***Fragariyo (working title as of July, 2022)***

Files:

* terminalFragmentor\_Main.py
* InternalFragmentor\_Main.py
* Modifications.py
* OutputAnalysis\_v2.py
* OutputIonTypeAnalysis.py
* OutputPropAnalysis.py
* Parameter\_Parser.py
* Parameter\_Parser\_terminal.py
* PeakMatch.py
* RenameIMTBXoutputs.py
* TerminalFragments\_template\_example.csv
* InternalFragsments\_template\_example.csv
* DatabaseXplorer.py (it can be used to print out theoretical databases for either terminal or internal fragments)

# Instructions

Fragariyo is as set of Python3 scripts. They can be run in Spyder or PyCharm. I cannot be run thru terminal…yet.

## Libraries installed when running the last version of Fragariyo

Package Version

----------------------------- -----------

aiohttp 3.8.1

aiosignal 1.2.0

alabaster 0.7.12

altgraph 0.17.2

appdirs 1.4.4

apptools 5.1.0

argcomplete 2.0.0

asteval 0.9.26

async-timeout 4.0.2

asynctest 0.13.0

attrs 21.4.0

Babel 2.9.1

backcall 0.2.0

brain-isotopic-distribution 1.5.9

cachey 0.2.1

certifi 2021.10.8

cffi 1.15.0

charset-normalizer 2.0.10

cloudpickle 2.0.0

colorama 0.4.4

configobj 5.0.6

cycler 0.11.0

dask 2021.12.0

debugpy 1.5.1

decorator 5.1.0

docstring-parser 0.13

docutils 0.17.1

entrypoints 0.3

envisage 6.0.1

et-xmlfile 1.1.0

freetype-py 2.2.0

frozenlist 1.2.0

fsspec 2021.11.1

future 0.18.2

HeapDict 1.0.1

hsluv 5.0.2

idna 3.3

imageio 2.13.5

imagesize 1.3.0

importlib-metadata 3.4.0

importlib-resources 5.4.0

ipykernel 6.6.1

ipython 7.31.0

ipython-genutils 0.2.0

IsoSpecPy 2.2.0

jedi 0.18.1

Jinja2 3.0.3

joblib 1.1.0

jsonschema 4.3.3

jupyter-client 7.1.0

jupyter-core 4.9.1

kiwisolver 1.3.2

lmfit 1.0.3

locket 0.2.1

magicgui 0.3.4

Mako 1.1.6

MarkupSafe 2.0.1

matplotlib 3.3.0

matplotlib-inline 0.1.3

mayavi 4.7.4

mpmath 1.2.1

multidict 5.2.0

napari 0.4.12

napari-console 0.0.4

napari-plugin-engine 0.2.0

napari-svg 0.1.5

nest-asyncio 1.5.4

networkx 2.6.3

numpy 1.21.4

numpydoc 1.1.0

obonet 0.2.5

openpyxl 3.0.9

packaging 21.2

pandas 1.3.4

parso 0.8.3

partd 1.2.0

PeakUtils 1.3.3

pefile 2021.9.3

pickleshare 0.7.5

Pillow 8.4.0

Pint 0.18

pip 20.1.1

pooch 1.5.2

prompt-toolkit 3.0.24

psutil 5.9.0

psygnal 0.2.0

pycparser 2.21

pydantic 1.9.0

pyface 7.3.0

Pygments 2.11.2

pygubu 0.20

pygubu-designer 0.24

pyinstaller 4.2

pyinstaller-hooks-contrib 2022.1

PyOpenGL 3.1.5

pyparsing 2.4.7

PyQt5 5.15.6

PyQt5-Qt5 5.15.2

PyQt5-sip 12.9.0

pyrsistent 0.18.0

pyteomics 4.5

pythoms 2.2.8

python-dateutil 2.8.2

pytz 2021.3

PyWavelets 1.2.0

pywin32 303

pywin32-ctypes 0.2.0

PyYAML 6.0

pyzmq 22.3.0

qtconsole 5.2.2

QtPy 2.0.0

requests 2.27.1

scikit-image 0.19.1

scikit-learn 1.0.2

scipy 1.6.0

seaborn 0.11.2

setuptools 47.1.0

six 1.16.0

sklearn 0.0

snowballstemmer 2.2.0

Sphinx 4.3.2

sphinxcontrib-applehelp 1.0.2

sphinxcontrib-devhelp 1.0.2

sphinxcontrib-htmlhelp 2.0.0

sphinxcontrib-jsmath 1.0.1

sphinxcontrib-qthelp 1.0.3

sphinxcontrib-serializinghtml 1.1.5

superqt 0.2.5.post1

