diapysef Documentation

Röst Lab

Mar 13, 2019

INSTALLATION

diapysef is a convenience package for working with DIA-PASEF data. It has functionalities to convert Bruker raw files into a format that OpenMS can understand. Thus OpenSwath can be used to analyze the data and TOPPView can

be used to visualize.

The diapysef package contains python bindings for the processing and analysis of mass spectrometry data coupled to TIMS-TOF based proteomics. It provides open-source access to algorithms specifically designed for DIA (data-independent acquisition) experiments. These bindings include algorithms that allow conversions of raw data (Bruker tdf.d), generation of spectral library, basic signal-processing (compression, filtering), and simple visualization of the raw data.

Note: The current documentation relates to the 0.3.3 version of diapysef. Note: Please acquire the Bruker tdf-sdk through official distributions.

CHAPTER

ONE

DATA ACCESS

After installation of diapysef and all of its dependencies, make sure that the Bruker sdk file is in your current working directory, and you should be able to import the package or call the scripts.

```
import diapysef
```

should import all the functions in diapysef, which allows access to the tdf data.

Succefully importing diapysef should result in the following output:

```
Found Bruker sdk. Access to the raw data is possible.
```

Otherwise, if it results in:

```
Bruker sdk not found. Some functionalities that need access to raw data will
not be available. To activate that functionality place libtimsdata.so (Linux)
or timsdata.dll in the src folder.
```

Please put a the Bruker sdk file in the working directory or make a symlink.

1.1 Getting Help

To get more detailed description of the available functions in diapysef, you can use the python function `help`.

```
>>> import diapysef
>>> help(diapysef)
...
...
```

CHAPTER

TWO

DATA STRUCTURE

The functions of diapysef can either be used in a python shell, or directly executed as a commandline tool.

diapysef is able to read directly into the sqlite-based .tdf raw files and store information about the acquisition methods.

```
import diapysef as dp
dia = pd.TimsData(pasefdata)
win = dia.get_windows()
win.to_csv("window_layout.csv")
print("File Written")
```

Alternatively, the function can be called directly in command line.

```
get_dia_windows.py pasefdata outputfile
```

With our pipeline using MaxQuant DDA data for our spectral library generation, we need to convert the ion mobility units to 1/K0 values for standardized measure. The MaxQuant output files evidence and all_peptides have ion mobility values in scan numbers, which correspond to the ion mobility 1/K0 values.

Converting the scan numbers to 1/K0 values require a raw .tdf file that contains the calibration information.

```
import diapysef as dp
pas = dp.PasefData(pasefdata)
mq = dp.PasefMQData(MQout_directory)

mq.get_all_peptides()
mq.annotate_ion_mobilitty(pas)
all_pep = mq.all_peptides
all_pep.to_csv("all_peptides_1K0.csv")

mq.get_evidence
mq.annotate_ion_mobility(pas)
ev = mq.evidence
ev.to_csv("evidence_1K0.csv")
```

Alternative, these functions can be called in command line:

```
annotate_mq_ionmobility.py MQOUT_director Pasefdata output_prefix
```

The above function writes a simple csv file that contains information of the pasef run including isolation window width, isolation window starts and ends, ion mobility drift time starts and ends, collision energy, etc. With these information, we can plot the window placements in the dimensions of m/z and ion mobility.

```
import diapysef as dp
import pandas as pd
```

(continues on next page)

(continued from previous page)

```
dia = dp.TimsData(pasefdata)
win = pd.read_csv("window_layout.csv")
dp.plot_window_layout(windows = win)
```

If you have a MaxQuant output of the library precursors, you can also map the precursors with the windows.

```
import diapysef as dp
import pandas as pd

dia = dp.TimsData(pasefdata)
win = pd.read_csv("window_layout.csv")
precursors = pd.read_csv("all_peptides.csv")
dp.plot_window_layout(windows = win, precursor_map = precursors)
```

Or alternatively, the function can be called in command line.

```
plot_dia_windows.py window_layout.csv annotated_all_peptides_1K0.csv
```

Note: Precursor map is the allPeptides.csv file generated from MaxQuant and needs to be properly annotated with 1/K0 values.

