seq-to-first-iso

Release 0.5.1

Lilian Yang-crosson, Pierre Poulain

Jul 10, 2019

CONTENTS

Version 0.5.1

Seq-to-first-iso computes the first two isotopologues intensity from peptide sequences.

It differentiates labelled and unlabelled amino acids with a 99.99 % 12C enrichment.

If you want to install seq-to-first-iso, use:

```
pip install seq-to-first-iso
```

CHAPTER

ONE

USER MANUAL

1.1 Theoretical background

1.1.1 Isotopologues in mass spectrometry

Depending on their isotopic composition, peptides have different isotopologues.

Element	Isotope	Relative abundance (%)
Hydrogen	H[1]	99.9885
Hydrogen	H[2]	0.0115
Carbon	C[12]	98.93
Carbon	C[13]	1.07
Nitrogen	N[14]	99.632
Nitrogen	N[15]	0.368
Oxygen	O[16]	99.757
Oxygen	O[17]	0.038
Oxygen	O[18]	0.205
Sulfur	S[32]	94.93
Sulfur	S[33]	0.76
Sulfur	S[34]	4.29

Stable isotopes of most common organic elements in peptides, values are taken from MIDAs[1]

The first isotopologue (noted ***M0***) contains peptides which elements are composed of only the **lightest stable isotopes**. In contrast the second isotopologue (***M1***) has one of its element with a **supplementary neutron**. Isotopologues follow notation *Mn* where *n* is the number of supplementary neutrons in the chemical formula compared to *M0*.

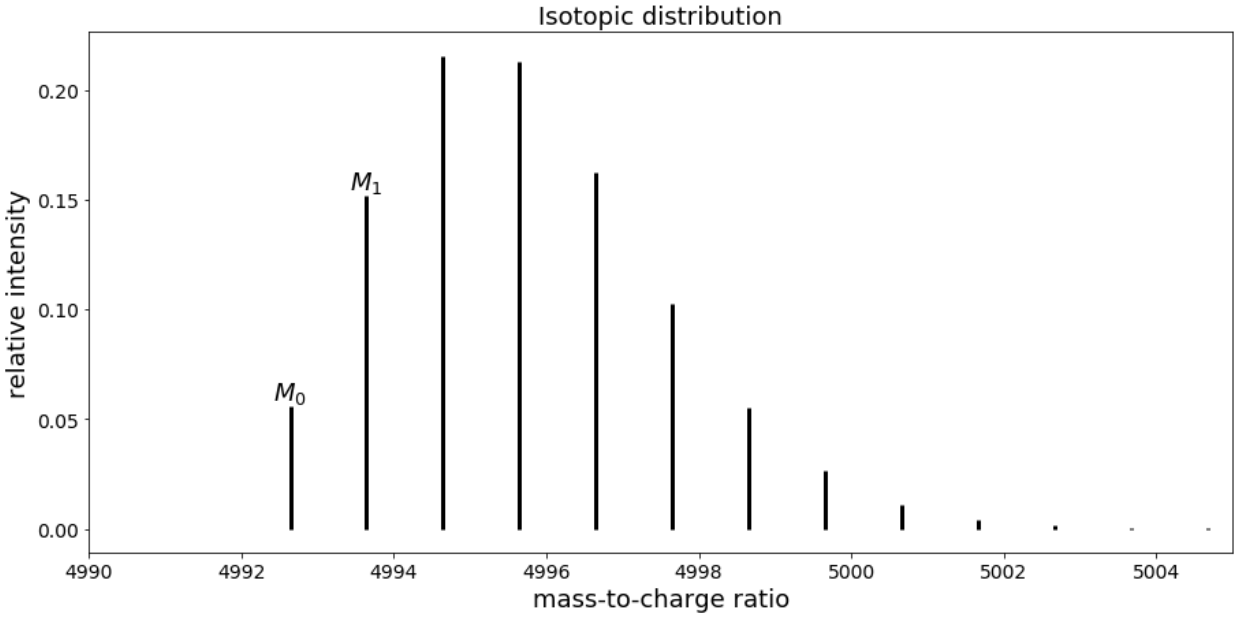
e.g: the chemical formula of Glycine is "C2H5O2N1", the different isotopic compositions for isotopologues *M0* and *M1* are described in the table below

