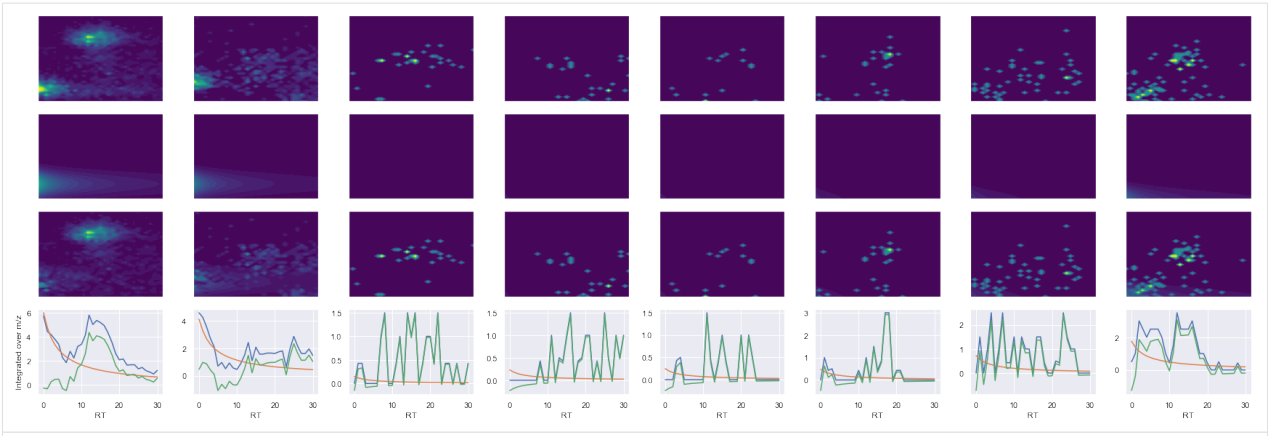
(continued from previous page)

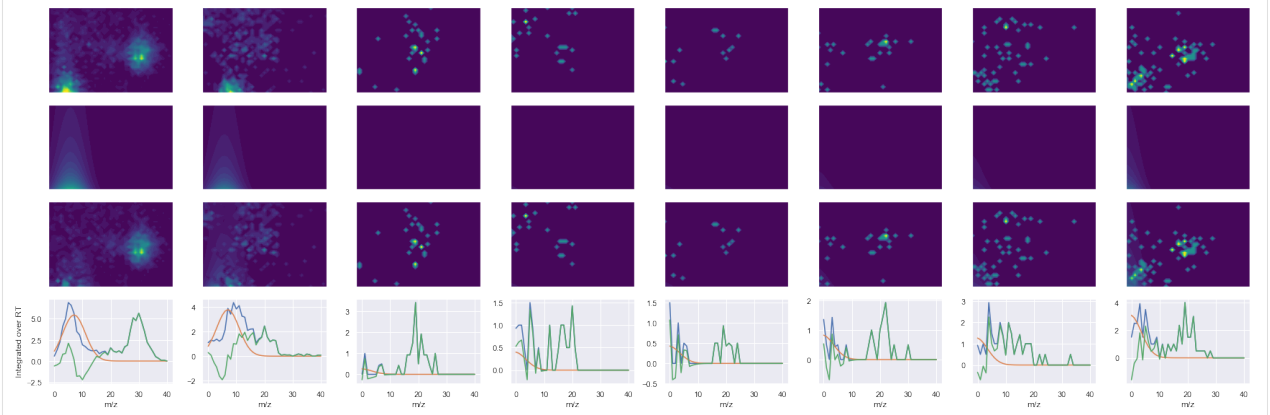
```
---------------------------------
ms2_name = 'ms2-057'
peptide, charge, rank, rt, pre = 'VNITPAEVGVLVGK', 2, 1, 3101.9, 698.41394043
pro = [416.28674316, 525.30310059, 572.37664795, 871.52471924, 968.57751465, 1069.
↪6252441]
```



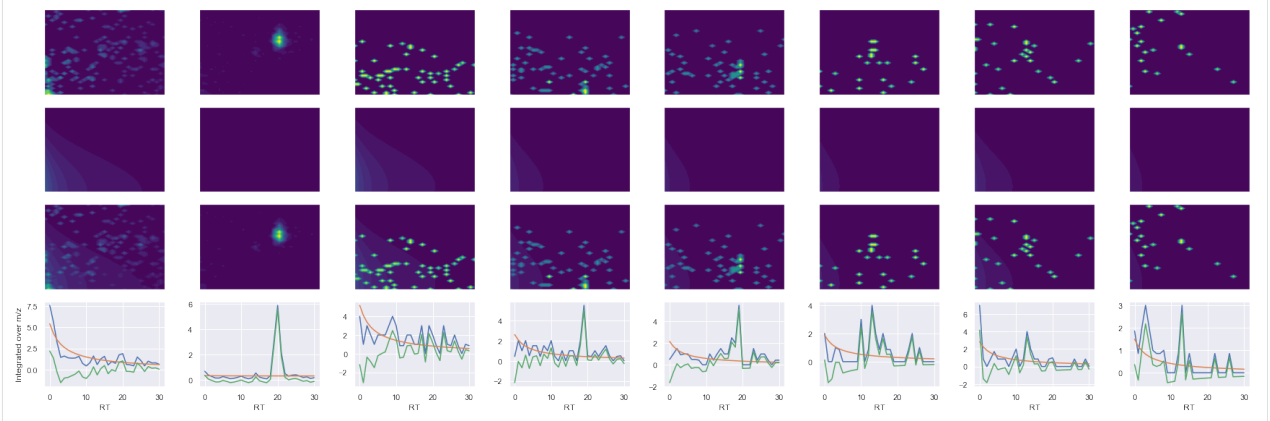


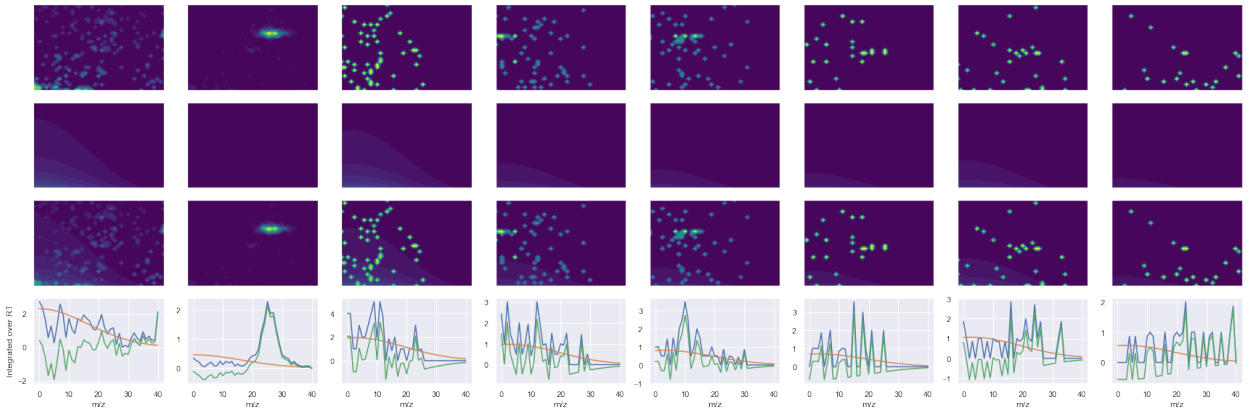

```
Raw MS1 & MS2: 1822.2111671039456 88.9707300880708
Fit MS1 & MS2: 1422.579423536166 40.31177107899468
[('VNITPAEVGVLVGK', 2, 1422.579423536166, 40.31177107899468)]
**************************************************************************
---------------------------------
ms2_name = 'ms2-064'
peptide, charge, rank, rt, pre = 'EAGNINQSLLTLGR', 2, 1, 2626.65, 743.40460205
pro = [446.27215576, 559.35620117, 672.44030762, 759.47229004, 887.53088379, 1001.
↪5737915]
```



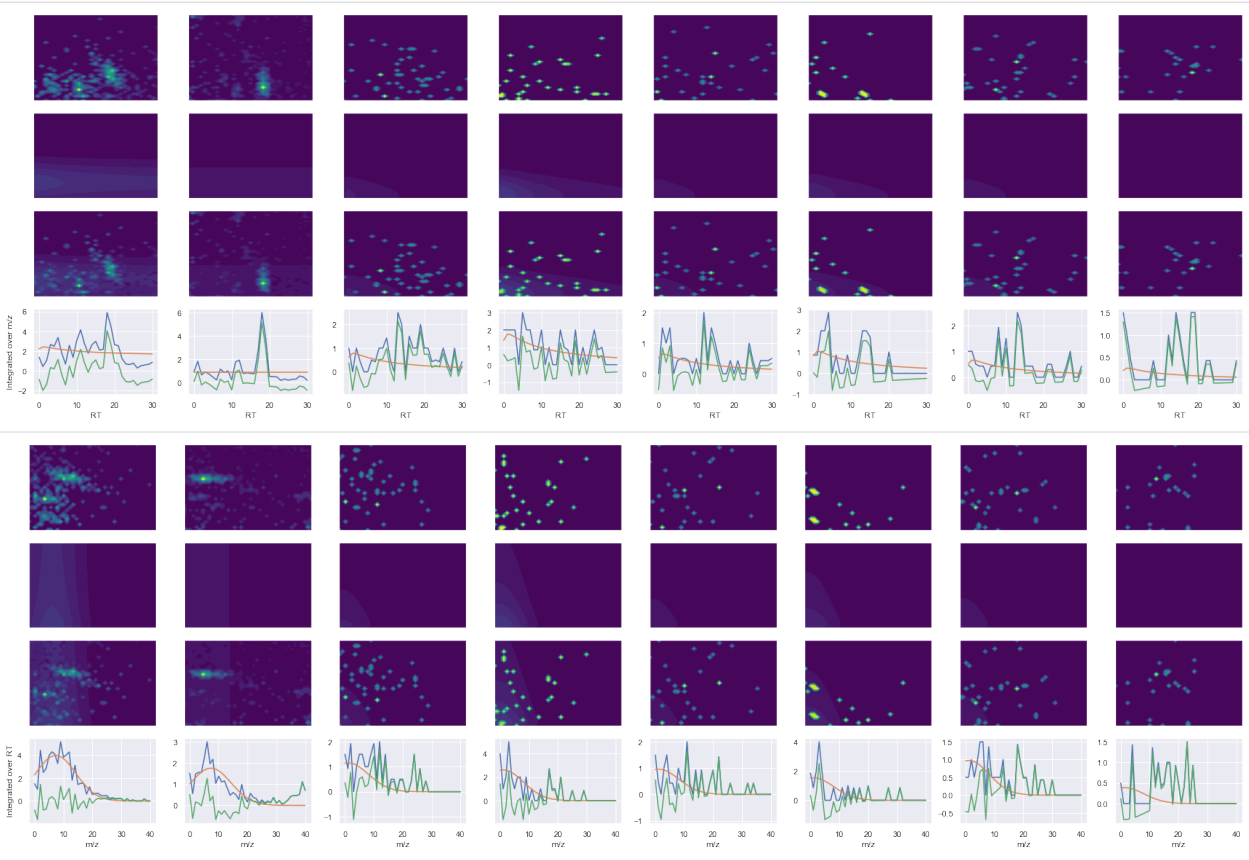



```
Raw MS1 & MS2: 5431.12478549956 124.70365258035318
Fit MS1 & MS2: 3157.8429424718797 36.32079911217963
[('EAGNINQSLLTLGR', 2, 3157.8429424718797, 36.32079911217963)]
**********************************************************************************
---------------------------------
ms2_name = 'ms2-089'
peptide, charge, rank, rt, pre = 'VEEASPGRPSSVDTLLSPTALIDSILR', 3, 1, 4217.8, 941.
↪84143066
pro = [488.31912231, 603.34606934, 716.43011475, 1098.6517334, 1185.6837158, 1311.
↪6175537]
```





```
Raw MS1 & MS2: 2300.6600852753154 130.07829467804535
Fit MS1 & MS2: 699.0792733909628 98.30380772826076
[('VEEASPGRPSSVDTLLSPTALIDSILR', 3, 699.0792733909628, 98.30380772826076)]
********************************************************************************
--------------------------------
ms2_name = 'ms2-096'
peptide, charge, rank, rt, pre = 'SLLSIPNTDYIQLLSEIAK', 2, 1, 4206.23, 1059.5882568
pro = [660.39263916, 773.47674561, 901.53533936, 1177.6827393, 1393.7573242, 1604.
↪8530273]
```



```
Raw MS1 & MS2: 2132.882937447126 114.49477907659808
Fit MS1 & MS2: 228.34049788928704 70.15997919249725
[('SLLSIPNTDYIQLLSEIAK', 2, 228.34049788928704, 70.15997919249725)]
```

1.5.5 Build testing data for the C++ library

We already use data from the `DIA_TEST_DATA_REPO` in our unit testing, so use the python implementation to build test data for the C++ tests.

[21]:
```
peptides_charges_and_rt = [
    ('DAVTPADFSEWSK', 2, 2609.69),
    ('EATFGVDESANK', 2, 830.17),
    ('GDLDAASYYAPVR', 2, 1882.61),
    # ms2-025
    ('ELHINLIPNKQDR', 3, 1842.77),
    ('KSDIDEIVLVGGSTR', 3, 2231.21),
    # ms2-052
    ('LAAEESVGTMGNR', 2, 826.85),
    ('TPVISGGPYYER', 2, 1467.61),
    # ms2-056
    ('TLEEDEEELFK', 2, 2659.49),
    ('VQVALEELQDLK', 2, 3044.61),
    # ms2-066
    ('GILAAEESVGTMGNR', 2, 1962.29),
    ('SPFTVGVAAPLDLSK', 2, 3263.73),
]
peptides = [p[0] for p in peptides_charges_and_rt]
srl_fname = base_dir + '/ProCan90/srl/hek_srl.OpenSwath.iRT.tsv'
tof_fname = base_dir + '/ProCan90/tof/ProCan90-M03-01.iRT.tof'
```

[22]:
```
plotter = ToffeeFragmentsPlotter(tof_fname)
```

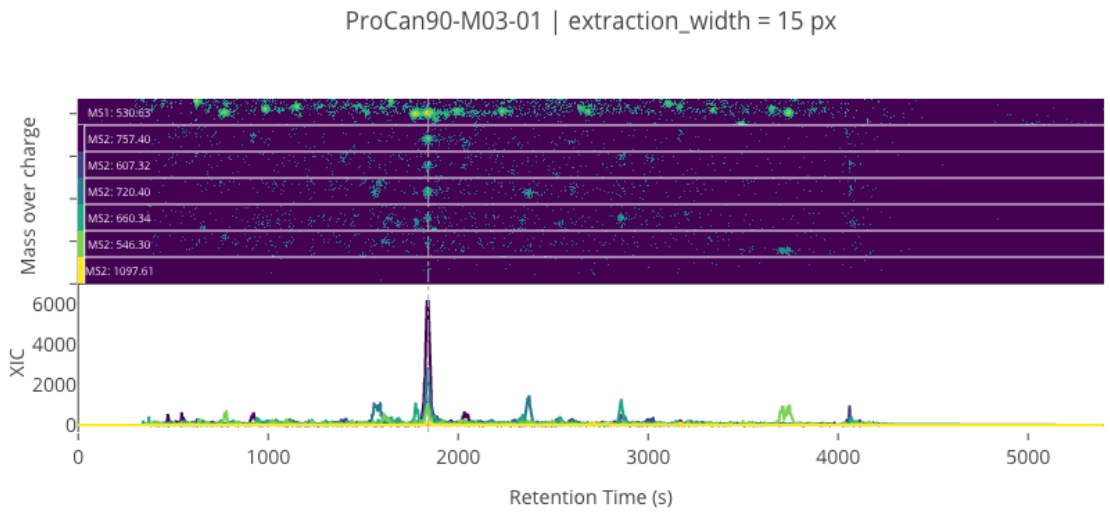
[23]:
```
srl = pd.read_csv(srl_fname, sep='\t')
srl = srl.loc[srl.PeptideSequence.isin(peptides)]

ms2_map = plotter.extractor.swath_run.mapPrecursorsToMS2Names(srl.PrecursorMz.
↪unique(), 0.0, 1.0)
srl['MS2Name'] = srl.PrecursorMz.map(ms2_map)

n_isotopes = 0
precursors, products = calculate_isotope_mz(
    precursor_and_product_df=srl,
    n_isotopes=n_isotopes,
    sort_by_intensity=True,
)

all_srl = srl.copy()
for ms2_name in sorted(all_srl.MS2Name.unique()):
    print('*' * 80)
    srl = all_srl.loc[all_srl.MS2Name == ms2_name]
    for peptide, charge, rt in [p for p in peptides_charges_and_rt if p[0] in srl.
↪PeptideSequence.unique()]:
        print('-' * 50)
        pqp = srl.loc[srl.PeptideSequence == peptide, 'TransitionGroupId'].unique()[0]
        ms2_name = srl.loc[srl.PeptideSequence == peptide, 'MS2Name'].unique()[0]
        pre = precursors.loc[precursors.Id == pqp, 'IsotopeMz'].tolist()
        pro = products.loc[products.GroupId == pqp, 'IsotopeMz'].tolist()[:6]
        assert len(pre) > 0
        assert len(pro) > 0
        print(f'ms2_name = \'{ms2_name}\'')
        print(f'peptide, charge, rt, pre = \'{peptide}\', {charge}, {rt}, {pre[0]}')
```

(continues on next page)

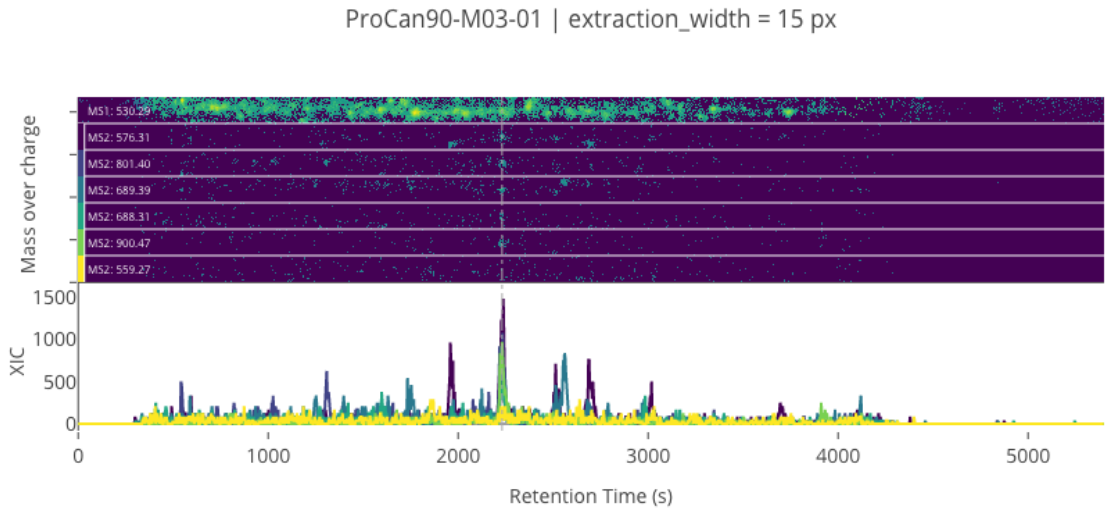
(continued from previous page)

```
        print(f'pro = {pro}')
        output_fname = '/tmp/fig.png'
        fig = plotter.create_plotly_figure(
            precursor_mz_list=pre,
            product_mz_list=pro,
            psm_name='{} | extraction_width = {} {}'.format(
                plotter.tof_fname.split('/')[-1].split('.')[0],
                plotter.extractor.extraction_width,
                'ppm' if plotter.extractor.use_ppm else 'px',
            ),
            number_of_isotopes=n_isotopes,
            log_heatmap=True,
            normalise_per_fragment=False,
            retention_time_line=rt,
            output_fname=output_fname,
        )
        display(Image(output_fname))
```