sympy 1.9

threadpoolctl 3.1.0

tifffile 2021.11.2

toolz 0.11.2

tornado 6.1

tqdm 4.62.3

traitlets 5.1.1

traits 6.3.2

traitsui 7.2.1

typing-extensions 4.0.1

uncertainties 3.1.6

urllib3 1.26.7

vispy 0.9.4

vtk 9.1.0

wcwidth 0.2.5

wrapt 1.13.3

wslink 1.3.1

yarl 1.7.2

zipp 3.7.0

# Match terminal fragments first

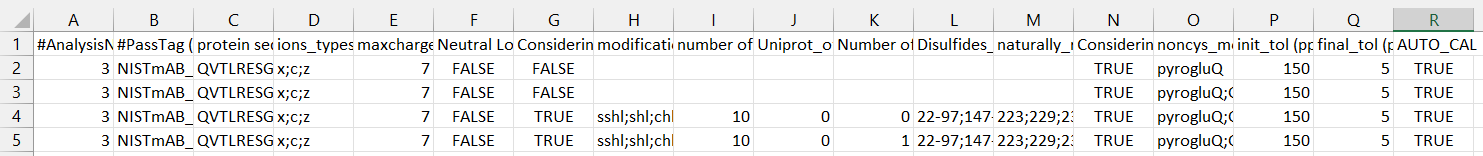
## terminalFragmentor\_Main.py

* + 1. Run main\_batch\_multipass function
    2. Attribute iso = True. Runs the module. It is used to renamed .isotopes files produced from Waters data. The .isotopes files have a generic name inside a folder named based on the file, but the .isotopes files will be renamed based on their sample folder.
    3. Choose Peaklist Files (File types acceptable are):
       1. .isotopes files from IMTBX/Grppr for Waters and from Mobilitron for Agilent files
    4. Upload .ions (a previously produced theoretical ion database) files when prompted to, if a theoretical database has been created.
    5. Choose Output Folder
    6. Load the Parameter file (see TerminalFragments\_template\_example.csv to learn how to fill a parameter file).

Text

Description automatically generatedFragmentation will start. The analysis number will appear as well as the current pass (PassName). An analysis is a group of passes.

## Terminal Fragment Parameter File



1. Analysis number
2. Name for the pass (it can be something informative about the pass e.g. ion type, mutant, etc)
3. One-letter-code protein sequence
4. Ion types (by inputting z-dot and c-dot ions are automatically included)
5. Maximum charge state to be used
6. Neutral losses bool. TRUE will considered neutral losses.
7. Disulfide bond bool. TRUE makes the script to determine the amount of disulfide bonds and where they are and to separate them if the two cysteine partners end up in different fragments
8. What modifications to use for the reduced cysteines
   1. sshl;shl;chhsshl;hl;sh;h
9. What is the maximum number of modifications to be used for the cysteine modifications (maximum number of cysteines that can be reduced)
10. Uniprot\_offset, int. Number that the sequence index must be offset to match the disulfide bond positions , if these positions were obtain from the uniport database. Put “0” is no offset needed
11. Number of disulfide bonds to be broken even if their cysteines are in the same fragment
12. Disulfide bond list
    1. Cys1\_index-Cys1partner\_index; Cys2\_index-Cys2partner\_index (e.g NISTmAb disulfide bonds: 22-97;147-203;223-1000;229-1000;232-1000;264-324;370-428).
13. Index for cysteines that do not participate in any disulfide bonds.
14. Considering modifications (not involved in disulfide bonds)
15. Modifications list (separate modifications with semi colons (e.g. NISTmAb modification for the n-term and glycan: pyrogluQ;GiiF)
    1. See Modifications.py section for more information
16. Initial error tolerance
17. Final error tolerance
18. Auto Calibration bool.If TRUE, Auto calibration will be performed using