Isotopologue	Carbon	Hydrogen	Oxygen	Nitrogen
M0	C[12]*2	H[1]*5	O[16]*2	N[14]*1
M1	C[12]*1, C[13]*1	H[1]*5	O[16]*2	N[14]*1
M1	C[12]*2	H[1]*4, H[2]*1	O[16]*2	N[14]*1
M1	C[12]*2	H[1]*5	O[16]*1, O[17]*1	N[14]*1
M1	C[12]*2	H[1]*5	O[16]*2	N[15]*1

Composition of M0 and M1 of Glycine

M0* can only have one composition**, meanwhile there are **multiple combinations for *M1, each with one of its elements swapped with a heavier isotope. The complexity of formulas increases even more with further isotopologues.

In high-resolution mass spectrometry, the mass spectrometer is able to differentiate peaks of isotopologues.



Mass spectrum of peptide with sequence "VGEVFINYIQRQNELFQGKLAYLIIDTCLSIVRPNDSKPLDNR"

In that case, the first peak corresponds to *M0* while the second one is *M1*, the third one *M2 etc*.

1.1.2 Computation of isotopologue intensity

Isotopologue intensity can be computed analytically. Formulas are adapted from an article by Wang, Benlian et al.[2].

We consider a peptide of formula

$$\mathrm{C}_{n(\mathrm{C})}\mathrm{H}_{n(\mathrm{H})}\mathrm{O}_{n(\mathrm{O})}\mathrm{N}_{n(\mathrm{N})}\mathrm{S}_{n(\mathrm{S})},$$

where *n(C)*, *n(H)*, *n(O)*, *n(N)* and *n(S)* denote the number of atoms of carbon, hydrogen, oxygen, nitrogen and sulfur respectively.

Compute M0 intensity

For such peptide, the normalized intensity of the monoisotopic ion is given by:

$$M_0 = a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})},$$

where *a(isotope)* is the relative abundance of the isotope.

Compute M1 intensity

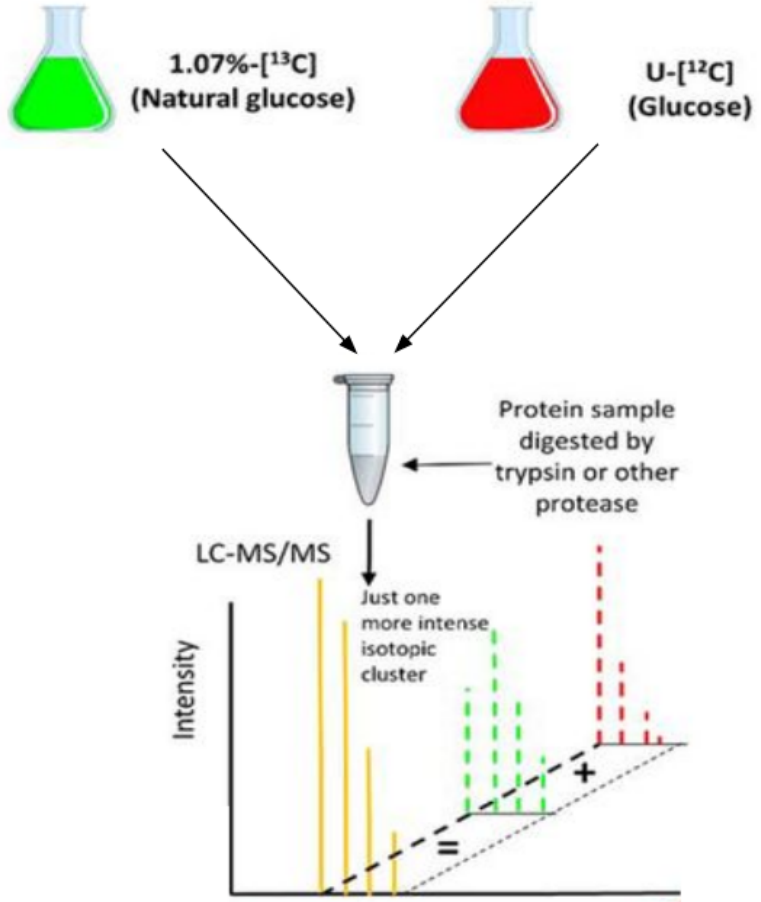
Following a polynomial expansion, intensity of the second isotopologue *M1* is:

$$\begin{aligned}
M_1 = {} & n(\mathrm{C}) \times a(\mathrm{C}[12])^{n(\mathrm{C})-1} \times a(\mathrm{C}[13]) \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \\
& + n(\mathrm{H}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})-1} \times a(\mathrm{H}[2]) \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \\
& + n(\mathrm{O}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})-1} \times a(\mathrm{O}[17]) \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \\
& + n(\mathrm{N}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})-1} \times a(\mathrm{N}[15]) \times a(\mathrm{S}[32])^{n(\mathrm{S})} \\
& + n(\mathrm{S}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})-1} \times a(\mathrm{S}[33])
\end{aligned}$$

We can observe that formulas follow a combinatorial explosion.

1.1.3 Quantify proteins with C[12] enrichment

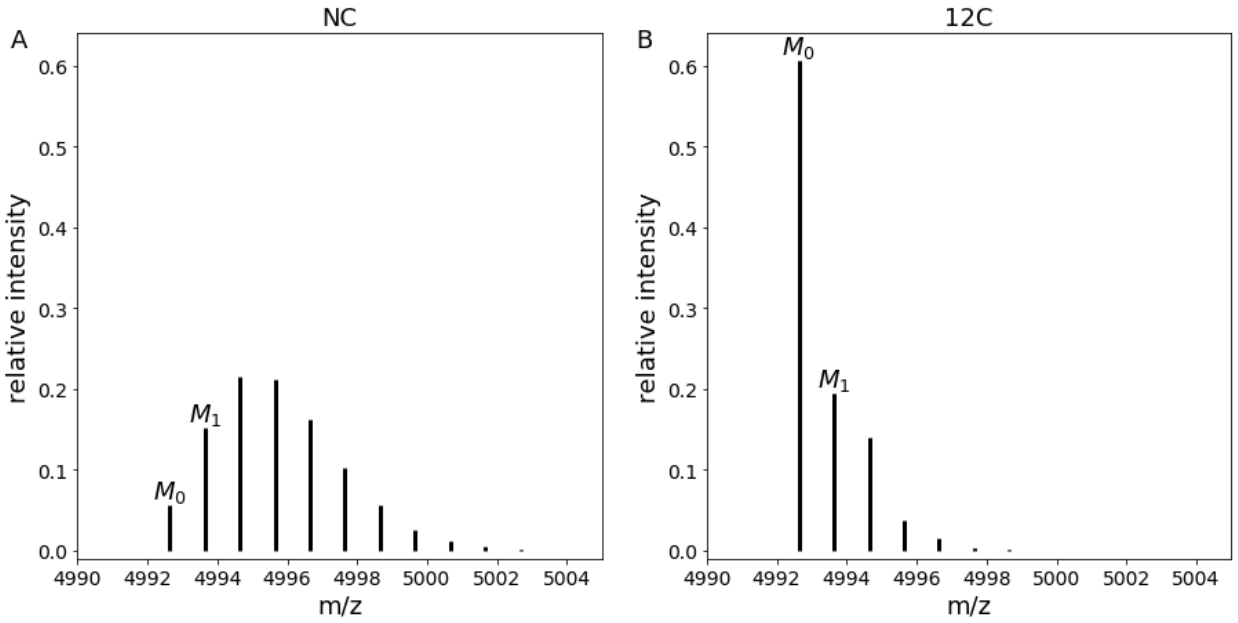
Simple Light Isotopic Metabolic Labeling (SLIM-labeling)[3] is a method developed by Léger et al. that allows **quantification of proteins via C[12] enrichment**. **Cells are provided** with a growth media containing **glucose enriched with C[12] at 99.99 % as the only carbon source**, the glucose is then assimilated by the cell to synthetize proteins which carbons have a C[12] abundance of 99.99 % instead of 98.93 %.



Usage of SLIM-labeling by combining two experimental conditions

SLIM-labeling allows the combination of proteins obtained in two different experimental conditions (Natural Carbon/Normal Condition **NC** and 99.99 % C[12] enrichment **12C**) in a single mass spectrometry run. By **comparing experimental and theoretical intensities** of adjacent isotopologues (in our case *M0* and *M1*), we are able to **get the ratio of proteins** between both conditions.