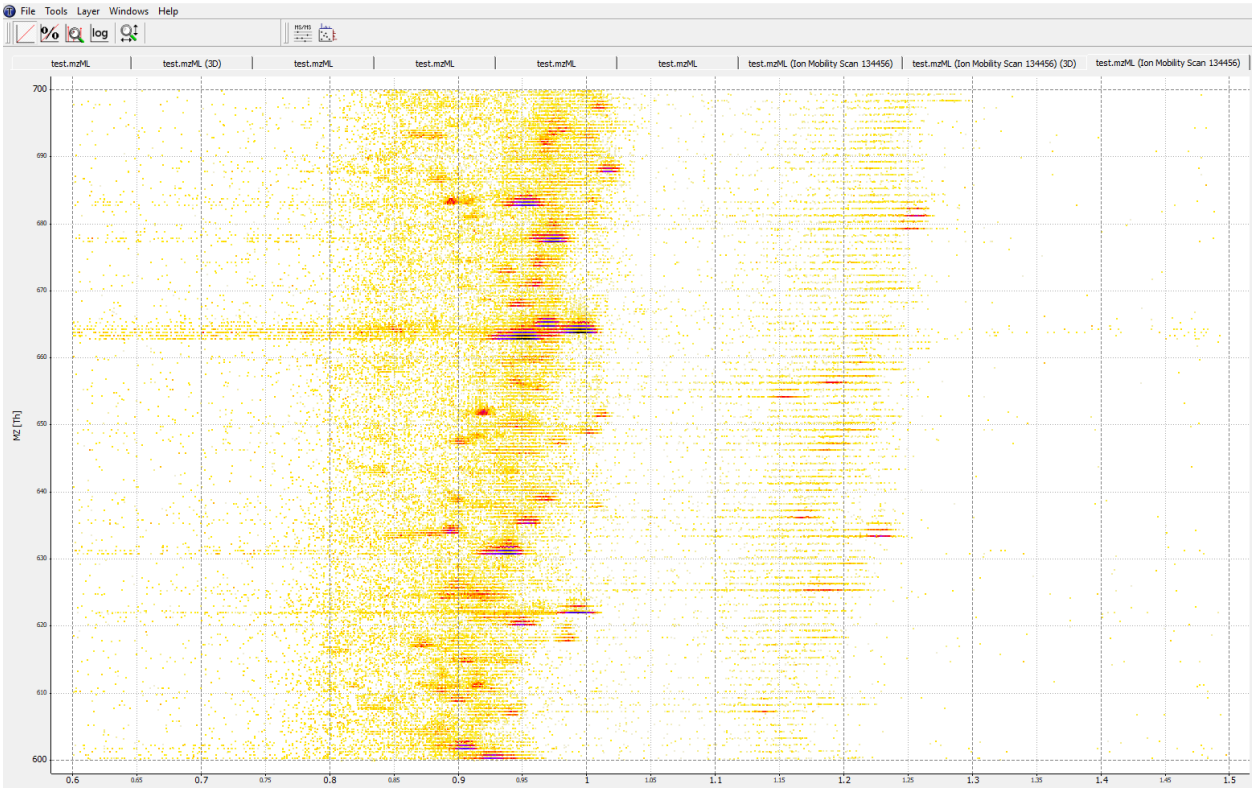
CHAPTER

THREE

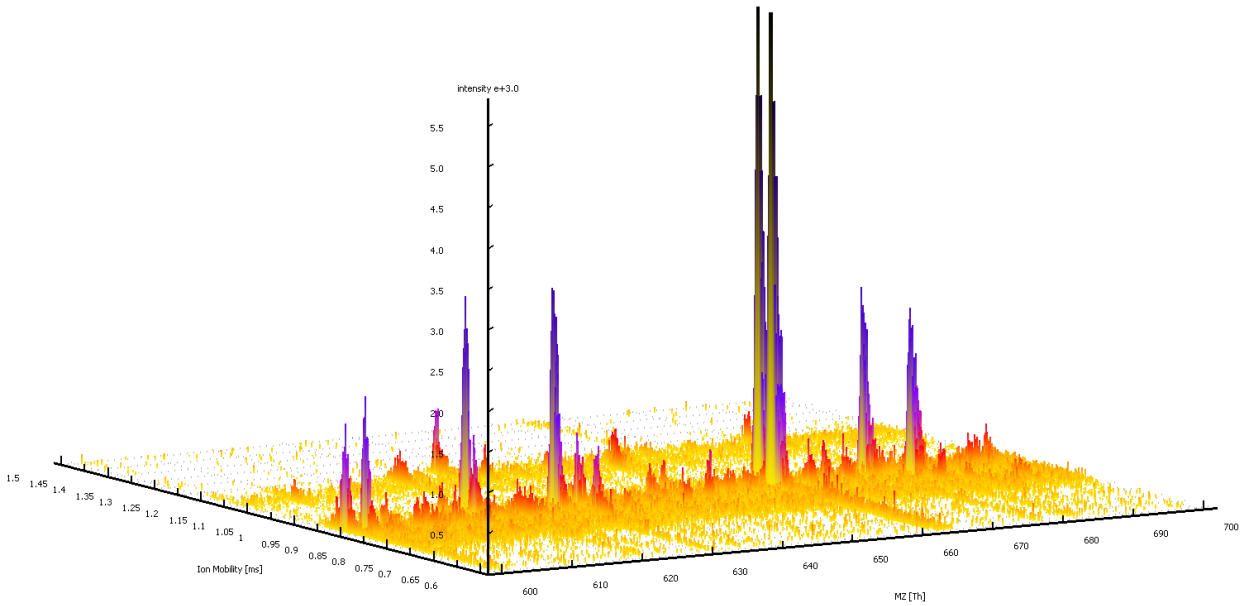
DATA VISUALIZATION

Other than using the `plot_window_layout` function in the package, the data can also be visualized using OpenMS tools after conversion to mzML files.

OpenMS 2.4.0 version is capable of visualizing data with ion mobility values. Using `ToppView`, you can now visualize the four dimensional data through slicing it by retention time and look at the MS spectrum.



This displays the map at retention time 4479 seconds, the features in a 2D map with ion mobility and m/z. Switching it into 3D view, you can see the intensity of the precursors.



OpenMS is now capable of handling ion mobility data with all of its previous usages remained functional.

CHAPTER

FOUR

LIBRARY GENERATION

For most data-independent acquisition (DIA) analysis, a well-represented spectral library is required for precursors, peptide, and protein identifications. Currently, we acquire our library through using deeply fractionated sample runs that are analyzed in MaxQuant.

The general steps for generating the spectral library are, annotating ion mobility values in the MQ output, correcting retention time and ion mobilty values against iRT peptides, formatting it to OpenSwath read-able format, generate the spectral library formats (.TraML, .pqp, .tsv) with OpenSwath.

4.1 Annotating Ion Mobility

Generating the library requires the DDA output files from MaxQuant. * msms.txt * allPeptides.txt * evidence.txt