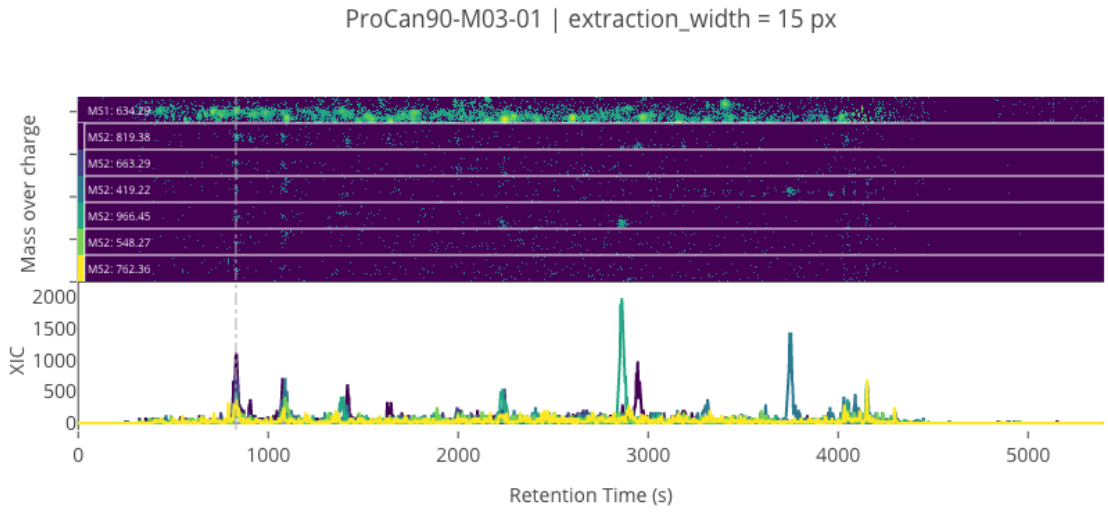
```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-025'
peptide, charge, rt, pre = 'ELHINLIPNKQDR', 3, 1842.77, 530.630126953125
pro = [757.395141601563, 607.31982421875, 720.403930664063, 660.342346191406, 546.
↪2994384765631, 1097.60620117188]
```



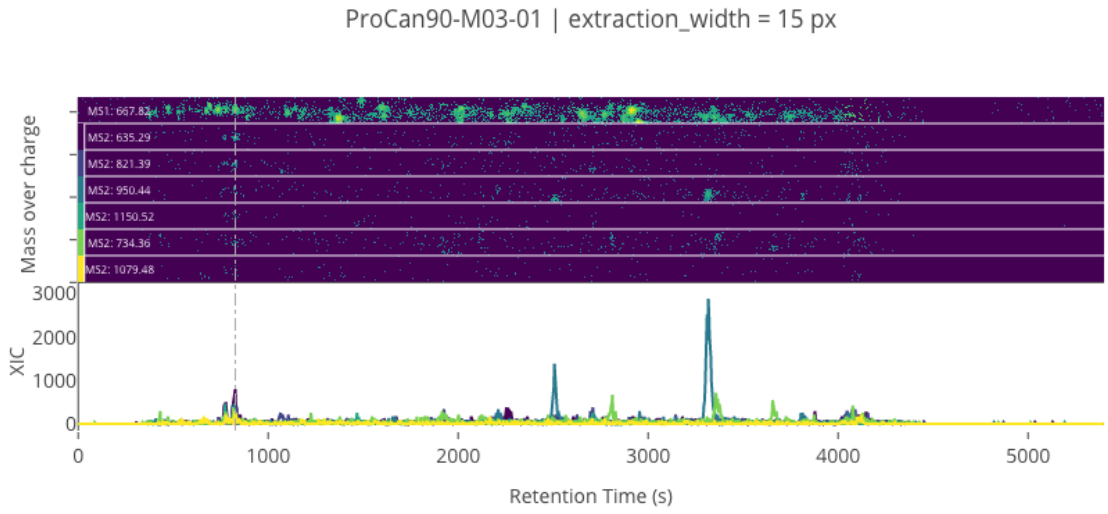

```
---------------------------------
ms2_name = 'ms2-025'
peptide, charge, rt, pre = 'KSDIDEIVLVGGSTR', 3, 2231.21, 530.28955078125
pro = [576.309997558594, 801.3988647460941, 689.39404296875, 688.314819335938, 900.
↪46728515625, 559.272216796875]
```



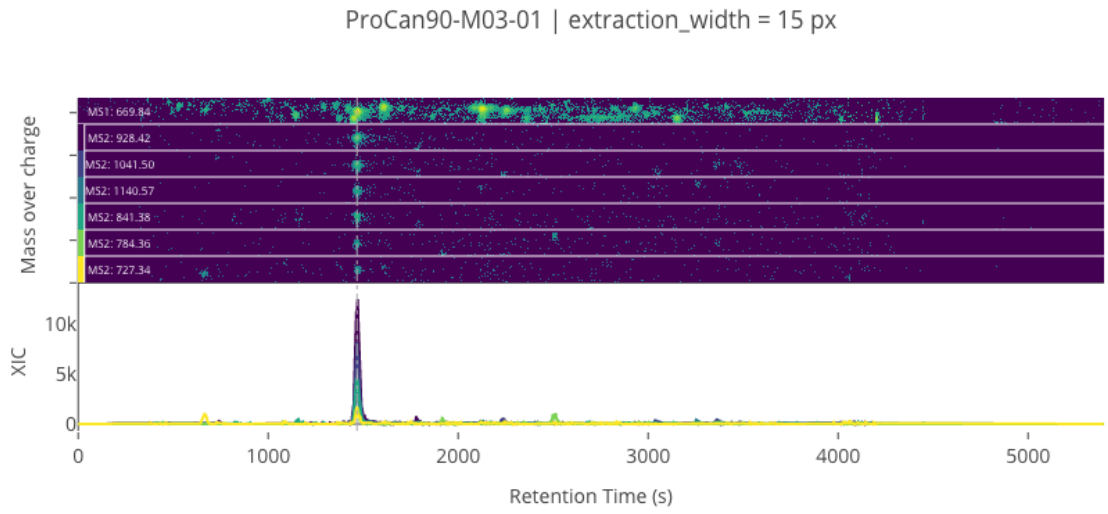

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-046'
peptide, charge, rt, pre = 'EATFGVDESANK', 2, 830.17, 634.293701171875
pro = [819.38427734375, 663.29443359375, 419.22488403320295, 966.4526977539059, 548.
↪2674560546881, 762.3628540039059]
```



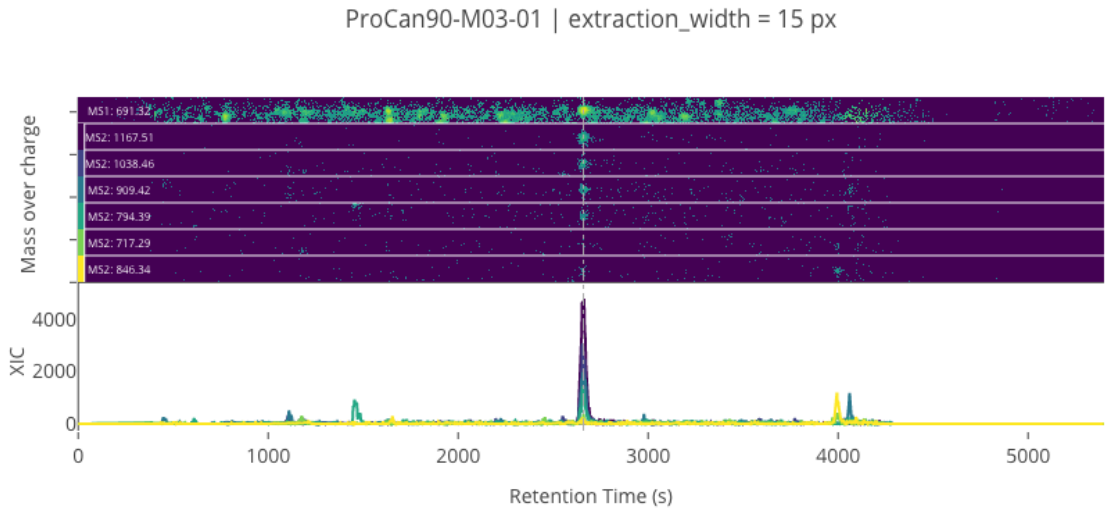

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-052'
peptide, charge, rt, pre = 'LAAEESVGTMGNR', 2, 826.85, 667.822082519531
pro = [635.29296875, 821.393432617188, 950.43603515625, 1150.51574707031, 734.
↪3613891601559, 1079.47863769531]
```



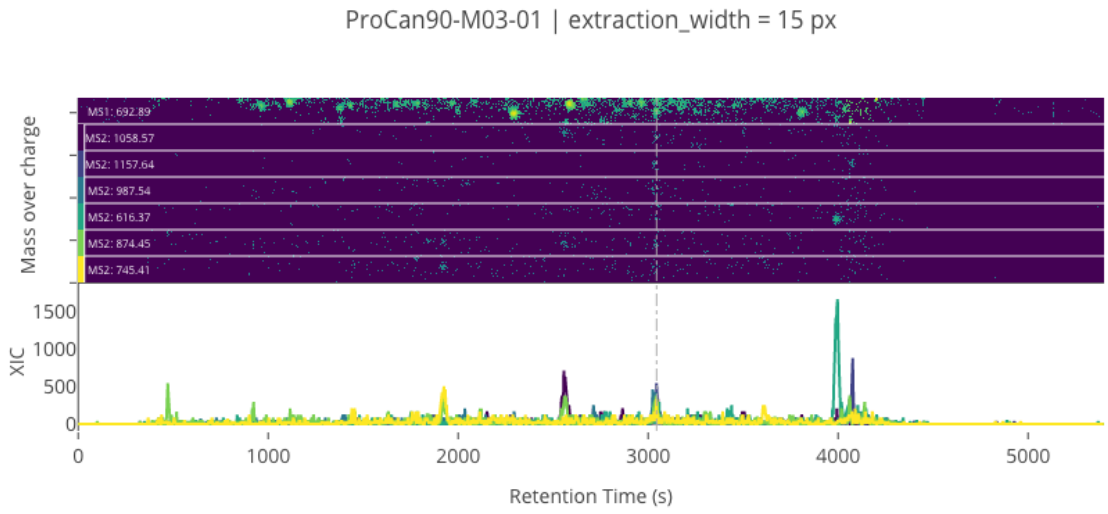

```
---------------------------------
ms2_name = 'ms2-052'
peptide, charge, rt, pre = 'TPVISGGPYYER', 2, 1467.61, 669.838073730469
pro = [928.415893554688, 1041.5, 1140.568359375, 841.383911132813, 784.362426757813,␣
↪727.340942382813]
```



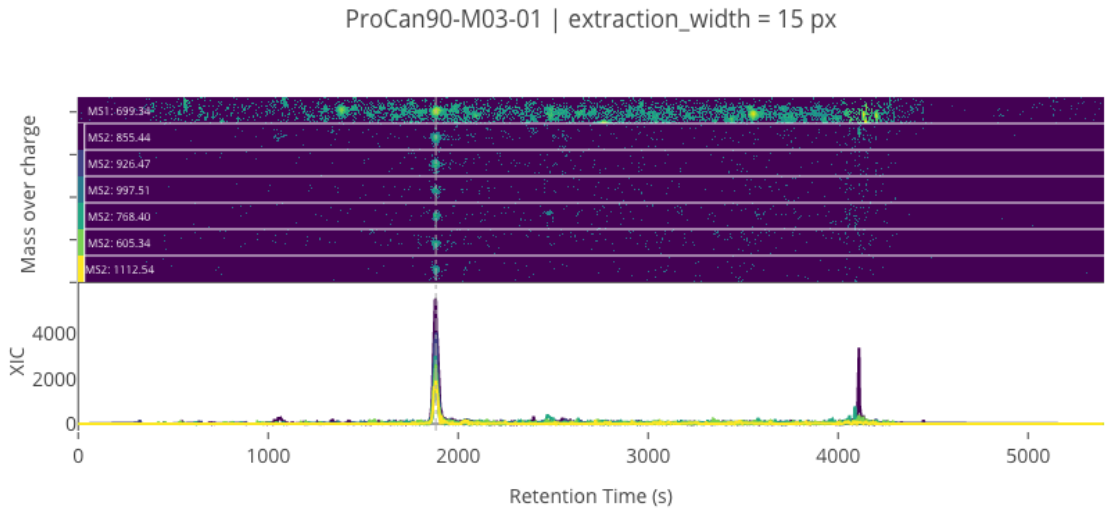

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-056'
peptide, charge, rt, pre = 'TLEEDEEELFK', 2, 2659.49, 691.3220825195309
pro = [1167.50524902344, 1038.46264648438, 909.4199829101559, 794.39306640625, 717.
↪2937622070309, 846.3363647460941]
```



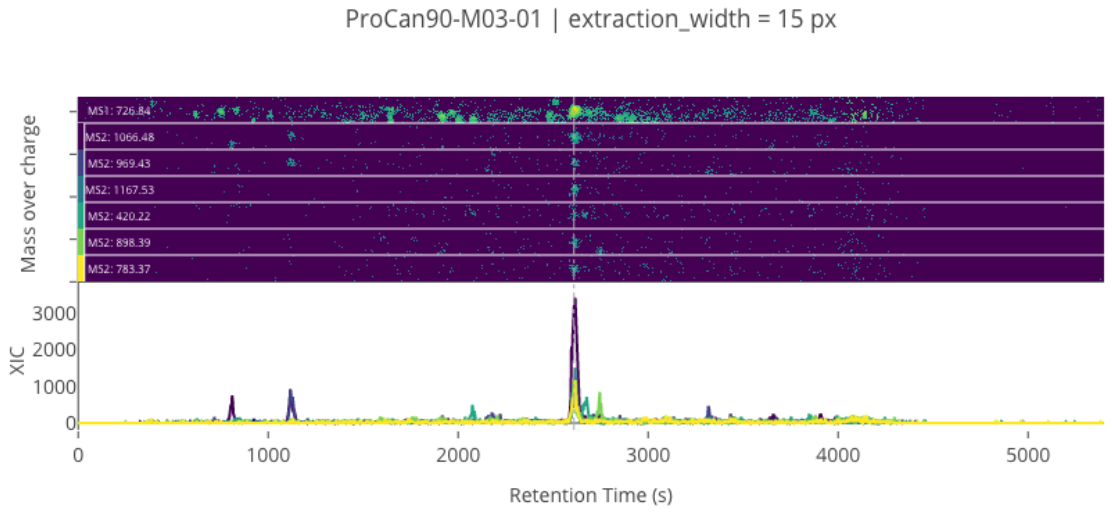

```
----------------------------------
ms2_name = 'ms2-056'
peptide, charge, rt, pre = 'VQVALEELQDLK', 2, 3044.61, 692.8877563476559
pro = [1058.57287597656, 1157.64123535156, 987.5357055664059, 616.366455078125, 874.
↪45166015625, 745.409057617188]
```



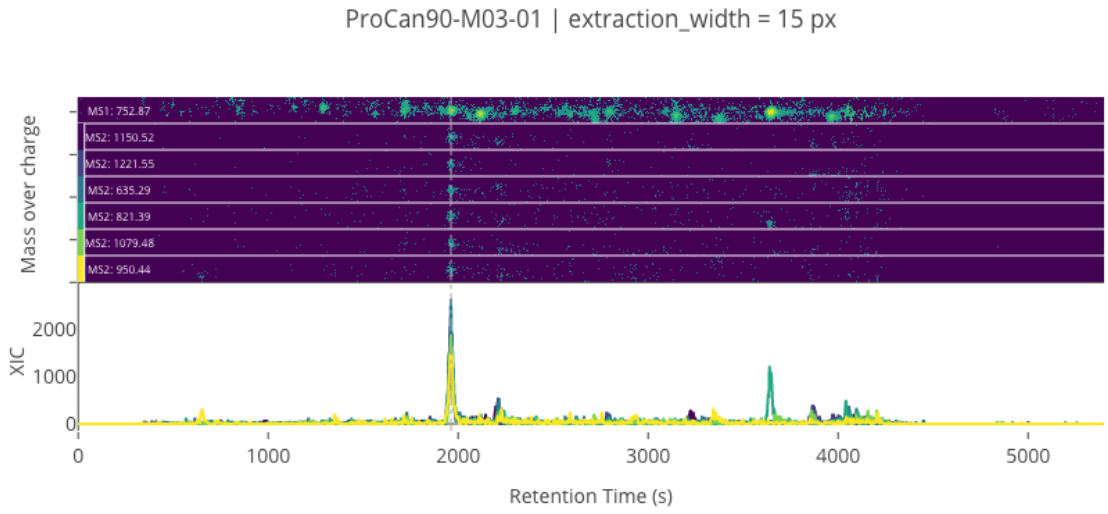