Another advantage of SLIM-labeling is that it increases the intensity of the first isotopologue, making it easier to detect.



Comparison of mass spectra in NC and 12C conditions for peptide with sequence "VGEVFINYIQRQNELFQGK-LAYLIIDTCLSIVRPNDSKPLDNR"

1.1.4 Strategy to take into account auxotrophies

A limitation of SLIM-labeling is that some organisms can have **auxotrophies to amino acids**, hence they cannot synthetize those for protein production. In this case, they **need to be provided with essential amino acids**. The problem is that 99.99 % C[12] enriched amino acids are not available nor are produced (as of 2019) thus **those essential amino acids will keep natural carbon abundance** (with 98.93 % C[12] and 1.07 % C[13]). Therefore, the formulas shown above become incorrect for SLIM-labeling (as abundances vary depending on wether the amino acid is labelled or not).

Seq-to-first-iso implements an algorithm that takes into account labelled and unlabelled amino acids for *M0* and *M1* computation.

To do so, we defined ***X*** as a virtual chemical element with **2 isotopes with abundance of natural carbon**. *X* can then be **substituted to the carbon of unlabelled amino acids** to compute correct isotopologue intensities.

New formulas were developed to take this new element into account:

$$M_0 = a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \times a(\mathrm{X}[12])^{n(\mathrm{X})}$$

and

$$\begin{aligned}
M_1 = {} & n(\mathrm{C}) \times a(\mathrm{C}[12])^{n(\mathrm{C})-1} \times a(\mathrm{C}[13]) \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \times a(\mathrm{X}[12])^{n(\mathrm{X})} \\
& + n(\mathrm{H}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})-1} \times a(\mathrm{H}[2]) \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \times a(\mathrm{X}[12])^{n(\mathrm{X})} \\
& + n(\mathrm{O}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})-1} \times a(\mathrm{O}[17]) \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \times a(\mathrm{X}[12])^{n(\mathrm{X})} \\
& + n(\mathrm{N}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})-1} \times a(\mathrm{N}[15]) \times a(\mathrm{S}[32])^{n(\mathrm{S})} \times a(\mathrm{X}[12])^{n(\mathrm{X})} \\
& + n(\mathrm{S}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})-1} \times a(\mathrm{S}[33]) \times a(\mathrm{X}[12])^{n(\mathrm{X})} \\
& + n(\mathrm{X}) \times a(\mathrm{C}[12])^{n(\mathrm{C})} \times a(\mathrm{H}[1])^{n(\mathrm{H})} \times a(\mathrm{O}[16])^{n(\mathrm{O})} \times a(\mathrm{N}[14])^{n(\mathrm{N})} \times a(\mathrm{S}[32])^{n(\mathrm{S})} \times a(\mathrm{X}[12])^{n(\mathrm{X}-1)} \times a(\mathrm{X}[13])
\end{aligned}$$

1.1.5 References

[1]: Alves, Gelio et al. "Molecular Isotopic Distribution Analysis (MIDAs) with adjustable mass accuracy." *Journal of the American Society for Mass Spectrometry* vol. 25,1 (2014): 57-70. doi:10.1007/s13361-013-0733-7

[2]: Wang, Benlian et al. "Isotopologue distributions of peptide product ions by tandem mass spectrometry: quantitation of low levels of deuterium incorporation." *Analytical biochemistry* vol. 367,1 (2007): 40-8. doi:10.1016/j.ab.2007.03.036

[3]: Léger, Thibaut et al. "A Simple Light Isotope Metabolic Labeling (SLIM-labeling) Strategy: A Powerful Tool to Address the Dynamics of Proteome Variations In Vivo." *Molecular & cellular proteomics : MCP* vol. 16,11 (2017): 2017-2031. doi:10.1074/mcp.M117.066936

1.2 KNIME configuration

You can install and use seq-to-first-iso with the Analytics platform KNIME to process data from SLIM-labeling.

Requirements:

- KNIME
- conda
- a little bit of Python knowledge

(Note: if you wish to use pandas > 0.23, you need to upgrade KNIME to 3.7.2 at least)

1.2.1 Set Python with KNIME

You need to install and configure a Python extension. This guide is adapted from the 3.7 Python installation guide from KNIME.