Before generating the assay library, the ion mobility values have to be converted from scan numbers in MaxQuant output to the standardized 1/K0 units.

Follow the instructions at :doc: *convenientfunctions* for annotating and writing out files with standardized units.

4.2 Correcting RT and IM values

For correcting the retention time and ion mobility values, an iRT peptide library file including ion mobility dimension is required. Having the MaxQuant output directory (annotated with ion mobiltiy 1/K0) and iRT file, we can generate an OpenSwath readable format of library.

```
import diapysef as dp
import pandas as pd

# read in the MQOUT files
msms = pd.read_csv("mqout_dir/msms.txt", sep = "\t")
# read in the annotated evidence file
evidence = pd.read_csv("mqout_dir/annotated_evidence.csv")
irt = "irt_file.tsv"

# Alignment of RT and IM values
ptsv = dp.pasef_to_tsv(evidence, msms, irt_file = irt, \
im_column="IonMobilityIndexK0", rt_alignment="nonlinear", im_alignment="linear")

# save the library table
ptsv.to_csv(ptsv, "mqout.tsv", sep = "\t", index = False)
```

The function allows a few options for the alignment of retention time and ion mobility values. The input `rt_alignment` has options for `nonlinear` for a lowess alignment, `linear` alignment, and `None`. For

im_alignment, it contains options of linear or None. For standard LC-MS/MS analysis, we suggest alignment using nonlinear for rt_alignment and linear for im_alignment.