```
**********************************************************************************
----------------------------------
ms2_name = 'ms2-057'
peptide, charge, rt, pre = 'GDLDAASYYAPVR', 2, 1882.61, 699.3384399414059
pro = [855.435913085938, 926.473022460938, 997.510131835938, 768.403930664063, 605.
↪340576171875, 1112.537109375]
```



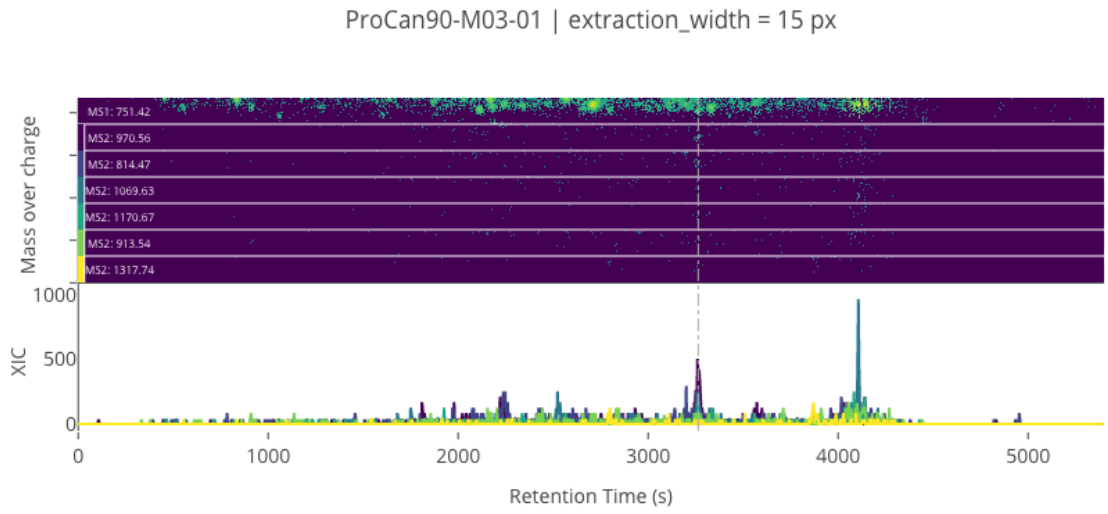

```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-062'
peptide, charge, rt, pre = 'DAVTPADFSEWSK', 2, 2609.69, 726.835693359375
pro = [1066.48400878906, 969.4312133789059, 1167.53173828125, 420.22415161132795, 898.
↪3941040039059, 783.3671875]
```




```
**********************************************************************************
---------------------------------
ms2_name = 'ms2-066'
peptide, charge, rt, pre = 'GILAAEESVGTMGNR', 2, 1962.29, 752.8748168945309
pro = [1150.51574707031, 1221.55285644531, 635.29296875, 821.393432617188, 1079.
↪47863769531, 950.43603515625]
```




```
----------------------------------
ms2_name = 'ms2-066'
peptide, charge, rt, pre = 'SPFTVGVAAPLDLSK', 2, 3263.73, 751.4166870117191
pro = [970.556762695313, 814.466918945313, 1069.62524414063, 1170.6728515625, 913.
↪5353393554691, 1317.74133300781]
```



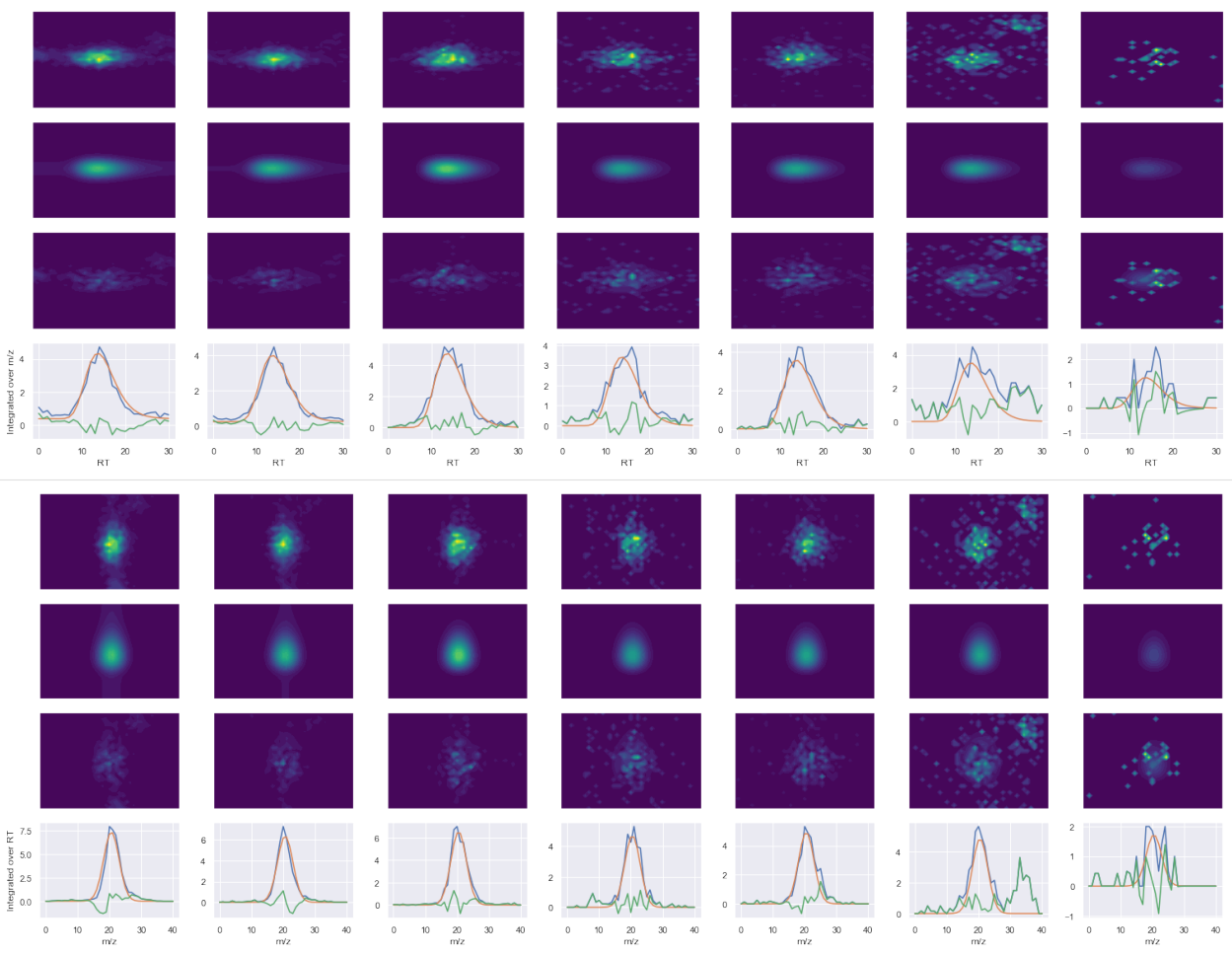
[24]:
```
operator = Requant(tof_fname, debug_plots=True)

for ms2_name in sorted(all_srl.MS2Name.unique()):
    print('*' * 80)
    srl = all_srl.loc[all_srl.MS2Name == ms2_name]
    for peptide, charge, rt in [p for p in peptides_charges_and_rt if p[0] in srl.
↪PeptideSequence.unique()]:
        print('-' * 50)
        pqp = srl.loc[srl.PeptideSequence == peptide, 'TransitionGroupId'].unique()[0]
        charge = srl.loc[srl.PeptideSequence == peptide, 'PrecursorCharge'].
↪unique()[0]
```

(continues on next page)

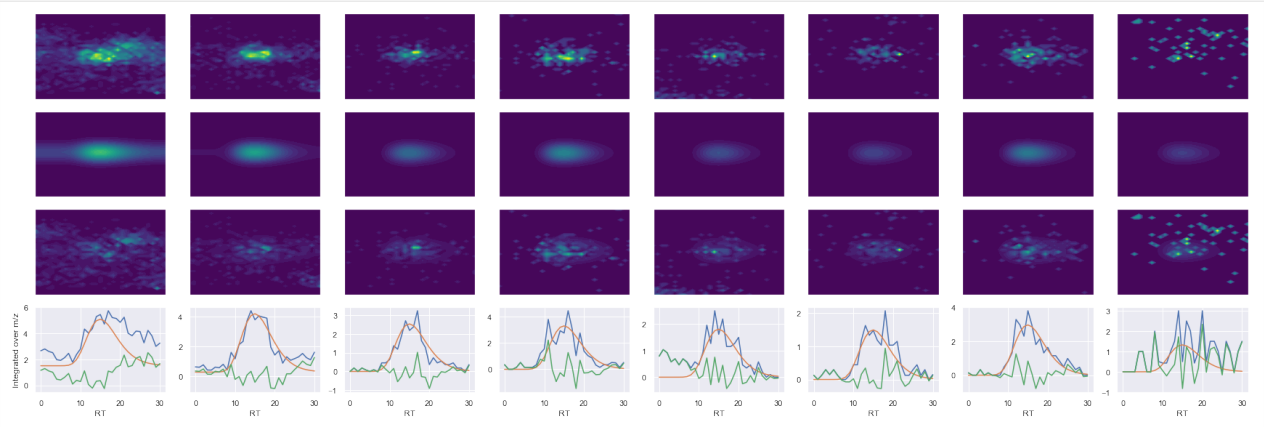
(continued from previous page)

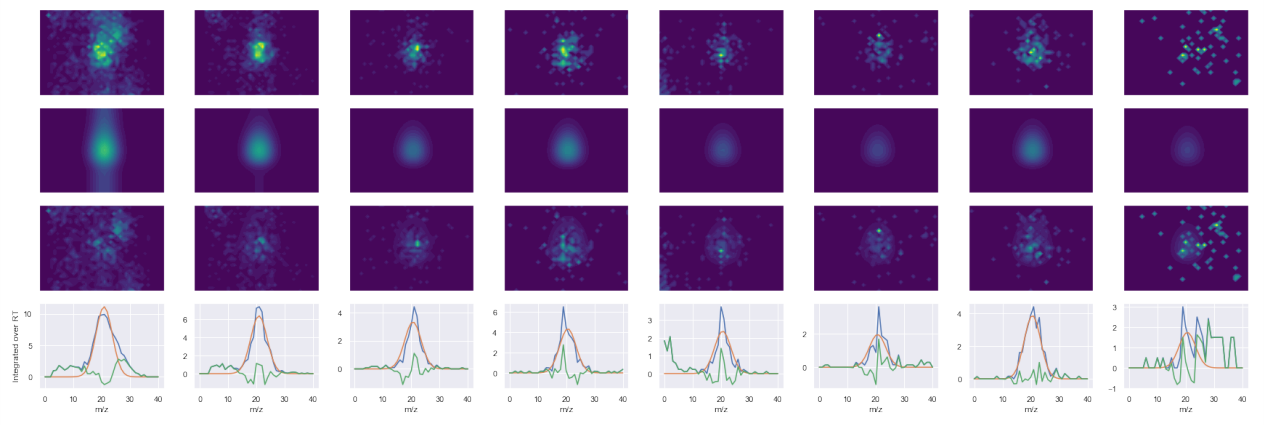
```
        ms2_name = srl.loc[srl.PeptideSequence == peptide, 'MS2Name'].unique()[0]
        rank = 1

        pre = precursors.loc[precursors.Id == pqp, 'IsotopeMz'].tolist()
        pro = products.loc[products.GroupId == pqp, 'IsotopeMz'].tolist()[:6]
        assert len(pre) > 0
        assert len(pro) > 0
        print(pqp, ms2_name)

        peak_group = requant.RequantPeakGroup(peptide, charge, rank, rt, pre[0], pro)
        print(operator.run(ms2_name, [peak_group]))
```

```
**********************************************************************************
---------------------------------
ELHINLIPNKQDR_3 ms2-025
546.2994384765631 (31, 38) != (31, 41)
```

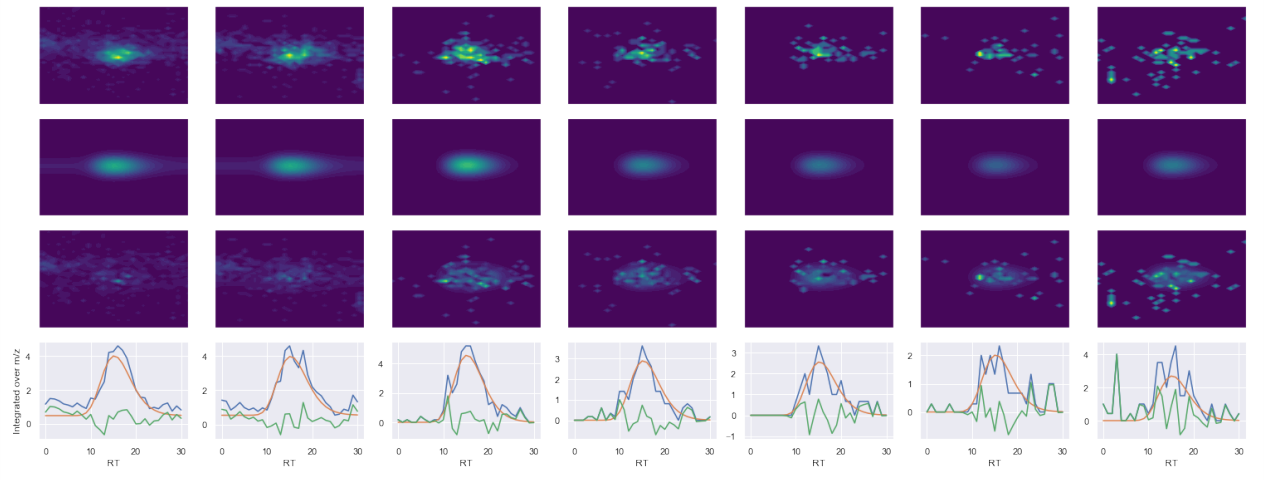


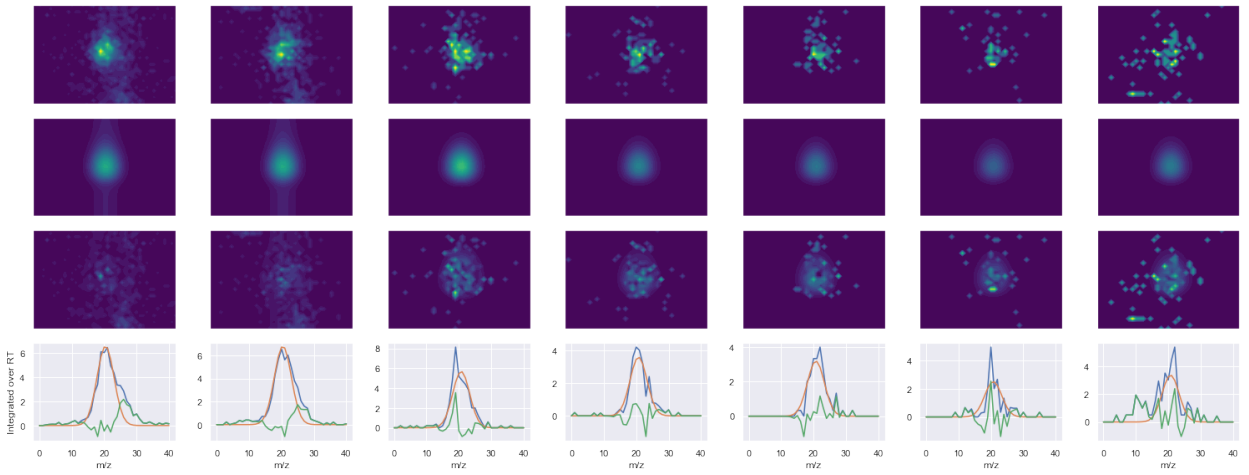

```
Raw MS1 & MS2: 7971.382612753305 1360.6360110167914
Fit MS1 & MS2: 5936.0447804902815 1105.2600737423677
[('ELHINLIPNKQDR', 3, 5936.0447804902815, 1105.2600737423677)]
---------------------------------
KSDIDEIVLVGGSTR_3 ms2-025
```



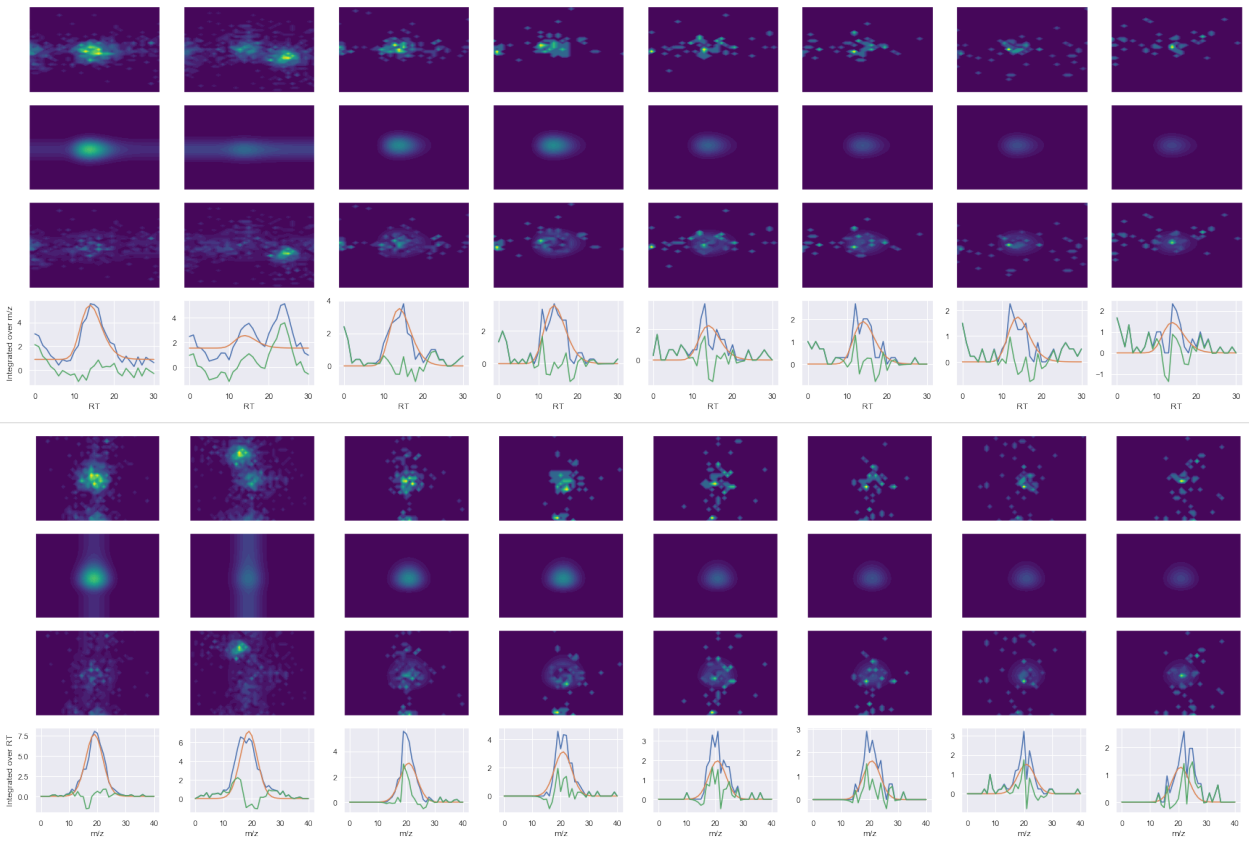