Set up the conda environment

On Windows, if you want to use conda with the default command-line interface *CMD*, you need to do some configuration, else use *Anaconda prompt* (or any other interface that recognizes conda) bundled with conda.

If conda was not added to the PATH environment variable during the installation, you have to configure your shell to use the `conda` commands:

- You can add the PATH to the folder containing conda, the command in *CMD* should look like:

```
set PATH=%PATH%;C:<PATH_WHERE_YOU_INSTALLED_CONDA>\Scripts
```

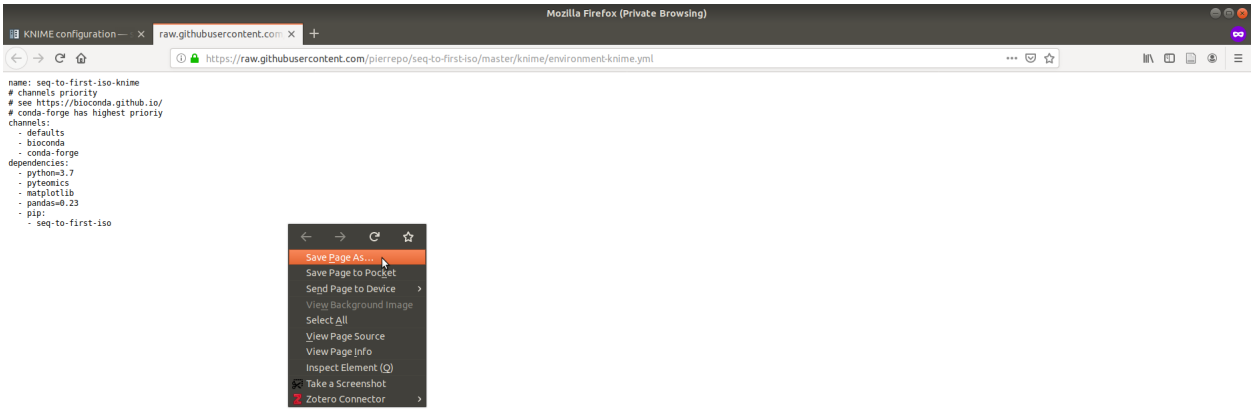
By default `<PATH_WHERE_YOU_INSTALLED_CONDA>` is `C:\Users\<Username>\<CONDA_INSTALLATION>` where `<Username>` is the Windows username and `<CONDA_INSTALLATION>` the conda installer used (e.g: "Miniconda3", "Anaconda4" ...). This command only works for the current command prompt and you will have to type it again if you want to use conda in a new command prompt.

- If you want *CMD* to always recognize conda, you will have to add conda to the system environment. One way to do it is with the command prompt:

```
setx PATH=%PATH%;C:<PATH_WHERE_YOU_INSTALLED_CONDA>\Scripts
```

Create a conda environment

In the GitHub repository, go to *knime/environment-knime.yml* and download the raw version of the file (*Right click* → *Save as* . . .).



Then in the directory where *environment-knime.yml* was downloaded open a shell/*Anaconda prompt* and use:

```
conda env create -f environment-knime.yml
```

Note: to move within folders with a shell use the `cd` *command.* Now if you do

```
conda env list
```

you should see `seq-to-first-iso-knime` in the environments.

Create a start script (KNIME <4.0)

For versions of KNIME lower than 4.0, a launch script is needed. Create a small script to start the conda environment by using the templates defined by KNIME. In our case `<ENVIRONMENT_NAME>` is `seq-to-first-iso-knime` while `<PATH_WHERE_YOU_INSTALLED_ANACONDA>` depend on the user's conda configuration and operating sytem.