## Terminal Fragment Results

Graphical user interface, application, table, Excel

Description automatically generated

The results are organized by termini and fragment site. The results are saved as.csv files and in binary .hits files (created with Pickle).

# Match internal fragments

## InternalFragmentor\_Main.py

1. Run main\_batch\_multipass()
2. Table

   Description automatically generatedChoose Peaklist Files (in-house .csv file). Files can be obtained from unmatched files obtained from matching terminals or they can be created from ions obtained from other sources.
   1. Neutral mass of experimental ion obtained from charge (z) and m/z values
   2. Charge
   3. C
   4. Mass-to-charge ratio
   5. Values *m/z* corresponding to the peak isotope envelope
   6. Values *intensity* corresponding to the peak isotope envelope
3. Choose Output Folder when prompted
4. Load .ions file, if a previous created database will be used
5. Graphical user interface, application

   Description automatically generatedChoose parameter file, if a new database needs to be created
   1. Analysis number
   2. Pass name (something descriptive about the pass (e.g ions searched for, mods searched for, etc)
   3. Sequence = Protein sequence one-letter code
   4. Iontypes allowed for internal fragments:
   5. Minimum charge to be considered
   6. Text

      Description automatically generated with medium confidenceMaximum charge to be considered (right now max and min are the same to keep the passes somehow smaller. In a future update hopefully this will be simplified).
   7. Minimum fragment length in amount of residues
   8. Maximun fragment length in amount of residues
   9. Modifications for non-cysteiens residues separate by semicolons (examples shoes an analysis with several passes where a different NISTmAb glycan is searched for).
   10. What modifications to use for the reduced cysteines
   11. What is the maximum number of modifications to be used for the cysteine modifications (maximum number of cysteines that can be reduced)
6. Disulfide bond bool. TRUE makes the script to determine the amount of disulfide bonds and where they are and to separate them if the two cysteine partners end up in different fragments
7. Uniprot\_offset, int. Number that the sequence index must be offset to match the disulfide bond positions , if these positions were obtain from the uniport database. Put “0” is no offset needed
8. Number of disulfide bonds to be broken even if their cysteines are in the same fragment
9. Disulfide bond list
   1. Cys1\_index-Cys1partner\_index; Cys2\_index-Cys2partner\_index (e.g NISTmAb disulfide bonds: 22-97;147-203;223-1000;229-1000;232-1000;264-324;370-428).
10. Index for cysteines that do not participate in any disulfide bonds.
11. Script will run ( a protein a large as BSA took several days and breaking the analysis down to several pieces)

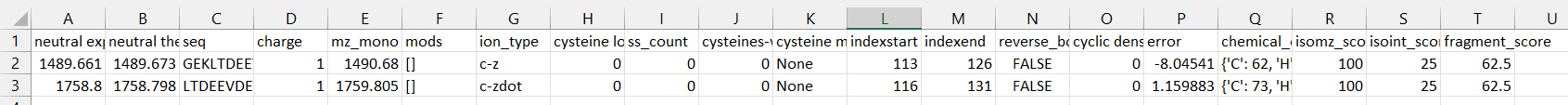
Text

Description automatically generated

## Internal Fragment Results

The results will be saved in a folder created with the name of the experimental ion .csv file: The folder contains:

1. A .theofrags file (a.ka. .ions file) a binary version of the theoretical database
2. A.txt.frags. A human readable version of the theoretical databased (they can be quite large). To be removed.
3. Chart, histogram