```
Raw MS1 & MS2: 3663.5420920012957 548.9509062646167
Fit MS1 & MS2: 1565.341063713942 467.98236860566055
[('KSDIDEIVLVGGSTR', 3, 1565.341063713942, 467.98236860566055)]
********************************************************************************
--------------------------------
EATFGVDESANK_2 ms2-046
419.22488403320295 (31, 36) != (31, 41)
```

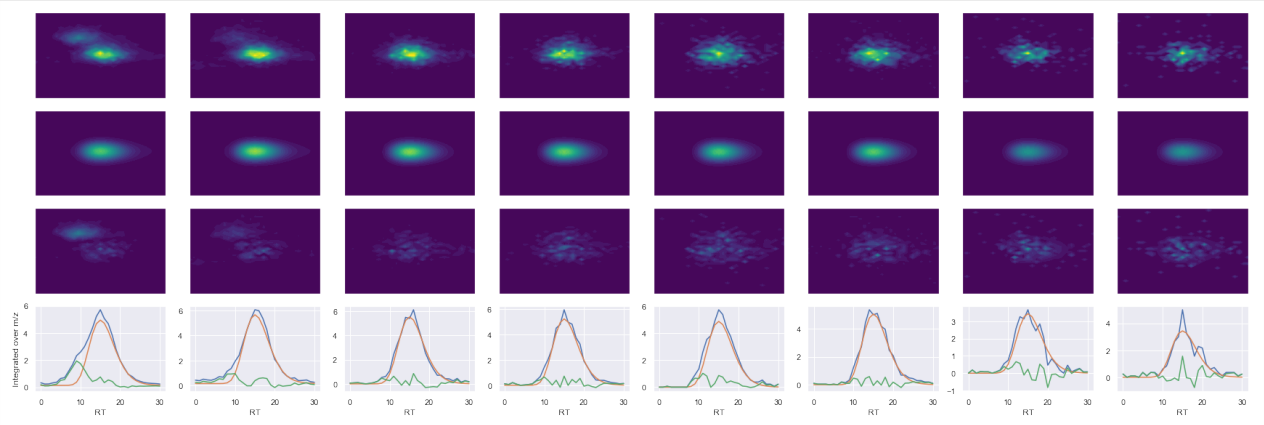


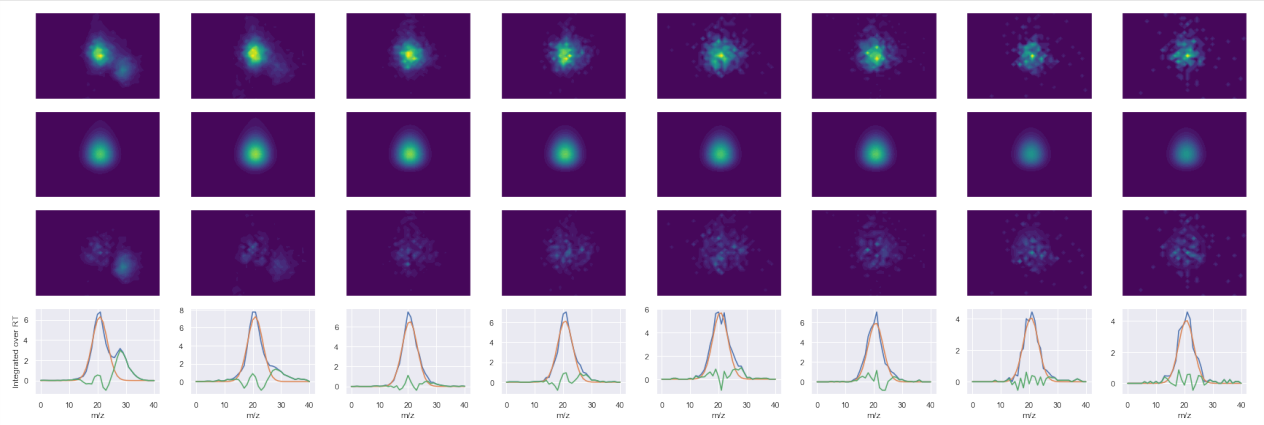



```
Raw MS1 & MS2: 2727.821488812674 294.9427344606671
Fit MS1 & MS2: 1408.4605688391075 249.52289699122136
[('EATFGVDESANK', 2, 1408.4605688391075, 249.52289699122136)]
**********************************************************************************
---------------------------------
LAAEESVGTMGNR_2 ms2-052
```

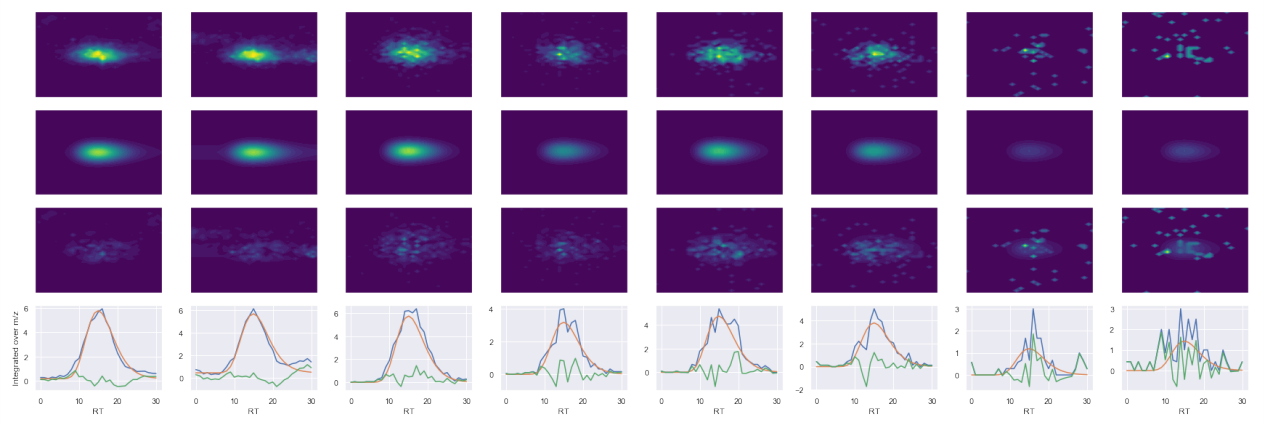


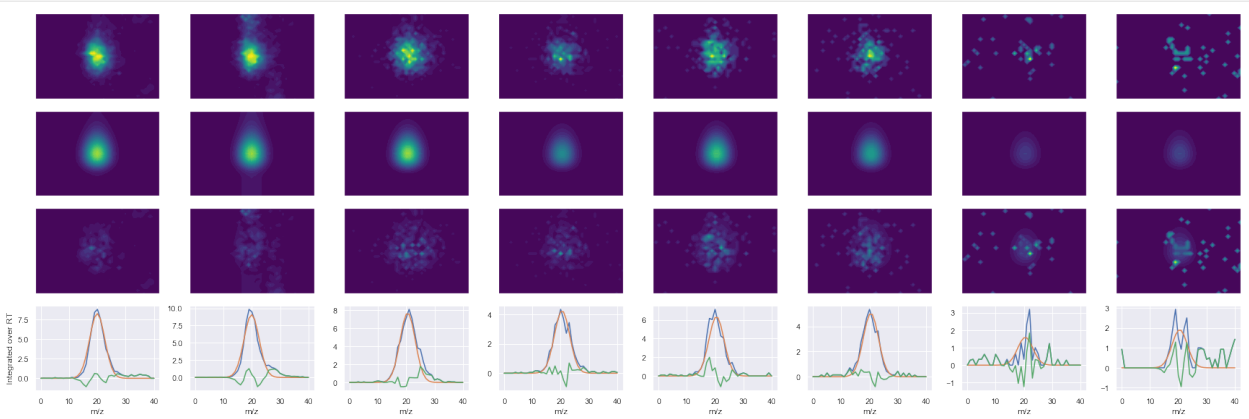

```
Raw MS1 & MS2: 2402.871796360551 274.4929809051114
Fit MS1 & MS2: 640.5041539081958 193.0167084570027
[('LAAEESVGTMGNR', 2, 640.5041539081958, 193.0167084570027)]
---------------------------------
TPVISGGPYYER_2 ms2-052
```



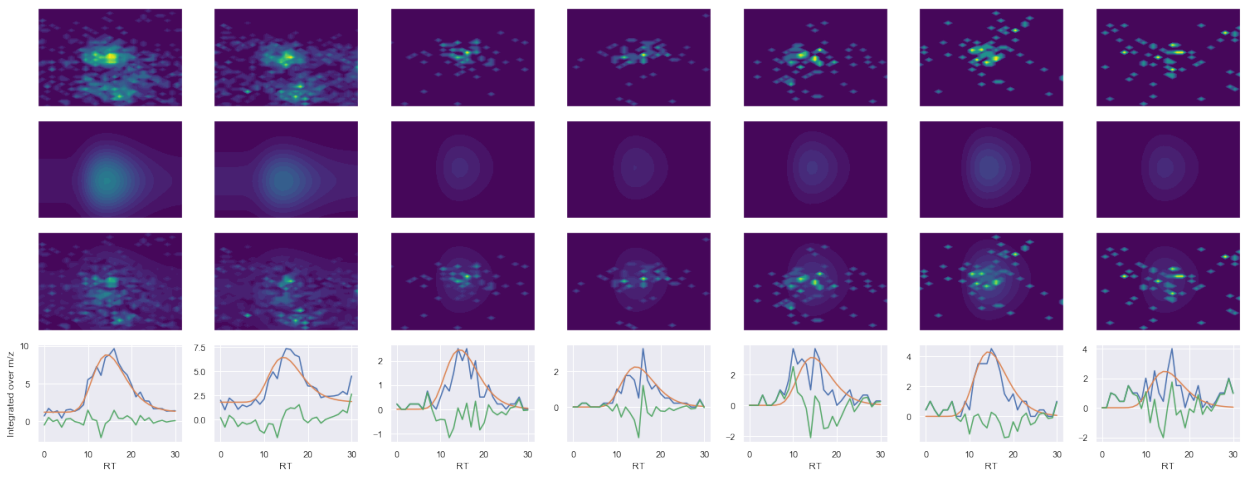


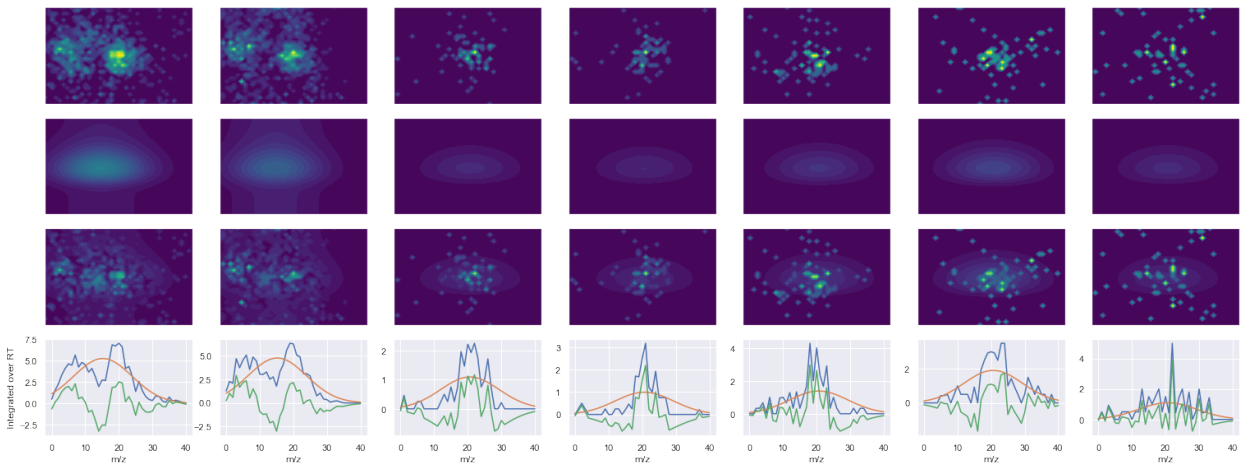

```
Raw MS1 & MS2: 18987.131273955467 3611.313791924786
Fit MS1 & MS2: 13335.611364021508 3232.3448694321823
[('TPVISGGPYYER', 2, 13335.611364021508, 3232.3448694321823)]
**********************************************************************************
---------------------------------
TLEEDEEELFK_2 ms2-056
```






```
Raw MS1 & MS2: 14489.968375957485 1537.2048833212073
Fit MS1 & MS2: 11542.81942725199 1334.0724131818538
[('TLEEDEEELFK', 2, 11542.81942725199, 1334.0724131818538)]
---------------------------------
VQVALEELQDLK_2 ms2-056
616.366455078125 (31, 40) != (31, 41)
```



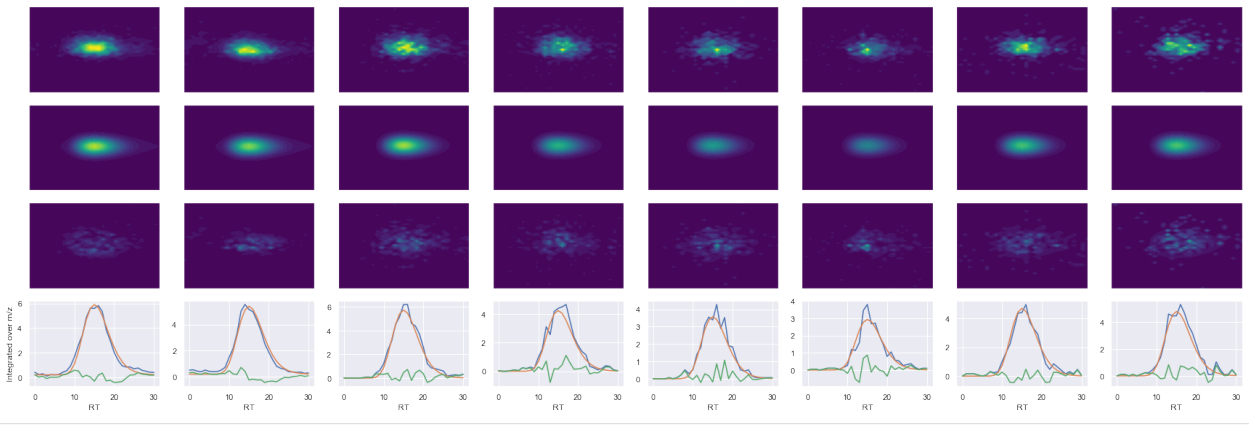


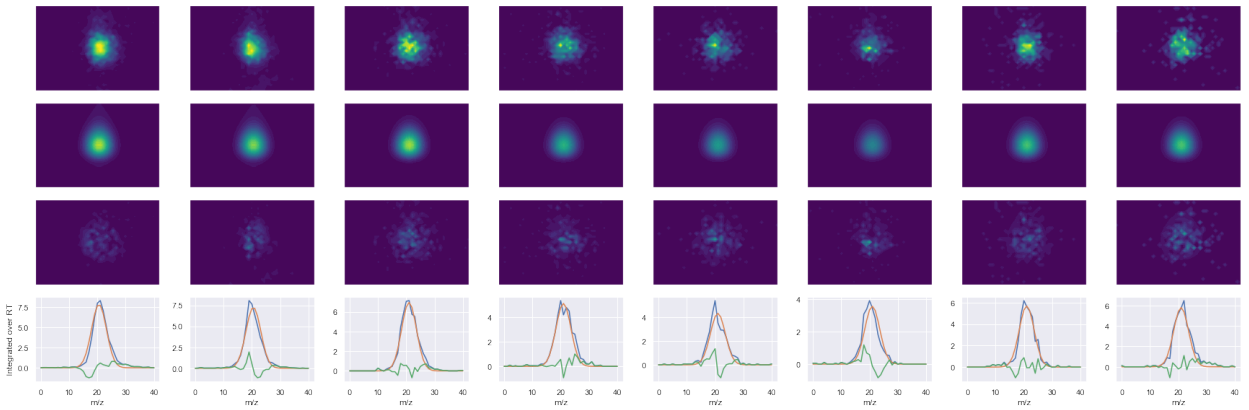