To test if your script works, try to execute it:

- For Windows, double-click on it
- For Linux/Mac, make your file executable

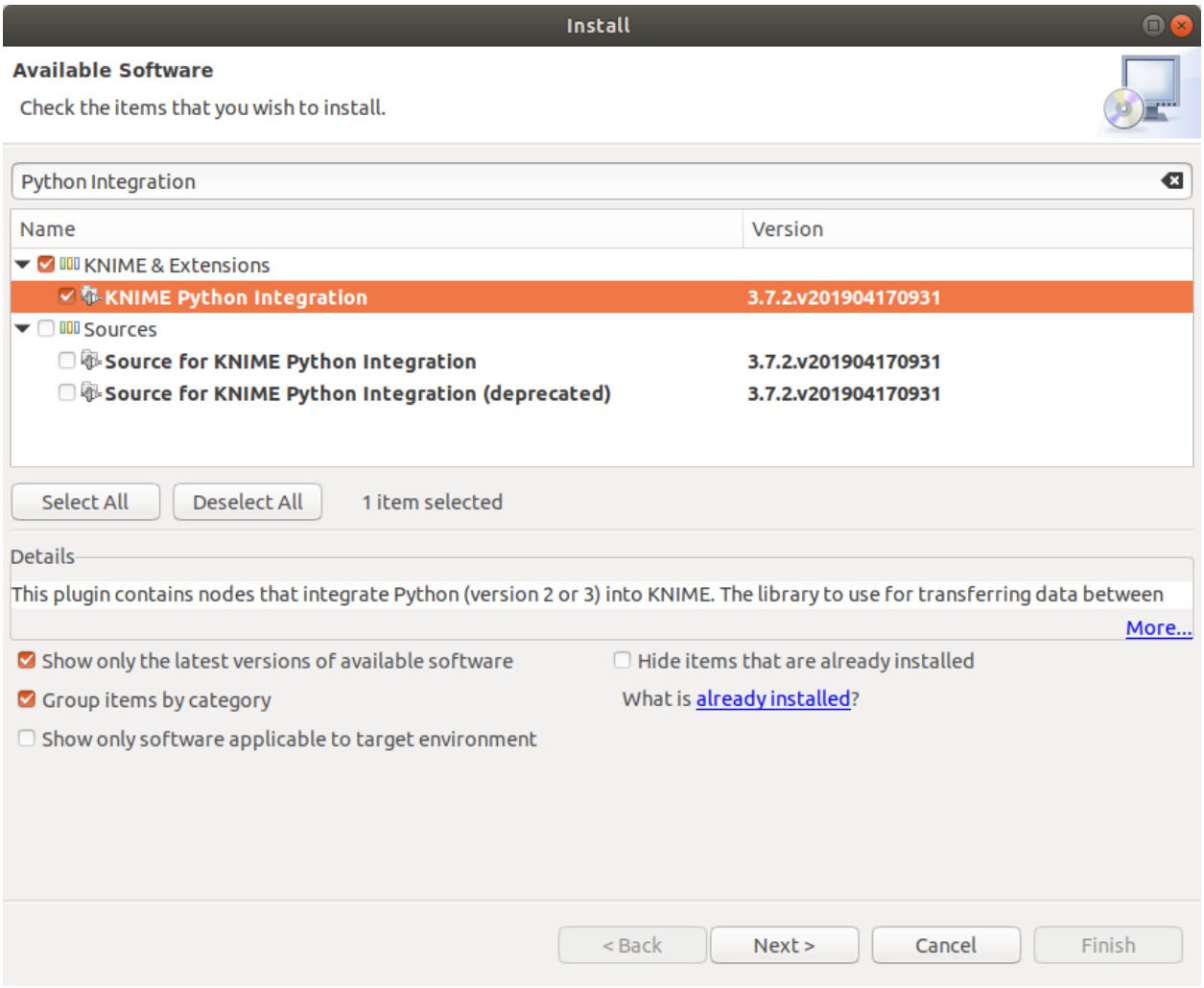
  ```
  chmod gou+x <script_name>
  ```

  then execute it with

```
./<script_name>
```

Configure the Python extension

In the KNIME interface, go to *File → Install KNIME Extensions* then search for *Python Integration* to find the KNIME Python Integration.



KNIME Python extension in the Install window

Now you just need to configure the Python executable KNIME will use. Go into *File → Preferences → KNIME → Python* then in the Python 3 subsection, paste the absolute path to your start script. *e.g: Windows path* `C:\Documents\<script_name>`, *Unix path* `home/<user>/<script_name>`.

For KNIME 4.X, you only need to write the path to the conda installation directory (your `<PATH_WHERE_YOU_INSTALLED_CONDA>`) and select *seq-to-first-iso-knime* in the dropdown menu for Python 3.