   Description automatically generatedPNG files showing the matching of the theoretical isotopic envelope and experimental isotopic envelope
4. TSV files (tab-delimited files) containing the details of the matched peaks
   * + - 1. Neutral mass (experimental ion)
         2. Neutral mass (theoretical ion)
         3. Fragment sequence
         4. Charge
         5. Monoisotopic *m/z* value
         6. Modifications
         7. Ion type
         8. Indices where the cysteines are located
         9. Number of disulfide bonds in internal fragment
         10. How many cysteines are being modified
         11. Cystine modification if disulfide bonds were reduced
         12. Index where the fragment starts
         13. Index where the fragment ends
         14. Bool. If TRUE the sequence is a reverse sequence. To be used as FDR (False Discovery Rate) calculations in the future.
         15. Cyclic density (number of cyclic regions form by disulfide bonds/ total residue number)
         16. Matching error
         17. Chemical compositions (in python dictionary form = {'C': 44, 'H': 72, 'N': 14, 'O': 18, 'S': 0, 'Fe': 0})
         18. Scoring in *m/z*
         19. Scoring in *intensity*
         20. Compound score

# Data Analysis

Unfortunately, they are currently not in a user-friendly state

## Sequence Coverage Calculations

* OutputAnalysis\_v2.py

Text

Description automatically generated

To a function or line of code to be active, and therefore to be run, press “ctrl”+”/”. To inactive it also press “ctrl”+”/”.

* frips\_main\_seq\_cov(hitsfiles, seqcov=True, seqcov\_iontype=False, combination=False)
  + Outputs example: A graph showing sequence coverage (coordinate information to replicate this graph is saved as a file: samplename\_analysis.csv)
* Chart, histogram

  Description automatically generatedseqcov, if TRUE calculates seq coverage and color codes fragments based on terminal type
* seqcov\_iontype, if TRUE calculates seq coverage and color codes fragments based on ion type.
* combinations, if TRUE two files can be selected and combine. If False, several files can be chosen, but analysis will be done individually.
* The output is also saved for each file (or combination) analyzed in a .csv file.
* frips\_main\_seq\_cov(hitsfiles, termi\_frags\_stats=True)
* It will produce a terminalstats.csv file with the average length and standard deviation for each termini, as well as the number of fragments for each termini
* frips\_main\_seq\_cov(hitsfiles, intenanalysis=True)
* It will produce a CSV file with the m/z and intensity values of peaks divided into N and C terminus. If several files with the same peaks are input, a graph tracking the intensity of the fragments can be made with the file. The intensities columns have not column name. They are organized as they were selected in the file chooser…a point to fixed in the future.

Table

Description automatically generated

* merging\_interalfrag\_tsv(tsvfiles, avg = True)
* Function to merge desired .tsv files (output containing internal fragment matches). If the avg attribute is true, the resulting .tsv file will only contain the fragments that appeared in all the original files merged.
* ClipsMS\_FragmentorPipe(hitsfiles)
* Function to transform .hits file into a compatible input to fragment\_plotter function ().
* Output = filename\_clipsfig3.csv
* # inputfiles = filedialog.askopenfilenames(title='Load Hits Files', filetypes=[('CSV', '\_clipsfig3.csv'),('Fragmentor Hits', '.tsv')])  
  # fragment\_plotter(inputfiles, internal=False)
* based on ClipsMS graphing code
* Chart

  Description automatically generatedOutput:B. Terminal Fragments C. Internal Fragments (color has been added using Illustrator.
* #Reducing FDR files  
  # average\_main(batch=True)  
  # compare\_main()
* Created to handle the complicated search spaces encountered in FRIPS experiments (for terminal fragment .hits files only)
* Average main can be done in batch mode. Replicates files are combined and only the fragments that appeared in all files will be kept in the final file.
* Text, application