```
Raw MS1 & MS2: 2704.855085515457 233.11135740071734
Fit MS1 & MS2: 1448.0969122618142 242.93041021698042
[('VQVALEELQDLK', 2, 1448.0969122618142, 242.93041021698042)]
```

(continues on next page)

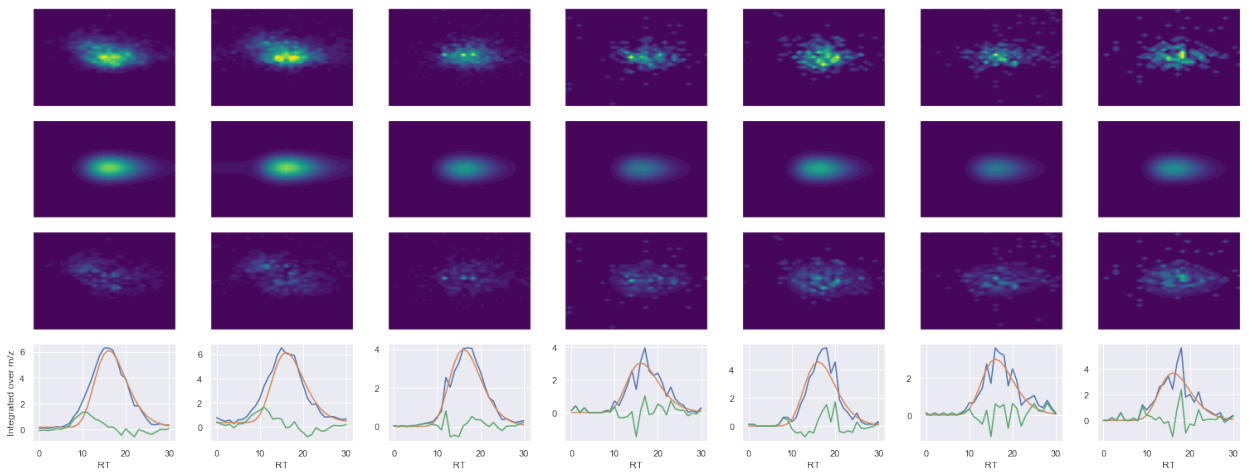
(continued from previous page)

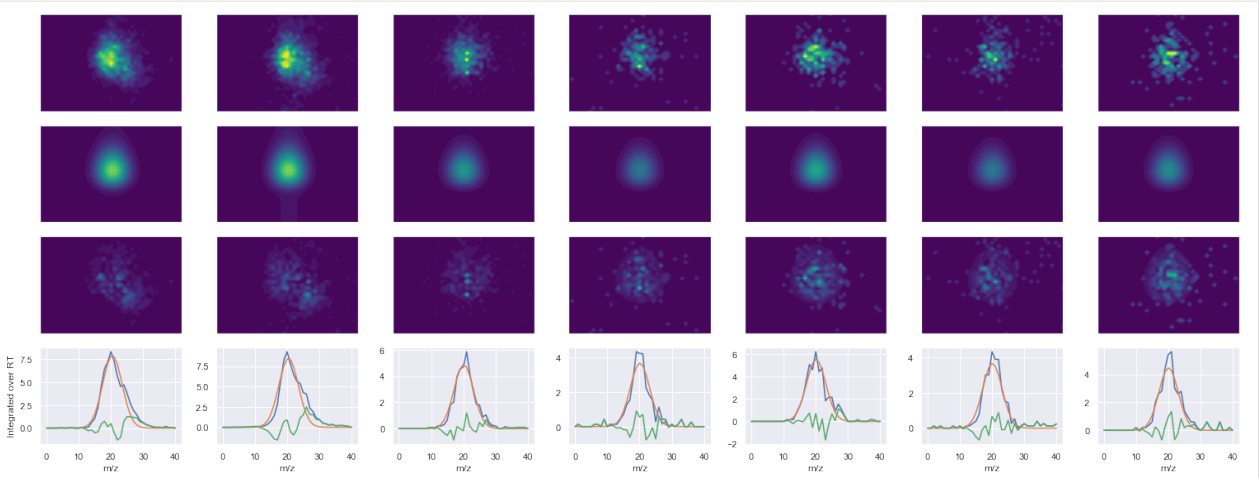
```
**********************************************************************************
--------------------------------
GDLDAASYYAPVR_2 ms2-057
```



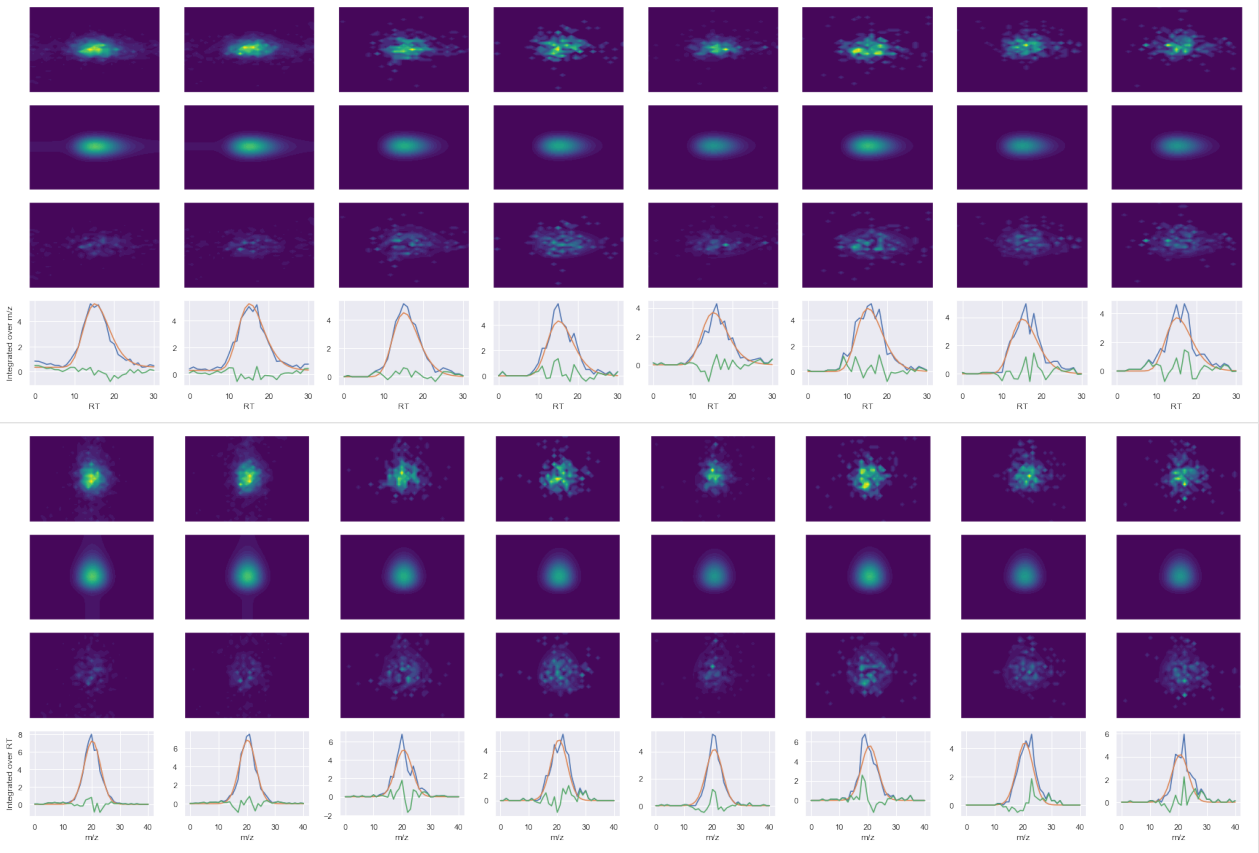



```
Raw MS1 & MS2: 9132.382213427156 2131.9018901521886
Fit MS1 & MS2: 7664.530453314555 1927.7800850246947
[('GDLDAASYYAPVR', 2, 7664.530453314555, 1927.7800850246947)]
**********************************************************************************
--------------------------------
DAVTPADFSEWSK_2 ms2-062
420.22415161132795 (31, 36) != (31, 41)
```

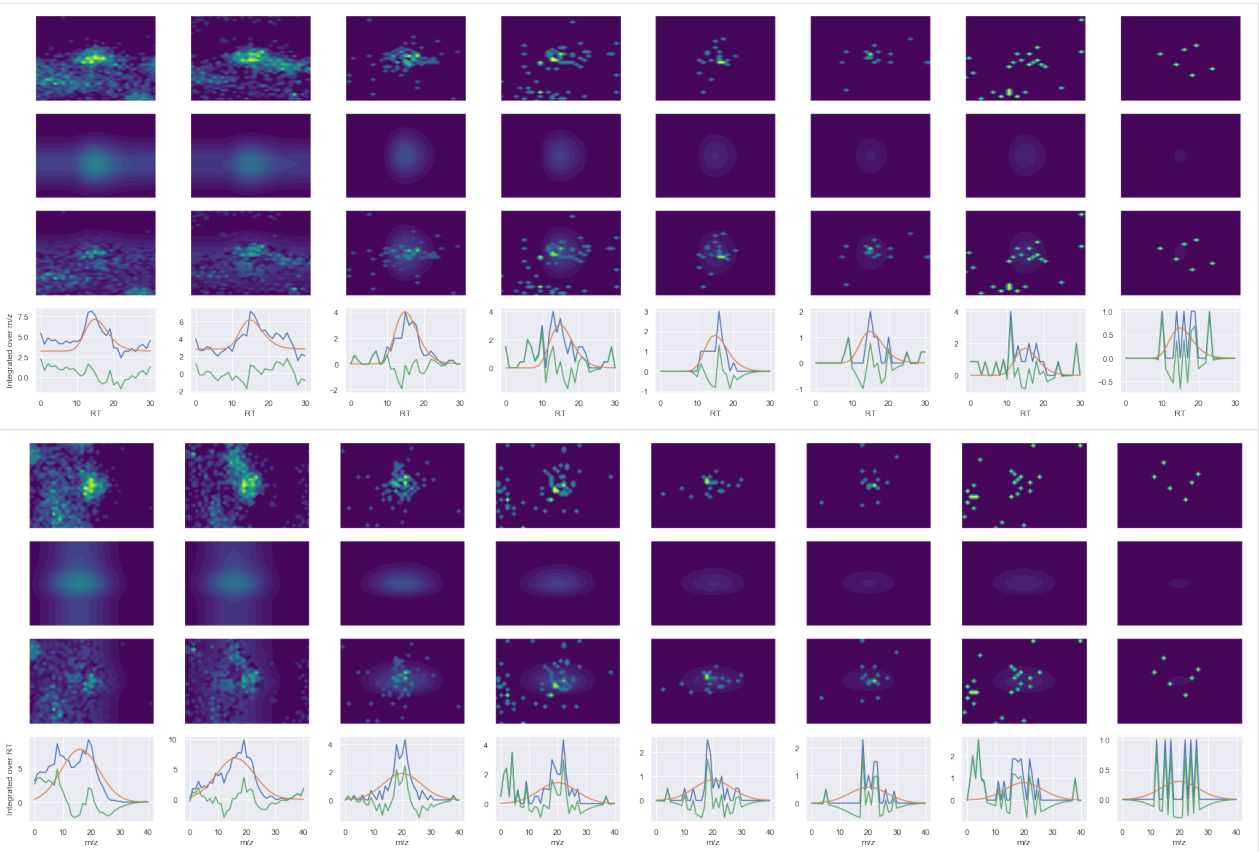





```
Raw MS1 & MS2: 10443.635559962993 1055.7539806959778
Fit MS1 & MS2: 8230.962938048931 991.0828664359409
[('DAVTPADFSEWSK', 2, 8230.962938048931, 991.0828664359409)]
**********************************************************************************
--------------------------------
GILAAEESVGTMGNR_2 ms2-066
```




```
Raw MS1 & MS2: 6142.502002452005 1221.8592417017833
Fit MS1 & MS2: 4804.03914396974 1101.5187370787942
[('GILAAEESVGTMGNR', 2, 4804.03914396974, 1101.5187370787942)]
--------------------------------
SPFTVGVAAPLDLSK_2 ms2-066
```




```
Raw MS1 & MS2: 2656.414106639162 136.0974727587502
Fit MS1 & MS2: 580.3664099675456 126.41985619771064
[('SPFTVGVAAPLDLSK', 2, 580.3664099675456, 126.41985619771064)]
```

```
[ ]:
```

CHAPTER

TWO

TOFFEE C++ API

Contents

2.1 Creating a Toffee File

The following API documentation gives a flavour for how you can create a `toffee` file. However, it is best explained by way of example, such as the following python code in the unit testing framework. As mentioned elsewhere, the python and C++ APIs are laregly equivalent.

```
class DummyToffeeFile():
    def __init__(
        self,
        name,
        num_rt_points=1626,
        scan_cycle_time=3.32,
        ms1_rt_offset=0.17,
        ms2_rt_offset=0.506,
        num_mz_points=1000,
        ms2_name=None,
        ims_type=toffee.IntrinsicMassSpacingType.TOF,
```

(continues on next page)

(continued from previous page)

```
        fraction_not_zero=0.01,
    ):
        self.name = name
        self.hash = sum(ord(ch) for ch in name)
        self.seed = self.hash % ((np.iinfo(np.int32).max + 1) * 2)
        logger.info('%s: hash=%s, seed=%s', self.name, self.hash, self.seed)
        np.random.seed(self.seed)

        self.header = self._header(ims_type)

        # create retention time vectors for each window
        self.scan_cycle_time = scan_cycle_time
        self.ms1_rt_offset = ms1_rt_offset
        self.ms2_rt_offset = ms2_rt_offset
        self.ms1_rt_vector = np.arange(num_rt_points) * self.scan_cycle_time + self.
↪ms1_rt_offset
        self.ms2_rt_vector = np.arange(num_rt_points) * self.scan_cycle_time + self.
↪ms2_rt_offset

        # set the mass over charge bounds of the data
        lower = 90
        upper = 2100
        (
            self.ims_alpha,
            self.ims_beta,
            self.ims_gamma,
            self.ims_coord_vector,
            self.mz_vector,
        ) = self.create_mz_vector(ims_type, lower, upper, num_mz_points)

        # we're going to create 1x MS1 window and 1x MS2 window
        self.ms1_name = toffee.ToffeeWriter.MS1_NAME
        if ms2_name is None:
            ms2_name = toffee.ToffeeWriter.ms2Name(41)
        self.ms2_name = ms2_name
        # window boundaries match ProCan90-M06-07.tof/ms2-041
        self.window_lower, self.window_center, self.window_upper = (608.5, 611.5, 614.
↪5)

        # create a dense matrix of fake data
        ms1_mz, ms1_rt = np.meshgrid(self.mz_vector, self.ms1_rt_vector)
        self.expected_intensities_ms1 = (ms1_rt * ms1_mz).astype(np.uint32)
        ms2_mz, ms2_rt = np.meshgrid(self.mz_vector, self.ms2_rt_vector)
        self.expected_intensities_ms2 = (ms2_rt * ms2_mz).astype(np.uint32)

        # generate our toffee writer so we can add windows to it
        self.tof_fname = name + '.tof'
        self.ims_type = ims_type
        writer = toffee.ToffeeWriter(self.tof_fname, self.ims_type)
        writer.addHeaderMetadata(self.header)

        # loop through the two windows, modify their intensities by setting
        # a random seclectin to zero, convert this into a point cloud and
        # then add it to the toffee writer
        self.scan_data = {}
        for map_name, rt_vector, expected_intensities in zip(
            [self.ms1_name, self.ms2_name],
```

(continues on next page)

(continued from previous page)

```
            [self.ms1_rt_vector, self.ms2_rt_vector],
            [self.expected_intensities_ms1, self.expected_intensities_ms2],
        ):
            # create the raw data container
            scan_data = toffee.RawSwathScanData(map_name, ims_type)
            if map_name == self.ms2_name:
                scan_data.windowLower = self.window_lower
                scan_data.windowCenter = self.window_center
                scan_data.windowUpper = self.window_upper
            else:
                scan_data.windowLower = -1
                scan_data.windowCenter = -1
                scan_data.windowUpper = -1

            # slice the data as if we were a mass spectrometer creating 'scans'
            for r, rt_value in enumerate(rt_vector):
                intensity_vector = expected_intensities[r, :]

                # add some zeros by generating a mask and assigning a random
                # fraction to be True - then set anything in the intensity
                # vector, masked by True, to be zero. In one case, set all
                # intensities to be zero (mimicking a scan with no data)
                if r % 10 == 0:
                    tmp_fraction_not_zero = 0.0
                else:
                    tmp_fraction_not_zero = fraction_not_zero
                mask = np.random.uniform(0, 1, size=intensity_vector.shape)
                mask[mask < tmp_fraction_not_zero] = 0
                mask[mask != 0] = 1
                mask = mask.astype(bool)
                intensity_vector[mask] = 0
                # double check the masking worked as expected
                if (~mask).any():
                    assert (intensity_vector > 0).any()
                mz = self.mz_vector[~mask]
                intens = intensity_vector[~mask]

                # add to the scan
                scan_data.addScan(rt_value, mz, intens)

            # create our point cloud
            pcl = scan_data.toPointCloud()

            # save scan data for later
            self.scan_data[map_name] = scan_data

            # assert that the point cloud properties were set correctly
            assert pcl.name == map_name
            assert pcl.imsType == ims_type
            assert pcl.windowLower == scan_data.windowLower
            assert pcl.windowCenter == scan_data.windowCenter
            assert pcl.windowUpper == scan_data.windowUpper

            # assert that the retention time was correctly calculated
            np.testing.assert_allclose(pcl.scanCycleTime, self.scan_cycle_time)
            if map_name == self.ms2_name:
                np.testing.assert_allclose(pcl.firstScanRetentionTimeOffset, self.ms2_
↪rt_offset)
```

(continues on next page)

(continued from previous page)

```
            else:
                np.testing.assert_allclose(pcl.firstScanRetentionTimeOffset, self.ms1_
↪rt_offset)

            ims_props = pcl.imsProperties
            assert len(ims_props.sliceIndex) == num_rt_points, \
                '{} {}\n{}'.format(len(ims_props.sliceIndex), num_rt_points, ims_
↪props.sliceIndex)

            # assert that the Intrinsic Mass Spacing parameters were calculated
            # correctly
            np.testing.assert_allclose(ims_props.medianAlpha, self.ims_alpha)
            np.testing.assert_allclose(ims_props.medianBeta, self.ims_beta)
            np.testing.assert_allclose(ims_props.gamma, self.ims_gamma)

            # add to the file
            writer.addPointCloud(pcl)

    @classmethod
    def create_mz_vector(cls, ims_type, lower, upper, num_mz_points):
        """
        Create a m/z vector that respects the IMS spacings for a Sciex (TOF)
        type instrument
        """
        if ims_type == toffee.IntrinsicMassSpacingType.TOF:
            return cls._create_mz_vector_tof(lower, upper, num_mz_points)
        elif ims_type == toffee.IntrinsicMassSpacingType.ORBITRAP:
            return cls._create_mz_vector_orbitrap(lower, upper, num_mz_points)
        raise ValueError('Upsupported IMS type: {}'.format(ims_type))

    @classmethod
    def _create_mz_vector_tof(cls, lower, upper, num_mz_points):
        """
        Create a m/z vector that respects the IMS spacings for a Sciex (TOF)
        type instrument
        """
        sqrt_lower = np.sqrt(lower)
        sqrt_upper = np.sqrt(upper)

        # and use these to create dummy Intrinsic Mass Spacing properties
        ims_alpha = (sqrt_upper - sqrt_lower) / num_mz_points
        ims_gamma = int(np.round(sqrt_lower / ims_alpha, decimals=0))
        ims_beta = sqrt_lower - ims_gamma * ims_alpha
        if ims_beta > 0:
            ims_gamma += 1
            ims_beta -= ims_alpha

        # from these create an mass over charge vector with known properties
        ims_coord = np.arange(num_mz_points + 1).astype(int)
        mz = ((ims_coord + ims_gamma) * ims_alpha + ims_beta) ** 2

        # add some padding to ims
        ims_gamma -= 10
        ims_coord += 10

        return ims_alpha, ims_beta, ims_gamma, ims_coord, mz
```

(continues on next page)

(continued from previous page)

```
    @classmethod
    def _create_mz_vector_orbitrap(cls, lower, upper, num_mz_points):
        """
        Create a m/z vector that respects the IMS spacings for a Thermo (Orbitrap)
        type instrument
        """
        inverse_sqrt_lower = 1 / np.sqrt(lower)
        inverse_sqrt_upper = 1 / np.sqrt(upper)

        # and use these to create dummy Intrinsic Mass Spacing properties
        ims_alpha = (inverse_sqrt_upper - inverse_sqrt_lower) / num_mz_points
        ims_gamma = int(np.round(inverse_sqrt_lower / ims_alpha, decimals=0))
        ims_beta = inverse_sqrt_lower - ims_gamma * ims_alpha
        if ims_beta > 0:
            ims_gamma -= 1
            ims_beta += ims_alpha

        # from these create an mass over charge vector with known properties
        ims_coord = np.arange(num_mz_points + 1).astype(int)
        mz = ((ims_coord + ims_gamma) * ims_alpha + ims_beta) ** -2