Next use Python 3 as default, then Apply and close.

If everything went alright, you should now be able to use Python script nodes with KNIME.

1.2.2 Use seq-to-first-iso with KNIME

Warning: the tutorial assumes you use seq-to-first-iso 0.5.0, functions in other versions might differ. If you want to use other versions, you have to make sure the changes across versions will not break your workflow.

The following steps are mostly examples, you can adapt them for your needs.

If the file is not parsed

You can use the parsing provided by seq-to-first-iso if you want to read from a formatted file (one sequence per line).

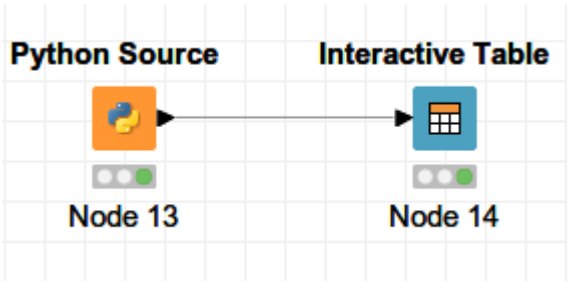
Create a Python scripting node by going in the Node Repository then *Scripting* → *Python* → *Python Source*. Then configure the node.

```
import seq_to_first_iso as stfi

# Parse the file, change <PATH_TO_YOUR_FILE> to your file path.
parsed_output = stfi.sequence_parser("<PATH_TO_YOUR_FILE>")
# Get rid of ignored_lines for seq_to_df.
parsed_output.pop("ignored_lines")

# List of unlabelled amino acids, change if you need.
unlabelled_aa = ["A", "C"]

# The name output_table will inform KNIME that
# the variable is an output table.
output_table = stfi.seq_to_df(unlabelled_aa=unlabelled_aa,
                              **parsed_output)
```



Minimal implementation of a Python node without prior input in KNIME

If peptides are already in a table

If you already have a table with peptide sequences, you can feed it to function seq_to_df() to get the output dataframe

Create a Python scripting node by going in the Node Repository then *Scripting* → *Python* → *Python Script (11)*. The node will receive a table as an input.

```
from seq_to_first_iso import seq_to_df

# Copy input table.
output_table = input_table.copy()

# Set the unlabelled amino acids.
unlabelled_aa = ["A", "C"]
```

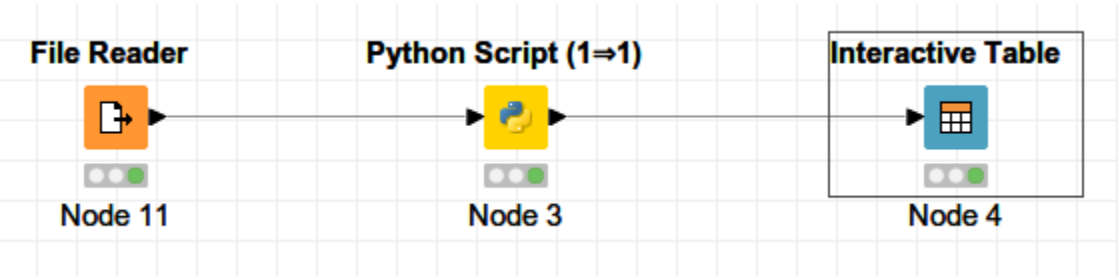
(continues on next page)

(continued from previous page)

```
# If the input table has more than 2 columns
# annotations will be the first column while
# sequences are the second one.
# Else, we consider the only column contains sequences.
try:
  sequences = output_table.iloc[:,1]
  # Cast pd.Series to list.
  annotations = list(output_table.iloc[:,0])
except:
  sequences = output_table.iloc[:,0]
  annotations = []

output_table = seq_to_df(sequences, unlabelled_aa, annotations=annotations)
```

In the example above, you may also chose which columns to take into account as arguments for seq_to_df(). *e.g: The fourth column of your table has lists of post-translational modifications, you can capture the column in a variable with* `ptms = list(output_table.iloc[:,3])` *then use the variable* `output_table = seq_to_df(sequences, unlabelled_aa, modifications=ptms)`.