Alternatively, we also have a script as a commandline tool that can be called with:

```
python create_library.py
```

For details and options of the script, simply type:

```
python create_library.py --help
```

4.3 Generating Assay Library

After generating the library with corrected retention time and ion mobility values, a standardized assay library can be generated through OpenSwath applications.

4.3.1 Standardize Assay Library

First, the library table can be converted into standard assay formats (.TraML, .pqp, .tsv) using OpenSwathAssayGenerator.

4.3.2 Inputs

OpenSwathAssayGenerator requires several inputs parameters. – in : Input file of library transitions – out: Output file with valid extensions – swath_windows_file: File contains isolation window information

An example would be this:

```
OpenSwathAssayGenerator -in mqout.tsv -out hela_assaylib.TraML \
                        -swath_windows_file window_setting.txt
```

4.3.3 Generate Decoy

The assay library can add decoy peptides for statistical validation of scores and identification. It can be done through OpenSwathDecoyGenerator.

4.3.4 Inputs

OpenSwathDecoyGenerator requires several input parameters. – in: Input file of the target assay library – out: Output file of the target-decoy library – method: Method of generating the decoy – switchKR: Boolean of switching the termini of the decoy peptides

An example would be this:

```
OpenSwathDecoyGenerator -in hela_assaylib.TraML \
                        -out hela_target_decoy_assaylib.TraML \
                        -method pseudo-reverse \
                        -switchKR true
```

After generating the target decoy library, the assay library file is ready for the input for OpenSwathWorkflow parameter, tr.

CHAPTER

FIVE

DATA CONVERSION

The generated .tdf files from DIA pasef runs can be converted to standard formats (mzML) with diapysef.

Using the script `convertTDFtoMzML.py` can convert the .tdf file to a single mzML file. It allows merging of frames for the same precursors, filtering of range of frames, splitting of files by overlapping window settings, and compression of data with PyMSNumpress.

5.1 Inputs

– `-a`: Analysis directory of the raw data (.d) (Required) – `-o`: Output filename (Required) – `-m`: Number of frames for merging – `-overlap`: Number of overlapping windows – `-r`: Range of frames to convert

For detailed options and descriptions, simple type:

```
convertTDFtoMzML.py --help
```

5.2 Example

```
data_dir=diaPasef_run.d
output_file='diaPasef_run.mzML'

convertTDFtoMzML.py -a=$data_dir -o=$output_file
```

The converted mzML files can be processed with the assay library in `OpenSwathWorkflow`.

CHAPTER

SIX

QUANTIFICATION AND IDENTIFICATION

Using the assay library and the mzML files, identification and quantification of peptides can be performed with `OpenSwathWorkflow` and `PyProphet`. For detailed description and documentation of the downstream analysis, please refer to their documentation website. The newest `OpenMS` version 2.4.0 includes functionalities in handling ion mobility informations. Here are some of the input parameters that are additional to the regular parameters.

6.1 Inputs

– `--ion_mobility_window`: Ion mobility extraction window of precursor – `--im_extraction_window_ms1`: Use ion mobility on MS1 level – `-irt_im_extraction_window`: iRT extraction of the ion mobilty correction values – `-use_ms1_ion_mobility`: Performs extraction on MS1 level ion mobility level – `-Calibration:ms1_im_calibration`: Use ms1 for ion mobility calibration – `-Calibration:im_correction_function`: Choose im correction function – `-Calibration:debug_im_file`: Record the ion mobility correction data – `-Scoring:Scores:use_ion_mobility_scores`: Add ion mobility for scoring

6.2 Output

`OpenSwathWorkflow` can generate `.tsv`, `.osw` for identification and scorign output. It is also capable of generating the chromatogram files with extension `.sqmass`. The quantified output `.tsv` and `.osw` can be statistically validated with `PyProphet`.

6.3 Statistical Validation

`PyProphet` can take the scores generated from `OpenSwathWorkflow` and statistically validate the precursor identifications. For detailed documentation, please refer to the website.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex

7.1 Acknowledgements

The tools and workflows are collaboratively developed with the data acquired by the Mann Group at MPI, Bremen , the Aebersold Group at IMSB, ETH Zurich, and Bruker Daltonics. The core pipeline is also referenced from OpenMS and PyProphet, and msproteomicstools.