        # add some padding to ims
        ims_gamma -= 10
        ims_coord += 10

        return ims_alpha, ims_beta, ims_gamma, ims_coord, mz

    @classmethod
    def _header(cls, ims_type):
        if ims_type == toffee.IntrinsicMassSpacingType.TOF:
            header_fname = 'Header-TOF.mzML'
        elif ims_type == toffee.IntrinsicMassSpacingType.ORBITRAP:
            header_fname = 'Header-Orbitrap.mzML'
        else:
            raise ValueError('Upsupported IMS type: {}'.format(ims_type))

        curdir = os.path.dirname(os.path.abspath(__file__))
        header_fname = os.path.join(curdir, header_fname)
        with open(header_fname, 'r') as f:
            return f.read()
```

2.1.1 ScanDataPointCloud

class ScanDataPointCloud

The data from a single MS1 or MS2 window collected as a point cloud of `X` points discovered from `N` scans

Public Members

std::string **name**

The name of the window. This should match one of the formats specified by *toffee::ToffeeWriter*

double **windowLower**

The lower (m/z) bound of the MS selection window - should be `-1` for MS1 scans

double **windowCenter**
The center (m/z) of the MS selection window - should be `-1` for MS1 scans

double **windowUpper**
The upper (m/z) bound of the MS selection window - should be `-1` for MS1 scans

double **scanCycleTime**
The time (in seconds) between each scan for this window. In a typical DIA workflow, we anticipate that this is constant for all windows in a *SwathRun*

double **firstScanRetentionTimeOffset**
The time (in seconds) from the start of the experiment to the first scan for this window.

IntrinsicMassSpacingType **imsType**
The Intrinsic Mass Spacing `type` (i.e. 'TOF', 'Orbitrap')

IMSProperties **imsProperties**
The Intrinsic Mass Spacing properties.

std::vector<uint32_t> **intensity**
The intensity value for each of the `X` points in the point cloud.

2.1.2 RawSwathScanData

class RawSwathScanData
The raw data from a single MS1 or MS2 window collected as a vector of `N` individual scans at discrete retention times

Public Functions

RawSwathScanData(**const** std::string &*name*, IntrinsicMassSpacingType *imsType*)

Parameters

- `name`: the name of the window. This should match one of the formats specified by *toffee::ToffeeWriter*

void **addScan**(double *rt*, **const** std::vector<double> &*mz*, **const** std::vector<uint32_t> &*intensity*)
Add a scan to the raw data

Parameters

- `rt`: the retention time of this scan
- `mz`: the mass over charge data collected during this scan
- `intensity`: the intensity data collected during this scan

void **addMassAccuracyWriter**(**const** std::shared_ptr<**const** MassAccuracyWriter> &*massAccuracyWriter*)
The mass accuracy can be used as a debugging tool to keep track of the actual and converted mass over charge values during conversion to point cloud (index space) using the Intrinsic Mass Spacing.

Parameters

- `if`: this is not null, then a debugging step will be taken during the conversion to a point cloud

ScanDataPointCloud **toPointCloud**() **const**
Convert the raw data into the point cloud format. Included in this is the conversion of mass over charge to (m/z) index space using the Intrinsic Mass Spacing properties (c.f. *toffee::IMassOverChargeRange*)

Public Members

const std::string **name**
The name of the window. This should match one of the formats specified by *toffee::ToffeeWriter*

double **windowLower**
The lower (m/z) bound of the MS selection window - should be `-1` for MS1 scans

double **windowCenter**
The center (m/z) of the MS selection window - should be `-1` for MS1 scans

double **windowUpper**
The upper (m/z) bound of the MS selection window - should be `-1` for MS1 scans

IntrinsicMassSpacingType **imsType**
The type of mass analyzer in terms of the Intrinsic Mass Spacing.

size_t **numberOfScans** = 0
The number (`N`) of scans in this window.

std::vector<double> **retentionTime**
A vector (of size `N`) retention times.

std::vector<std::vector<double>> **mzData**
A vector (of size `N`) of vectors where the nested vector contains the mass over charge data collected during the `n-th` scan

std::vector<std::vector<uint32_t>> **intensityData**
A vector (of size `N`) of vectors where the nested vector contains the intensity data collected during the `n-th` scan

2.1.3 ToffeeWriter

class ToffeeWriter
If we consider the raw data collected within a given window across a full gradient during a SWATH-MS run, each data point represents:

- **Mass over charge** The abscissa refers to the mass over charge ratio of the peptide ion
- **Retention time**: The ordinate refers to the time at which the peptide ion is elutes from the Liquid Chromatography column
- **Intensity**: The applicate is a count of the number of times a peptide ion is detected on the sensor

Through the Intrinsic Mass Spacing, mass over charge can be represented by an unsigned integer, as can intensity. Furthermore, the retention time for each data point can be considered as an index into an associated (but much smaller) vector of doubles. In this manner, we can save the raw data into a file format as 3 vectors of unsigned 32-bit integers with some associated metadata.

`toffee` takes this exact approach, saving each SWATH-MS window as a group in an HDF5 file; where each group has:

- `double` attributes `lower`, `center`, and `upper` to describe the properties of the window;
- `double` attributes `scanCycleTime` (t_c) and `firstScanRetentionTimeOffset` (t_0) that describe the retention time vector for each window such that $t(i) = t_c i + t_0$ in seconds;
- a `uint32` dataset of `retentionTimeIdx`, of equal length as the retention time vector, that represents indices into the `imsCoord` and `intensity` between which the corresponding value of $t(i)$ is applied;

- `double` attributes `imsAlpha` (${IMS}$) and `imsBeta` (${trunc}$) for the Intrinsic Mass Spacing m/z conversion such that ${m}{z} = ({IMS} i_{{m/z}} + {trunc})^2$; and
- `uint32` datasets of `imsCoord` and `intensity` for the point cloud, with one entry for each point in the cloud.

Thus, through this *ToffeeWriter*, we can generate a file with lossless compression that can represent a SWATH-MS `mzML` file with a ten-fold decrease in the file size - making it on par with the file size of the original vendor format.

Public Functions

ToffeeWriter(**const** std::string &*h5FilePath*, IntrinsicMassSpacingType *imsType*)

Parameters

- `h5FilePath`: the output path of the toffee file. If a file already exists at that path, it will be over-written

void **addPointCloud**(**const** *ScanDataPointCloud* &*pcl*) **const**

Add a point cloud for a given window to the toffee file. If a window with the same name already exists in the file, it will throw an exception

Parameters

- `pcl`: the point cloud to be added

void **addHeaderMetadata**(**const** std::string &*header*) **const**

Add a header string to the HDF5 file. In normal circumstances, this will be a PSI-formatted XML string matching the header for mzML files

Public Static Functions

std::string **ms2Name**(**const** size_t &*idx*)

Name of the MS2 HDF5 group corresponding to the specified index.

std::string **imsTypeAsStr**(**const** IntrinsicMassSpacingType &*imsType*)

Name of the IMS Type in string format.

IntrinsicMassSpacingType **imsTypeFromStr**(**const** std::string &*imsTypeAsStr*)

Convert the name of the IMS Type in string format to the enum.

Public Static Attributes

const std::string **ATTR_MAJOR_VERSION** = "FILE_FORMAT_MAJOR_VERSION"

Name of the file major version HDF5 attribute.

const std::string **ATTR_MINOR_VERSION** = "FILE_FORMAT_MINOR_VERSION"

Name of the file minor version HDF5 attribute.

const std::string **ATTR_LIBRARY_VERSION** = "CREATED_BY_LIBRARY_VERSION"

Name of the softare version used to create the file HDF5 attribute.

const std::string **ATTR_EXPERIMENT_METADATA** = "metadataXML"

Name of the run's header HDF5 attribute.

const std::string **ATTR_IMS_TYPE** = "IMSType"
Name of a window's Intrinsic Mass Spacing `type` HDF5 attribute Valid values are: TOF, Orbitrap

const std::string **ATTR_IMS_ALPHA** = "IMSAlpha"
Name of a window's Intrinsic Mass Spacing `alpha` HDF5 attribute.

const std::string **ATTR_IMS_BETA** = "IMSBeta"
Name of a window's Intrinsic Mass Spacing `beta` HDF5 attribute.

const std::string **ATTR_IMS_GAMMA** = "IMSGamma"
Name of a window's Intrinsic Mass Spacing `gamma` HDF5 attribute.

const std::string **ATTR_WINDOW_BOUNDS_LOWER** = "precursorLower"
Name of a window's mass over charge lower bounds HDF5 attribute.

const std::string **ATTR_WINDOW_BOUNDS_UPPER** = "precursorUpper"
Name of a window's center mass over charge HDF5 attribute.

const std::string **ATTR_WINDOW_CENTER** = "precursorCenter"
Name of a window's mass over charge upper bounds HDF5 attribute.

const std::string **ATTR_SCAN_CYCLE_TIME** = "scanCycleTime"
Name of a window's scan cycle time HDF5 attribute.

const std::string **ATTR_WINDOW_RETENTION_TIME_OFFSET** = "firstScanRetentionTimeOffset"
Name of a window's first retention time offset HDF5 attribute.

const std::string **DATASET_RETENTION_TIME_IDX** = "retentionTimeIdx"
Name of a window's retention time index HDF5 dataset.

const std::string **DATASET_IMS_ALPHA_PER_SCAN** = "IMSAlphaPerScan"
Name of a window's Intrinsic Mass Spacing `alpha per scan` HDF5 dataset.

const std::string **DATASET_IMS_BETA_PER_SCAN** = "IMSBetaPerScan"
Name of a window's Intrinsic Mass Spacing `beta per scan` HDF5 dataset.

const std::string **DATASET_MZ_IMS_IDX** = "imsCoord"
Name of a window's (m/z) index HDF5 dataset.

const std::string **DATASET_INTENSITY** = "intensity"
Name of a window's intensity HDF5 dataset.

const std::string **MS1_NAME** = std::string{"ms1"}
Name of the MS1 HDF5 group.

const std::string **MS2_PREFIX** = std::string{"ms2-"}
Prefix for the MS2 HDF5 groups.

2.2 Accessing Data in a Toffee File

2.2.1 SwathRun

class SwathRun
A class that enables access to properties of a full toffee file.

Public Functions

SwathRun(**const** std::string &*fname*)

Parameters

- `fname`: path to the toffee file of interest

std::string **header**() **const**
Extract the metadata header from the specified file. In usual circumstances one could expect this to look like a PSI-formatter XML description of the run

int **formatMajorVersion**() **const**
The file format major version.

int **formatMinorVersion**() **const**
The file format minor version.

IntrinsicMassSpacingType **imsType**() **const**
The type of mass analyzer Intrinsic Mass Spacing.

bool **hasMS1Data**() **const**
Return true if this toffee file contains data for the MS1 window.

std::vector<*MS2WindowDescriptor*> **ms2Windows**() **const**
Return a vector of descriptions of each the MS2 windows contained in the toffee file

std::map<double, std::string> **mapPrecursorsToMS2Names**(**const** std::vector<double> &*precursorMzValues*, double *lowerMzOverlap*, double *upperMzOverlap*) **const**
It is often useful to know which precursors map to specific MS2 windows in the toffee file. For instance, this can be useful for performance reasons when iterating through a spectral library

Parameters

- `precursorMzValues`: A vector of mass over charge values that correspond to the precursors that we wish to map to MS2 windows
- `lowerMzOverlap`: It is usual to have MS2 windows overlap in an experiment, this allows us to exclude precursors with this offest from the lower boundary of an MS2 window
- `upperMzOverlap`: It is usual to have MS2 windows overlap in an experiment, this allows us to exclude precursors with this offest from the upper boundary of an MS2 window

SwathMap **loadSwathMap**(**const** std::string &*mapName*, bool *lossyAdjustIMSCoords* = false) **const**
Load a *SwathMap* for a specific window from the toffee file. As this is a read-only operation, it is thread-safe and multiple threads can read from the same *SwathRun* concurrently.

This function is primarily for the python wrapping, when calling from C++, we recommend using *SwathRun::loadSwathMapPtr*

Parameters

- `mapName`: name of the window of interest (must match the naming convention to *ToffeeWriter*)
- `lossyAdjustIMSCoords`: if true, the IMS coords are translated from the scan specific alpha and beta to the window alpha and beta in a way that minimizes the ppm error. Note, this will minimize the mass error of the transform, but this is a lossy operation that cannot be undone!

std::shared_ptr<*SwathMap*> **loadSwathMapPtr**(**const** std::string &*mapName*, bool *lossyAdjustIMSCoords* = false) **const**
Load a *SwathMap* for a specific window from the toffee file. As this is a read-only operation, it is thread-safe and multiple threads can read from the same *SwathRun* concurrently.

Parameters

- `mapName`: name of the window of interest (must match the naming convention to *ToffeeWriter*)
- `lossyAdjustIMSCoords`: if true, the IMS coords are translated from the scan specific alpha and beta to the window alpha and beta in a way that minimizes the ppm error. Note, this will minimize the mass error of the transform, but this is a lossy operation that cannot be undone!

SwathMapInMemorySpectrumAccess **loadSwathMapInMemorySpectrumAccess**(**const** std::string &*mapName*) **const**

Load a SwathMapInMemorySpectrumAccess for a specific window from the toffee file. As this is a read-only operation, it is thread-safe and multiple threads can read from the same *SwathRun* concurrently.

This function is primarily for the python wrapping, when calling from C++, we recommend using *SwathRun::loadSwathMapPtr*

Parameters

- `mapName`: name of the window of interest (must match the naming convetion to *ToffeeWriter*)

std::shared_ptr<SwathMapInMemorySpectrumAccess> **loadSwathMapInMemorySpectrumAccessPtr**(**const** std::string &*mapName*) **const**

Load a SwathMapInMemorySpectrumAccess for a specific window from the toffee file. As this is a read-only operation, it is thread-safe and multiple threads can read from the same *SwathRun* concurrently.

Parameters

- `mapName`: name of the window of interest (must match the naming convetion to *ToffeeWriter*)

SwathMapSpectrumAccess **loadSwathMapSpectrumAccess**(**const** std::string &*mapName*) **const**

Load a SwathMapSpectrumAccess for a specific window from the toffee file. As this is a read-only operation, it is thread-safe and multiple threads can read from the same *SwathRun* concurrently.

This function is primarily for the python wrapping, when calling from C++, we recommend using *SwathRun::loadSwathMapPtr*

Parameters

- `mapName`: name of the window of interest (must match the naming convetion to *ToffeeWriter*)

std::shared_ptr<SwathMapSpectrumAccess> **loadSwathMapSpectrumAccessPtr**(**const** std::string &*mapName*) **const**

Load a SwathMapSpectrumAccess for a specific window from the toffee file. As this is a read-only operation, it is thread-safe and multiple threads can read from the same *SwathRun* concurrently.

Parameters

- `mapName`: name of the window of interest (must match the naming convetion to *`ToffeeWriter`*)

class MS2WindowDescriptor

A simple data class for describing an MS2 window.

Public Functions

bool **operator==**(**const** *MS2WindowDescriptor* &*other*) **const**

Windows are considered equal if and only if they have the same name, and upper and lower mass over charge ranges

Public Members

std::string **name**

The name of the window in the toffee file.

double **lower**

The lower bound of the range of precursor mass over charge values that are selected to be fragmented in this MS2 window

double **upper**

The upper bound of the range of precursor mass over charge values that are selected to be fragmented in this MS2 window

2.2.2 SwathMap

class SwathMap

The *SwathMap* class is one of the key elements of the the `toffee` library. It allows access raw Time of Flight mass spectrometry data in near constant time, with relatively a small memory footprint.

Public Types

using RTreeIndex = int32_t

Defines the type used to index the mass over charge and retention time values in the RTree. For search performance and memory optimisation, we use `int32_t` as our index type if the resolution of the mass spectrometer means greater than 4e9 mass over charge values can be recorded using the intrinsic mass spacing method of toffee, then this will need to change

using RtImsCoord = boost::geometry::model::point<*RTreeIndex*, 2, boost::geometry::cs::cartesian>

A coordinate in the Point Cloud with retention time as the 'x' value and (m/z) index as the 'y' value.

using RTreeIntensity = uint32_t

Defines the type used to record intensity values in the RTree. For memory optimisation, intensity values remain as `uint32_t` if the sensitivity of the mass spectrometer means signals greater than 4e9 can be recorded, then this will need to change

using PCLPoint = std::pair<*RtImsCoord*, *RTreeIntensity*>

A point in the point cloud, with the signal intesity as the 'value'.

using RStar = boost::geometry::index::rstar<32>

The integer value passed here defines the number of points per leaf node of the RTree. It is selected based on a balance between search performance and memory. Using 4 is analogous to a quad-tree.

using RTree = boost::geometry::index::rtree<*PCLPoint*, *RStar*>
A spatial search tree from boost that can be used for high performance spatial searching of the data contained within a *SwathMap*

using Box = boost::geometry::model::box<*RtImsCoord*>
A type used for defining axis aligned box queries of the RTree.

using Segment = boost::geometry::model::segment<*RtImsCoord*>
A type used for defining segment queries of the RTree.

Public Functions

SwathMap(std::string *name*, double *lower*, double *center*, double *upper*, bool *isMS1*, double *scanCycleTime*, double *firstScanRetentionTimeOffset*, size_t *numberOfScans*, std::shared_ptr<**const** *MassOverChargeTransformer*> *mzTransformer*, **const** std::vector<*PCLPoint*> &*points*)
Primary constructor of a *SwathMap*. In general operation, it is unlikely that a user of the library will need to call this directly. Instead, prefer functions on `SwathRun` that load in windows.

Parameters

- `name`: name of the MS selection window that is being represented by this data. This must take the form of *ToffeeWriter::MS1_NAME* or *ToffeeWriter::ms2Name*
- `lower`: lower (m/z) bound of the MS selection window
- `center`: center (m/z) of the MS selection window
- `upper`: upper (m/z) bound of the MS selection window
- `isMS1`: if true, this *SwathMap* represents data from the MS1 precursor ions, otherwise, the data is from MS2 fragment ions
- `scanCycleTime`: the time (in seconds) between each scan for this window
- `firstScanRetentionTimeOffset`: the time (in seconds) from the start of the experiment to the first scan for this window
- `numberOfScans`: the number of entries in the retention time vector
- `mzTransformer`: transforms to and from the mass over charge world and IMS index space
- `points`: A vector of *toffee::SwathMap::PCLPoint* representing the signal intensities detected by the mass spectrometer

SwathMap()

SwathMap(**const** *SwathMap* &*other*)

SwathMap &**operator=**(**const** *SwathMap* &*other*)

SwathMap(*SwathMap* &&*other*)

SwathMap &**operator=**(*SwathMap* &&*other*)

std::string **name**() **const**
Return the name of the *SwathMap* as it was recorded in the original toffee file

bool **isMS1**() **const**
Return true if this *SwathMap* represents data from the MS1 precursor ions, otherwise false if the data is from MS2 fragment ions

double **precursorLowerMz**() **const**
Lower (m/z) bound of the MS selection window.

double **precursorCenterMz**() **const**
Center (m/z) of the MS selection window.

double **precursorUpperMz**() **const**
Upper (m/z) bound of the MS selection window.

double **scanCycleTime**() **const**
The time (in seconds) between each scan for this window. In a typical DIA workflow, we anticipate that this is constant for all windows in a *SwathRun*

double **firstScanRetentionTimeOffset**() **const**
The time (in seconds) from the start of the experiment to the first scan for this window.

const Eigen::VectorXd &**retentionTime**() **const**
A vector (of size N) retention times where N is the number of scans performed, or spectra, for this window

ptrdiff_t **numberOfSpectra**() **const**
The number of scans performed, or spectra, for this window.

const Eigen::VectorXd &**massOverCharge**() **const**
A vector (of size M) mass over charge (m/z) values, where M represents all possible (m/z) values that **could** be detected by the mass spectrometer in its current configuration

std::shared_ptr<**const** *MassOverChargeTransformer*> **getMzTransformer**() **const**
Get a transformer that can report the bounds of the mass over charge for this *SwathMap*, as well as perform conversions between (m/z) and (m/z) index

Chromatogram2DDense **extractedIonChromatogram**(**const** *IMassOverChargeRange* &*mzRange*) **const**
Extract a dense two-dimensional chromatogram from the *SwathMap* based on the specified mass over change range. In this instance, the returned chromatogram will span the full retention time range if you only require a subset of retention times, see *toffee::SwathMap::filteredExtractedIonChromatogram*

Warning If the mass range is large, you can expect a large memory requirement for this function. In this case, you may wish to use *toffee::SwathMap::extractedIonChromatogramSparse*

Parameters

- mzRange: specifies the upper and lower bounds of mass over charge for querying the *SwathMap*

Chromatogram2DSparse **extractedIonChromatogramSparse**(**const** *IMassOverChargeRange* &*mzRange*) **const**