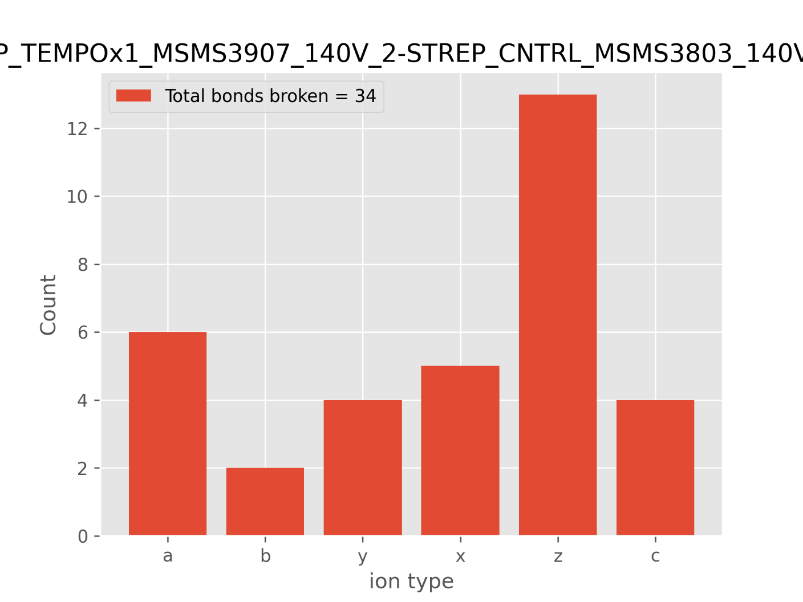
  Description automatically generatedA batch file is used for high throughput mode:

1. Path for the directory where the files to be analyzed are located
2. The name of the replicates. In the example there are there replicates.
3. Path of directory where the final average file is to be saved.
4. New name for output file

* Compare\_main…it removes common peaks in two files. The order is indicated by the prompt on the file dialog box when choosing files. Created in mind for comparing control vs FRIPS samples, but can be applied to other cases.

## Ion type Analysis

* A picture containing text

  Description automatically generatedOutputIonTypeAnalysis.py
* Ion\_analysis function creates bar graphs of the number of ions per ion type. The output file extension can be modified (‘.png’, .’svg’, and ‘pdf’. For the type allow see: https://matplotlib.org/stable/api/\_as\_gen/matplotlib.pyplot.savefig.html )
* Table

  Description automatically generatedMods\_analysis function outputs a CSV file (Mods\_analysis.tsvNeutralLosses\_analysis.tsv) containing information about the modifications found. They are organized by sample and per ion type. Next, they are organized by index. Needs to be worked to be more readable.
* Graphical user interface, application, table, Excel

  Description automatically generatedneutloss\_analysis function looks for neutral losses and organizes them as it was done for modifications. The output is saved in a file named “Mods\_analysis.tsv”.
* input\_PYMOL is a function that outputs a list of the indices where fragment sites are found. This list is in a format to be readily used by PYMOL. The first list is the output. The final list is the one A picture containing text

  Description automatically generatedobtained if replicates are input into the function, and only the common indices are outputted (occurs when attribute replicate is set to TRU).

For example, to color the resides in PYMOL just do take the function output (the string of number united by a plus sign). Write in PYMOL “color [color name], res [output list].

A picture containing graphical user interface

Description automatically generated

## Fragmentation Propensities

* Text

  Description automatically generatedOutputPropAnalysis.py
  + Graphical user interface, text

    Description automatically generatedOnly function used by author of this SOP is main\_frag\_propensities. It outputs, propensities of fragmentation for each residue after and before the residue.
  + Graphical user interface, application, table, Excel

    Description automatically generatedThe CSV files contain the following:
  + The image files contain (blue bars is propensitiy, orange graph I sfrequency of brakage):
  + Chart, histogram

    Description automatically generated

# Modifications Repository

* Text

  Description automatically generatedModifications.py
  + A dictionary that contains the information about modifications:
  + Dictionary keys: the string defined by the user to be the name of the modification. It must be compatible with pyteomics modX (https://pyteomics.readthedocs.io/en/latest/parser.html#modx). Pretty much no numbers allowed.
  + Text

    Description automatically generatedDictionary values: Modification object define in Modifications.py itself