Extract a sparse two-dimensional chromatogram from the *SwathMap* based on the specified mass over change range. In this instance, the returned chromatogram will span the full retention time range if you only require a subset of retention times, see *toffee::SwathMap::filteredExtractedIonChromatogramSparse*

Parameters

- mzRange: specifies the upper and lower bounds of mass over charge for querying the *SwathMap*

Chromatogram2DDense **filteredExtractedIonChromatogram**(**const** *IMassOverChargeRange* &*mzRange*, **const** *IRetentionTimeRange* &*rtRange*) **const**
Extract a dense two-dimensional chromatogram from the *SwathMap* based on the specified mass over change and retention time ranges.

Warning If the ranges are large, you can expect a large memory requirement for this function. In this case, you may wish to use *toffee::SwathMap::filteredExtractedIonChromatogramSparse*

Parameters

- `mzRange`: specifies the upper and lower bounds of mass over charge for querying the *SwathMap*
- `rtRange`: specifies the upper and lower bounds of retention time for querying the *SwathMap*

Chromatogram2DSparse **filteredExtractedIonChromatogramSparse**(**const** *IMassOverChargeRange &mzRange*, **const** *IRetentionTimeRange &rtRange*) **const**

Extract a sparse two-dimensional chromatogram from the *SwathMap* based on the specified mass over change and retention time ranges.

Parameters

- `mzRange`: specifies the upper and lower bounds of mass over charge for querying the *SwathMap*
- `rtRange`: specifies the upper and lower bounds of retention time for querying the *SwathMap*

class MassOverChargeTransformer

A simple class for converting between (m/z) space and (m/z) index space. Furthermore, it is typically used to define the full range of allowable mass over charge of a *SwathMap*. Importantly, this class and its concrete children allow a user to convert between mass over charge space and (m/z) index space using the Intrinsic Mass Spacing parameters `alpha` and `beta` where:

$$\frac{m}{z} = \left(\alpha_{IMS} i_{\sqrt{m/z}} + \beta_{trunc}\right)^2$$

where α_{IMS} is the IMS value, $i_{\sqrt{m/z}}$ is the unsigned integer index of the m/z value in $\sqrt{m/z}$ space, and β_{trunc} is to control for any truncation error that may be introduced from the conversion process.

The principle use of the ranges is to enable multiple different ways of defining a range, and then converting between world and index space. In the instance of converting from world to index, the range classes will 'snap' to the closest index. The upper and lower bounds are *inclusive*, meaning that a range with lowerIMSCoord, upperIMSCoord = 154, 154 would return data with a single row (the data contained along index 154). Similarly, a range with lowerIMSCoord, upperIMSCoord = 153, 155 would return data with three rows, centered at imsCoord 154.

Subclassed by toffee::MassOverChargeTransformerOrbitrap, *toffee::MassOverChargeTransformerTOF*

Public Functions

MassOverChargeTransformer()

virtual ~MassOverChargeTransformer()

virtual double **lowerMzInWorldCoords**() **const** = 0

The lower bound of mass over charge range in "normal" space.

virtual double **upperMzInWorldCoords**() **const** = 0

The upper bound of mass over charge range in "normal" space.

virtual int32_t **lowerMzInIMSCoords**() **const** = 0
The lower bound of mass over charge range in (m/z) index space.

virtual int32_t **upperMzInIMSCoords**() **const** = 0
The upper bound of mass over charge range in (m/z) index space.

IntrinsicMassSpacingType **imsType**() **const**
The Intrinsic Mass Spacing type See *toffee::IMassOverChargeRange* for more details

double **imsAlpha**() **const**
The Intrinsic Mass Spacing `alpha` parameter. See *toffee::IMassOverChargeRange* for more details

double **imsBeta**() **const**
The Intrinsic Mass Spacing `beta` parameter See *toffee::IMassOverChargeRange* for more details

int32_t **imsGamma**() **const**
The Intrinsic Mass Spacing `gamma` parameter See *toffee::IMassOverChargeRange* for more details

int32_t **toIMSCoords**(double *mz*) **const**
Convert the specified mass over charge into Intrinsic Mass Spacing index space

Parameters

- `mz`: mass over charge value to be converted

virtual int32_t **toIMSCoordsPreGammaAdjustment**(double *mz*) **const** = 0
Convert the specified mass over charge into Intrinsic Mass Spacing index space before applying IMS gamma. This function allows the coord to be negative, and should only every be used when first creating the the point cloud

Parameters

- `mz`: mass over charge value to be converted

virtual double **toWorldCoords**(int32_t *imsIdx*) **const** = 0
Convert the specified Intrinsic Mass Spacing index into mass over charge space

Parameters

- `imsCoord`: IMS index to be converted

class MassOverChargeTransformerTOF : **public** toffee::*MassOverChargeTransformer*
A simple class for converting between (m/z) space and (m/z) index space. Furthermore, it is typically used to define the full range of allowable mass over charge of a *SwathMap*.

See *toffee::IMassOverChargeRange* for how this conversion takes place

Public Functions

MassOverChargeTransformerTOF()

MassOverChargeTransformerTOF(double *imsAlpha*, double *imsBeta*, int32_t *imsGamma*)

Parameters

- `imsAlpha`: the Intrinsic Mass Spacing `alpha` parameter
- `imsBeta`: the Intrinsic Mass Spacing `beta` parameter

MassOverChargeTransformerTOF(double *imsAlpha*, double *imsBeta*, int32_t *imsGamma*, int32_t *lowerMzInIMSCoords*, int32_t *upperMzInIMSCoords*)

Parameters

- `imsAlpha`: the Intrinsic Mass Spacing `alpha` parameter
- `imsBeta`: the Intrinsic Mass Spacing `beta` parameter
- `lowerMzInIMSCoords`: the lower bound of mass over charge range in IMS index space
- `upperMzInIMSCoords`: the upper bound of mass over charge range in IMS index space

double **lowerMzInWorldCoords**() **const**
The lower bound of mass over charge range in "normal" space.

double **upperMzInWorldCoords**() **const**
The upper bound of mass over charge range in "normal" space.

int32_t **lowerMzInIMSCoords**() **const**
The lower bound of mass over charge range in (m/z) index space.

int32_t **upperMzInIMSCoords**() **const**
The upper bound of mass over charge range in (m/z) index space.

int32_t **toIMSCoordsPreGammaAdjustment**(double *mz*) **const**
Convert the specified mass over charge into Intrinsic Mass Spacing index space before applying IMS gamma. This function allows the coord to be negative, and should only every be used when first creating the the point cloud

Parameters

- `mz`: mass over charge value to be converted

double **toWorldCoords**(int32_t *imsIdx*) **const**
Convert the specified Intrinsic Mass Spacing index into mass over charge space

Parameters

- `imsCoord`: IMS index to be converted

2.2.3 Spectra & Chromatograms

class Spectrum1D
A simple data holding class primarily used to return results from querying a *SwathMap*.

Public Functions

Spectrum1D(**const** Eigen::VectorXd &*mz*)
Initialise the *Spectrum1D* with a given set of mass over charge values and setting a correspondingly sized intensity vector to zeroes

Parameters

- `mz`: a vector (of size `M`) mass over charge values captured by this spectrum

Spectrum1D(**const** std::vector<double> &*mz*, **const** std::vector<int> &*intens*)
Initialise the *Spectrum1D* with a given set of mass over charge values and a correspondingly sized intensity vector

Parameters

- `mz`: a vector (of size `M`) mass over charge values captured by this spectrum
- `intens`: a vector (of size `M`) intensity values captured by this spectrum

Public Members

Eigen::VectorXd **massOverCharge**
An Eigen::VectorXd (of size `M`) mass over charge values captured by this spectrum

Eigen::VectorXi **intensities**
An Eigen::VectorXi (of size `M`) intensity values captured by this spectrum

class Chromatogram1D
A simple data holding class primarily used to return results from querying a *SwathMap*.

Public Functions

Chromatogram1D(**const** Eigen::VectorXd &*rt*)
Initialise the *Chromatogram1D* with a given set of retention times and setting a correspondingly sized intensity vector to zeroes

Parameters

- `rt`: a vector (of size `N`) retention times captured by this chromatogram

Public Members

Eigen::VectorXd **retentionTime**
An Eigen::VectorXd (of size `N`) retention times captured by this chromatogram

Eigen::VectorXi **intensities**
An Eigen::VectorXi (of size `N`) intensity values captured by this chromatogram

class Chromatogram2DSparse
A simple data holding class primarily used to return results from querying a *SwathMap*. The sparse moiniker refers to the fact that zero intensities are excluded from the intensity matrix to improve performance

Public Types

using Value = int
The type used for representing the intensity values.

using Triplet = Eigen::Triplet<*Value*>
A (retention time index, (m/z) index, intensity) triple that corresponds to a point in the point cloud. These are primarily used for loading the toffee::Chromatogram2DSparse::Martix

using Matrix = Eigen::SparseMatrix<*Value*>
An Eigen::SparseMatrix in column major layout that is used to represent the intensities

Public Functions

Chromatogram2DSparse(size_t *retentionTimeSize*, size_t *massOverChargeSize*, **const** std::vector<*Triplet*> &*triplets*)

Parameters

- `retentionTimeSize`: the number of retention times captured by this chromatogram
- `massOverChargeSize`: the number of mass over charge values captured by this chromatogram
- `triplets`: a vector of triplets that represent points in the point cloud that are used to initialise the *Chromatogram2DSparse::intensities* sparse matrix

Chromatogram2DSparse(*Chromatogram2DSparse* &&*other*)

Chromatogram2DSparse &**operator=**(*Chromatogram2DSparse* &&*other*)

Public Members

Eigen::VectorXd **retentionTime**
An Eigen::VectorXd (of size `N`) retention times captured by this chromatogram

Eigen::VectorXd **massOverCharge**
An Eigen::VectorXd (of size `M`) mass over charge values captured by this chromatogram

Matrix **intensities**
An Eigen::SparseMatrix (of shape `NxM`) in column major layout, of intensity values captured by this chromatogram

class Chromatogram2DDense
A simple data holding class primarily used to return results from querying a *SwathMap*. The dense moniker refers to the fact that zero intensities are included in the intensity matrix and may result in a large memory footprint.

Warning Careless useage of this class may result in very large memory requirements of the calling function. In these cases, *Chromatogram2DSparse* may be useful.

Public Functions

Chromatogram2DDense(**const** *Chromatogram2DSparse* &*chromatogram2DSparse*)
Convert a two-dimensional chromatogram with sparse intensities into a two-dimensional chromatogram with intensities in a dense matrix format.

Parameters

- `chromatogram2DSparse`: the sparse representation of the chromatogram

Public Members

Eigen::VectorXd **retentionTime**
An Eigen::VectorXd (of size `N`) retention times captured by this chromatogram

Eigen::VectorXd **massOverCharge**
An Eigen::VectorXd (of size `M`) mass over charge values captured by this chromatogram

Eigen::MatrixXi **intensities**
An Eigen::MatrixXi (of shape `NxM`) in column major layout, of intensity values captured by this chromatogram

2.2.4 MassOverChargeRange

class IMassOverChargeRange
Subclassed by *toffee::MassOverChargeRange*, *toffee::MassOverChargeRangeIMSCoords*, toffee::MassOverChargeRangeWithPixelHalfWidth, toffee::MassOverChargeRangeWithPPMFullWidth

Public Functions

virtual ~IMassOverChargeRange()

virtual int32_t **lowerMzInIMSCoords**(**const** std::shared_ptr<**const** *MassOverChargeTransformer*> &*mzTransformer*) **const** = 0
Using the Intrinsic Mass Spacing parameters `alpha` and `beta`, convert the lower bound of the concrete child class into its corresponding (m/z) index

virtual int32_t **upperMzInIMSCoords**(**const** std::shared_ptr<**const** *MassOverChargeTransformer*> &*mzTransformer*) **const** = 0
Using the Intrinsic Mass Spacing parameters `alpha` and `beta`, convert the upper bound of the concrete child class into its corresponding (m/z) index

Warning: doxygenclass: Cannot find class "toffee::MassOverChargeRangeWithPixelWidth" in doxygen xml output for project "toffee" from directory: _build/doxygen/xml/

Warning: doxygenclass: Cannot find class "toffee::MassOverChargeRangeWithPPMWidth" in doxygen xml output for project "toffee" from directory: _build/doxygen/xml/

class MassOverChargeRange : **public** toffee::*IMassOverChargeRange*
An implementation of *IMassOverChargeRange* that allows the user to explicitly set the upper and lower bounds of the mass over charge range

Public Functions

MassOverChargeRange(double *lowerMz*, double *upperMz*)

Parameters

- `lowerMz`: the lower bound of mass over charge for this range
- `upperMz`: the upper bound of mass over charge for this range

double **massOverChargeLower**() **const**
The lower bound of mass over charge for this range.

double **massOverChargeUpper**() **const**
The upper bound of mass over charge for this range.

int32_t **lowerMzInIMSCoords**(**const** std::shared_ptr<**const** *MassOverChargeTransformer*> &*mzTransformer*) **const**
See *IMassOverChargeRange::lowerMzInIMSCoords*.

int32_t **upperMzInIMSCoords**(**const** std::shared_ptr<**const** *MassOverChargeTransformer*> &*mzTransformer*) **const**
See *IMassOverChargeRange::upperMzInIMSCoords*.

class MassOverChargeRangeIMSCoords : **public** toffee::*IMassOverChargeRange*
An implementation of *IMassOverChargeRange* that allows the user to explicitly set the upper and lower bounds of the mass over charge range in terms of the (m/z) index space

Public Functions

MassOverChargeRangeIMSCoords(int32_t *lowerIMSCoord*, int32_t *upperIMSCoord*)

Parameters

- `lowerIMSCoord`: the lower bound of mass over charge for this range in (m/z) index space
- `upperIMSCoord`: the upper bound of mass over charge for this range in (m/z) index space

int32_t **lowerMzInIMSCoords**(**const** std::shared_ptr<**const** *MassOverChargeTransformer*> &*mzTransformer*) **const**
See *IMassOverChargeRange::lowerMzInIMSCoords*.

int32_t **upperMzInIMSCoords**(**const** std::shared_ptr<**const** *MassOverChargeTransformer*> &*mzTransformer*) **const**
See *IMassOverChargeRange::upperMzInIMSCoords*.

2.2.5 RetentionTimeRange

class IRetentionTimeRange
An interface class that allows users to define retention time ranges using various methods. These ranges are used by *toffee::SwathMap* when querying the intensity data.

The principle use of the ranges is to enable multiple different ways of defining a range, and then converting between world and index space. In the instance of converting from world to index, the range classes will 'snap' to the closest index. The upper and lower bounds are *inclusive*, meaning that a range with lowerRTIdx, upperRTIdx = 154, 154 would return data with a single row (the data contained along index 154). Similarly, a range with lowerRTIdx, upperRTIdx = 153, 155 would return data with three rows, centered at retentionTimeIdx 154.

Subclassed by *toffee::RetentionTimeRange*, toffee::RetentionTimeRangeWithPixelHalfWidth

Public Functions

virtual ~IRetentionTimeRange()

virtual int32_t **lowerRTIdx**(**const** Eigen::VectorXd &*retentionTimes*) **const** = 0
Find the index in the specified retention time vector that contains the value that is *closest* to the lower bound specified in the construction of a concrete child class

Warning This function assumes, but does not check, that the retention time vector is sorted in ascending order

Parameters

- `retentionTimes`: a vector of retention times in ascending order

virtual int32_t **upperRTIdx**(**const** Eigen::VectorXd &*retentionTimes*) **const** = 0

Find the index in the specified retention time vector that contains the value that is *closest* to the upper bound specified in the construction of a concrete child class

Warning This function assumes, but does not check, that the retention time vector is sorted in ascending order

Parameters

- `retentionTimes`: a vector of retention times in ascending order

Public Static Functions

int32_t **toRTIdx**(double *retentionTime*, **const** Eigen::VectorXd &*retentionTimes*)

Find the index in the specified retention time vector that contains the value that is *closest* to the specified retention time we are searching for. If there are two values equidistant, arbitrarily preference the higher index.

Warning This function assumes, but does not check, that the retention time vector is sorted in ascending order

Parameters

- `retentionTime`: the retention time value that we wish to search for
- `retentionTimes`: a vector of retention times in ascending order

Warning: doxygenclass: Cannot find class "toffee::RetentionTimeRangeWithPixelWidth" in doxygen xml output for project "toffee" from directory: _build/doxygen/xml/

class RetentionTimeRange : **public** toffee::*IRetentionTimeRange*

An implementation of *IRetentionTimeRange* that allows the user to explicitly set the upper and lower bounds of the retention time range

Public Functions

RetentionTimeRange(double *lowerRetentionTime*, double *upperRetentionTime*)

Parameters

- `lowerRetentionTime`: the lower bound of retention time for this range
- `upperRetentionTime`: the upper bound of retention time for this range

double **retentionTimeLower**() **const**

The lower bound of retention time for this range.

double **retentionTimeUpper**() **const**

The upper bound of retention time for this range.

int32_t **lowerRTIdx**(**const** Eigen::VectorXd &*retentionTimes*) **const**

See *IRetentionTimeRange::lowerRTIdx*.

int32_t **upperRTIdx**(**const** Eigen::VectorXd &*retentionTimes*) **const**
See *IRetentionTimeRange::upperRTIdx*.

2.3 Versioning

class Version

Public Static Functions

static int **combineFileFormatVersions**(int *majorVersion*, int *minorVersion*)
A helper function to combine major and minor version numbers into a form that can be easily compared

Public Static Attributes

const std::string **LIBRARY_VERSION** = "dev"
The version of the library.

const int **FILE_FORMAT_MAJOR_VERSION_CURRENT** = 1
The latest major version of the `toffee` file format.

const int **FILE_FORMAT_MINOR_VERSION_CURRENT** = 2
The latest minor version of the `toffee` file format.

const int **FILE_FORMAT_MAJOR_VERSION_COMPATIBILITY** = 0
The version of the major `toffee` file that we are backwards compatible with

const int **FILE_FORMAT_MINOR_VERSION_COMPATIBILITY** = 2
The version of the minor `toffee` file that we are backwards compatible with

const int **FILE_FORMAT_NUM_ALLOWED_MINOR_VERSIONS** = 1000
We do not support minor versions greater than or equal to this number.

Master: || Dev:

`Toffee` is a library and file format for Time of Flight SWATH-MS data. The file format provides lossless compression that results in files of a similar size to those from the proprietary and closed vendor format. In addition, the high-performance C++ library implements spatial data structures to allow a user to extract spectrographic (slice along mass over charge axis) and chromatographic (slice along retention time) data in constant time.

Toffee was born out of a need to store and access SWATH-MS data in a state-of-the-art high-throughput proteomics facility, ProCan, capable of generating thousands of files per month. Using the `mzML` file format in this environment would quickly outstrip the storage hardware available and we believe that such a limitation limits the potential of this technology. The challenges around `mzML` can be summarised into three categories:

1. **File size**: Biobank-scale proteomics facilities may run upwards of 100,000 SWATH-MS runs; operating in a manner typical to ProCan results in Sciex `wiff` files 1-2 GB per each that unpack to 10-20 GB when converted to `mzML` leading to petabytes of data that needs to be stored and archived. Furthermore, this increase in file size adds significant time to processing, making analytics software largely IO-bound. On the ProCan90 dataset, toffee files are 95-100% the size of the original vendor files.
2. **Random access**: Indexed `mzML` substantially improves randomly accessing single scan data (at constant retention time), yet algorithms often require slices along the mass over charge axis and this requires iterating over the

full `mzML` file. Toffee facilitates a different access model allowing near constant time slicing in both retention time and mass over charge axes.

3. **Testability**: A key challenge to improving downstream software is the slow iterative cycle imposed by storing experimental data in `mzML`. Building reliable and robust algorithms requires a strong testing framework of both unit and regression tests and a test harness that encourages developers to use it. The IO-bound nature of `mzML` files risks artificial barriers to test adoption. However, by solving points 1 and 2 above, extremely small (small enough to be committed to the repository) toffee files can be generated with exemplar data for integration into a unit and regression testing frameworks.

Toffee files are based on the open HDF5 format and can thus be read by many different programming languages. Within the toffee documentation, the ToffeeWriter class outlines the structure of the HDF5 file, and should be considered the canonical description.

In addition to the file format, toffee is also a high-performance C++ library for accessing the data in toffee files. By and large, the python classes are direct wrappings of the C++ code and API documentation can be considered largely equivalent. We use pybind11 for wrapping, and this will automatically take care of conversions of `numpy` and `scipy` matrices to corresponding `Eigen` matrices, albeit by creating copies.

CHAPTER

THREE

FOR USERS

Toffee is made available through the conda python packaging system. It can be installed using:

```
conda install --yes -c cmriprocan toffee
```

It is also included in a simple `cmriprocan/toffee` Docker image with conda and toffee only, along with `cmriprocan/openms-toffee` that is a Docker image for those operating a containerised workflow.

CHAPTER

FOUR

FOR DEVELOPERS

We are basing our development workflow around Microsoft Visual Studio Code and conda. The following should help you set up a development environment. In general, we aim to use conda 'env' to manage dependencies.

1. If you haven't already, install `git` using your favourite method and clone this repository
2. If you haven't already, install `conda`
3. If you haven't already, install `anaconda-client` using `conda install --yes anaconda-client`
4. If you haven't already (and you're on a mac), install the MacOS SDK

```
curl -L -o MacOSX10.9.sdk.tar.xz https://github.com/phracker/MacOSX-SDKs/releases/
↪download/10.13/MacOSX10.9.sdk.tar.xz
tar -xzfv MacOSX10.9.sdk.tar.xz
sudo mv MacOSX10.9.sdk /opt/MacOSX10.9.sdk
```

5. Log in to anaconda client as `ProCanSoftEngRobot` – you may need to ask the team for credentials
6. If you haven't already, download `VSCode` and install
7. If you haven't already, open VSCode and install the following extensions (look for the icon of the left side that looks like a square)

- Microsoft "python" extension
- Microsoft "C++" extension
- vector-of-bool "CMake Tools" extension
- Microsoft "Visual Studio Code Tools for AI" extension (this gets you jupyter notebooks working, among other things)

8. From within VSCode, open this repository's root directory; you don't need to worry about workspaces
9. Open up a terminal in VSCode (ctrl + backtick works on MacOS)

- Change into the `.dev-environment` folder and run `bash create_dev_conda_environment.sh` – this will set up all of the dependecies in a conda environment called ‘‘dev-toffee‘

10. Open the Command Palette (cmd + shift + p on MacOS) and search for "python: select interpreter" and chose any value. This will create a `settings.json` fie in `.vscode` in the root of the repository.
11. Copy the following into `<repository-root>/.vscode/settings.json`, being sure to replace `<your-anaconda-root>` with the correct path

```
{
    "python.pythonPath": "<your-anaconda-root>/envs/dev-toffee/bin/python",
    "cmake.cmakePath": "<your-anaconda-root>/envs/dev-toffee/bin/cmake",
    "cmake.generator": "Ninja",
```

(continues on next page)

(continued from previous page)

```
    "cmake.configureSettings": {
        "CMAKE_MAKE_PROGRAM": "<your-anaconda-root>/envs/dev-toffee/bin/ninja",
        "CMAKE_C_COMPILER": "<your-anaconda-root>/envs/dev-toffee/bin/clang",
        "CMAKE_CXX_COMPILER": "<your-anaconda-root>/envs/dev-toffee/bin/clangxx"
    },
    "cmake.configureOnOpen": true,
    "files.associations": {
        "array": "cpp",
        "*.tcc": "cpp",
        "cctype": "cpp",
        "clocale": "cpp",
        "cmath": "cpp",
        "complex": "cpp",
        "cstdarg": "cpp",
        "cstddef": "cpp",
        "cstdint": "cpp",
        "cstdio": "cpp",
        "cstdlib": "cpp",
        "cstring": "cpp",
        "ctime": "cpp",
        "cwchar": "cpp",
        "cwctype": "cpp",
        "deque": "cpp",
        "forward_list": "cpp",
        "list": "cpp",
        "unordered_map": "cpp",
        "unordered_set": "cpp",
        "vector": "cpp",
        "exception": "cpp",
        "optional": "cpp",
        "fstream": "cpp",
        "functional": "cpp",
        "initializer_list": "cpp",
        "iomanip": "cpp",
        "iosfwd": "cpp",
        "iostream": "cpp",
        "istream": "cpp",
        "limits": "cpp",
        "memory": "cpp",
        "new": "cpp",
        "numeric": "cpp",
        "ostream": "cpp",
        "sstream": "cpp",
        "stdexcept": "cpp",
        "streambuf": "cpp",
        "string_view": "cpp",
        "system_error": "cpp",
        "cinttypes": "cpp",
        "type_traits": "cpp",
        "tuple": "cpp",
        "typeindex": "cpp",
        "typeinfo": "cpp",
        "utility": "cpp",
        "valarray": "cpp",
        "variant": "cpp",
        "atomic": "cpp"
    },
```

(continues on next page)

(continued from previous page)

```
    "python.linting.pylintEnabled": false,
    "git.autofetch": true,
    "python.linting.flake8Enabled": true,
    "python.linting.flake8Args": [
        "--max-line-length=120"
    ],
    "editor.rulers": [120]
    "python.unitTest.unittestEnabled": false,
    "python.unitTest.nosetestsEnabled": false,
    "python.unitTest.pyTestEnabled": true,
    "editor.minimap.enabled": false,
    "C_Cpp.intelliSenseEngineFallback": "Disabled",
}
```

We follow the OpenVDB style guide for the C++ and PEP-8 for our python code, so please aim to stay consistent with the rest of the code base. Contributions will be pass through peer review and style will be one element that is reviewed.

CHAPTER

FIVE

CHANGES

CHAPTER

SIX

CHANGE LOG

6.1 0.14

6.1.1 0.14.3

- Introduced a new concept of using a raw toffee file to "re-quantify" the results of PyProphet. In essense, we can use the retention time reported by PyProphet, and the m/z values in the search library to anchor the data we extract from the toffee file. From here, we can then fit an analytic 2D Gaussian surface to the raw data using least-squares. See the docs/jupyter/requant.ipynb for details of both the equations and the results. The function can be called using the following:

```
usage: requantify_pyprophet_sqlite [-h] [--max_q_value_rs MAX_Q_VALUE_RS]
                                   [--max_peptide_q_value_rs MAX_PEPTIDE_Q_VALUE_RS]
                                   [--max_protein_q_value_rs MAX_PROTEIN_Q_VALUE_RS]
                                   [--max_peptide_q_value_experiment_wide MAX_PEPTIDE_
↪Q_VALUE_EXPERIMENT_WIDE]
                                   [--max_protein_q_value_experiment_wide MAX_PROTEIN_
↪Q_VALUE_EXPERIMENT_WIDE]
                                   [--max_peptide_q_value_global MAX_PEPTIDE_Q_VALUE_
↪GLOBAL]
                                   [--max_protein_q_value_global MAX_PROTEIN_Q_VALUE_
↪GLOBAL]
                                   [--max_peak_group_rank MAX_PEAK_GROUP_RANK]
                                   [--lower_window_overlap LOWER_WINDOW_OVERLAP]
                                   [--upper_window_overlap UPPER_WINDOW_OVERLAP]
                                  output_filename toffee_filename
                                  pyprophet_filename

Take the SQLite output from PyProphet and re-quantifies the intensities. The
new file will contain the following columns || "ProteinName": The identifier
of the protein || "Sequence": The identifier of the peptide ||
"FullPeptideName": The identifier of the precursor || "Charge": The charge of
the precursor || "peak_group_rank": The rank of the precursor peak group ||
"MS1Intensity": The newly quantified MS1 intensity || "MS2Intensity": The
newly quantified MS2 intensity || "ModelParamSigmaRT": The Sigma RT parameter
of the analytic model || "ModelParamSigmaMz": The Sigma m/z parameter of the
analytic model || "ModelParamRT0": The RT0 parameter of the analytic model ||
"ModelParamMz0MS1": The m/z_0 parameter of the analytic model for MS1 ||
"ModelParamMz0MS2": The m/z_0 parameter of the analytic model for MS2 ||
"ModelParamAmplitudes": The amplitude parameters of the analytic model with
";" separating MS1 and MS2, and "," separating each fragment.

positional arguments:
```

(continues on next page)

(continued from previous page)

```
  output_filename       Filename for the output results (*.csv.gz).
  toffee_filename       The raw data toffee filename (*.tof).
  pyprophet_filename    Filename for the PyProphet SQLite results that matches
                        the toffee file (*.osw).

optional arguments:
  -h, --help            show this help message and exit
  --max_q_value_rs MAX_Q_VALUE_RS
                        Run specific peak group FDR threshold.
  --max_peptide_q_value_rs MAX_PEPTIDE_Q_VALUE_RS
                        Run specific peptide FDR threshold.
  --max_protein_q_value_rs MAX_PROTEIN_Q_VALUE_RS
                        Run specific protein FDR threshold.
  --max_peptide_q_value_experiment_wide MAX_PEPTIDE_Q_VALUE_EXPERIMENT_WIDE
                        Experiment wide peptide FDR threshold.
  --max_protein_q_value_experiment_wide MAX_PROTEIN_Q_VALUE_EXPERIMENT_WIDE
                        Experiment wide protein FDR threshold.
  --max_peptide_q_value_global MAX_PEPTIDE_Q_VALUE_GLOBAL
                        Global peptide FDR threshold.
  --max_protein_q_value_global MAX_PROTEIN_Q_VALUE_GLOBAL
                        Global protein FDR threshold.
  --max_peak_group_rank MAX_PEAK_GROUP_RANK
                        Number of peak groups to consider.
  --lower_window_overlap LOWER_WINDOW_OVERLAP
                        Positive value to indicate the MS2 window lower
                        overlap (in Da).This should match the settings used in
                        OpenMSToffee/OpenSwath.
  --upper_window_overlap UPPER_WINDOW_OVERLAP
                        Positive value to indicate the MS2 window upper
                        overlap (in Da).This should match the settings used in
                        OpenMSToffee/OpenSwath
```

6.1.2 0.14.2

- Significant performance improvement in the Sciex raw data reader – memory usage down by >60% and runtime down by 50%

6.1.3 0.14.1

- Added new conversion method that converts raw Sciex data directly to toffee (PD-892)

```
$ raw_sciex_data_to_toffee --help
usage: raw_sciex_data_to_toffee [-h] [--filter_ms2_window FILTER_MS2_WINDOW]
                                [--hide_progress_bar] [--debug]
                                zip_filename toffee_filename

Convert raw Sciex zip data file to toffee

positional arguments:
  zip_filename          The input filename (*.zip).
  toffee_filename       The output filename (*.tof).

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  --filter_ms2_window FILTER_MS2_WINDOW
                        If positive integer, only this MS2 window will be
                        included.
  --hide_progress_bar   If set, then progress bar will not be shown
  --debug               If set, then debugging logs will be printed
```

6.2 0.13

6.2.1 0.13.1

- Changed license to MIT and fixed documentation for https://toffee.readthedocs.io

6.3 0.12

6.3.1 0.12.18

- Updated features for the manual validation tool based on feedback from first round of validation (PD-881)

6.3.2 0.12.17

- Added a small app to enable visual/manual validation of retention times picked for specific peptide queueries (PD-881)
- Changed return signature of `ToffeeFragmentsPlotter::load_raw_data` to include MS1 chromatogram

6.3.3 0.12.16

- Added optional flag so that when you are loading a SwathMap, you can adjust the IMS coords to minimise the PPM error when slicing the data as a 2D image. (PD-879)

6.3.4 0.12.15

- Bumped `psims` requirement to `0.1.27` that incorporates our fix for the bug in `lxml` into their `__exit__` method. This should be much more robust against catching other errors during the xml serialisation. Removed the fix from our code (PD-875)

6.3.5 0.12.14

- Zero-intensity points in a spectra are not copied from mzML to toffee. These can be losslessly recovered. (PD-876)

6.3.6 0.12.13

- Added helper function to SwathRun to give immediate knowledge of if there is any MS1 data in the toffee file (PD-642)

6.3.7 0.12.12

- Fixed the bug where `lxml` would crash on closing large mzML files (PD-875)
- Extracted header data directly from the mzML file and stored in the toffee file (PD-873). This required that headers were moved from being an HDF5 attribute to a dataset, so the file format version has been bumped to `1.2`. This is not a breaking change within toffee.

6.3.8 0.12.11

- Robustness improvements to last – and a slight change to CircleCI config to hopefully build the Docker image.

6.3.9 0.12.10

- Enabled conversion of mzML to toffee files using pyteomics. This now means the toffee library is completely stand-alone from the OpenMS code base. `psims` and `pyteomcs` both need to be installed using `pip` as their `conda` versions are not up to date. (PD-871)

6.3.10 0.12.9

- Reverted changes to the IMS indices that were made when constructing a `SwathMap` as this lead to downstream lossy data when, for example, creating in-silico dilutions. (PD-870)

6.3.11 0.12.8

- Added code to efficiently sub-sample toffee files to only include data for specifically requrested peptides. This is very useful for creating small files that can be used in downstream regression testing, without requiring GB of download. (PD-869)

6.3.12 0.12.7

- Added first step for visualisation – this is based on plotly and enables an interactive figure to be generated for a given peptide (transition group) with a specified number of isotopes. (PD-868)

6.3.13 0.12.6

- Added code to enable combining two toffee files where one serves as a background and peptides from the other are added with an 'in-silico' dilution at known retention times. This is extremely useful for testing purposes. (PD-867)

6.3.14 0.12.5

- Added ability to convert toffee to mzML using the `psims` library (PD-793)

6.3.15 0.12.4

- Renamed `SwathMapSummary` to `SwathMapInMemorySpectrumAccess`, and gave this a common base class with `SwathMapSpectrumAccess`
- Added function to return m/z transformer for `SwathMapInMemorySpectrumAccess` and `SwathMapSpectrumAccess`.
- Updated the example notebook that shows how to sub-sample a toffee file for just the iRT precursors.

6.3.16 0.12.3

- Added an uncompressed cache file for using with `SwathMapSpectrumAccess`. This gives a significant improvement in performance, as you are no longer uncompressing data, effectively meaning that HDF5 acts like a memmap.

6.3.17 0.12.2

- Small change to the IMS alpha calculation step. There are certain situations where numerical error will mean that the alpha value will flip-flop between iterations. This is caught and one value is accepted, a nicer error is thrown when that doesn't work.

6.3.18 0.12.1

In general, we have switched away from using least squares to calculate alpha and beta in favour of the more robust direct method prototyped using python – the results of this prototyping are currently being used in the preparation of the Toffee manuscript and will be included as supplementary material. There is a regression test that compares the results of the python code to this C++ implementation to ensure they are equivalent.

This now enables us to store alpha and beta on a "per scan" basis and thus get lossless compression between m/z and the integer index space. The file format version has been bumped to `v1.1`, although it remains backwards compatible to `v0.2`.

Toffee data can now be loaded in three modes:

- `SwathMap` which uses the median values for alpha and beta and enables the user to slice the raw data like an image. Using the library in this manner results in a 2-5 ppm mass accuracy loss as alpha and beta do not vary across retention time.
- `SwathMapSummary` where you can only access the data to quickly produce plots such as `totalIonChromatogram`. This mode can only be used on files created with a format `>= v1.1`.
- `SwathMapSpectrumAccess` where you can only access the data scan-by-scan in a manner akin to how one would read an mzML or wiff file. Using the library in this mode is essentially lossless (ppm error < 1e-6), at the cost of not being able to extract data by slicing through the mass over charge axis. This mode can only be used on files created with a format `>= v1.1`.

All of these modes are loaded through the `SwathRun` object as before, and there is no reason that the same algorithm cannot make use of both depending on need. They are const correct, and so will play nicely in shared memory parallelism.

6.4 0.11

6.4.1 0.11.1

- Changed the method for calculating the IMS coords to be more accurate via Levenberg-Marquardt non-linear least squares (PD-800)
- Version of toffee library used to create a file now stored as a parameter

6.5 0.10

6.5.1 0.10.7

- Added ability to convert toffee SawthMap back to raw data (PD-793)

6.5.2 0.10.6

- Fixed duplicate m/z IMS coordinates bug (PD-749)

6.5.3 0.10.5

- Fixed IMS gamma underflow bug

6.5.4 0.10.4

- Fixed IMS gamma off-by-one error that could occur when looking at the lowest m/z value in a window

6.6 License

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- search

INDEX

T