Minimal implementation of a Python node with table input

CHAPTER

TWO

TUTORIAL

2.1 API of seq-to-first-iso

seq-to-first-iso computes the first two isotopologue intentities (M0 and M1) from peptide sequences with natural carbon and with 99.99 % 12C enriched carbon.

The program can take into account unlabelled amino acids to simulate auxotrophies to amino acids.

seq-to-first-iso is available as a Python module.

[1]:
```
from pprint import pprint

from pkg_resources import get_distribution  # Comes with setuptools.
import pandas as pd
from pyteomics import mass

import seq_to_first_iso as stfi
```

[2]:
```
try:
    print(f"pyteomics version: {get_distribution('pyteomics').version}")
except:
    print("pyteomics version not found")

print(f"pandas version: {pd.__version__}\n"
      f"seq-to-first-iso version: {stfi.__version__}"
     )
```

```
pyteomics version: 4.1.2
pandas version: 0.24.2
seq-to-first-iso version: 0.5.0
```

[3]:
```
# Variables used for showcase.
peptide_seq = "YAQEISRAR"
unlabelled_amino_acids = ["A", "R"]
```

2.1.1 Abundances defined in seq-to-first-iso

[4]:
```
isotopic_abundance = stfi.isotopic_abundance
C12_abundance = stfi.C12_abundance
```

[5]:
```
pprint(isotopic_abundance)
```

```
{'C[12]': 0.9893,
 'C[13]': 0.0107,
 'H[1]': 0.999885,
 'H[2]': 0.000115,
 'N[14]': 0.99632,
 'N[15]': 0.00368,
 'O[16]': 0.99757,
 'O[17]': 0.00038,
 'O[18]': 0.00205,
 'S[32]': 0.9493,
 'S[33]': 0.0076,
 'S[34]': 0.0429,
 'X[12]': 0.9893,
 'X[13]': 0.0107}
```

[6]:
```
pprint(C12_abundance)
```

```
{'C[12]': 0.9999,
 'C[13]': 9.999999999998899e-05,
 'H[1]': 0.999885,
 'H[2]': 0.000115,
 'N[14]': 0.99632,
 'N[15]': 0.00368,
 'O[16]': 0.99757,
 'O[17]': 0.00038,
 'O[18]': 0.00205,
 'S[32]': 0.9493,
 'S[33]': 0.0076,
 'S[34]': 0.0429,
 'X[12]': 0.9893,
 'X[13]': 0.0107}
```

isotopic_abundance and C12_abundance are dictionaries with abundances of common isotopes of organic elements. C12_abundance has a 12C abundance of 99.99 %, hence 13C abundance is 0.01 %.
Element X is a **virtual element** created to replace the carbon of unlabelled amino acids, it has the **same isotopic abundances as natural carbon**.

2.1.2 Separate sequences according to unlabelled amino acids

[7]:
```
help(stfi.separate_labelled)
```

```
Help on function separate_labelled in module seq_to_first_iso.seq_to_first_iso:

separate_labelled(sequence, unlabelled_aa)
    Get the sequence of unlabelled amino acids from a sequence.

    Parameters
    -------
    sequence : str
        String of amino acids.
    unlabelled_aa : container object
        Container (list, string...) of unlabelled amino acids.

    Returns
```

(continues on next page)

(continued from previous page)

```
    -----
    tuple(str, str)
        | The sequences as a tuple of string with:
        |    - the sequence without the unlabelled amino acids
        |    - the unlabelled amino acids in the sequence
```

[8]:
```
# Separate sequence "YAQEISRAR" with amino acids A and R unlabelled.
labelled_sequence, unlabelled_sequence = stfi.separate_labelled(peptide_seq,␣
↪unlabelled_aa=unlabelled_amino_acids)
print(f"sequence with labelled carbon: {labelled_sequence}\n"
      f"sequence with unlabelled carbon: {unlabelled_sequence}")
```

```
sequence with labelled carbon: YQEIS
sequence with unlabelled carbon: ARAR
```

2.1.3 Obtain a composition with element X

[9]:
```
help(stfi.seq_to_xcomp)
```

```
Help on function seq_to_xcomp in module seq_to_first_iso.seq_to_first_iso:

seq_to_xcomp(sequence_l, sequence_nl)
    Take 2 amino acid sequences and return the composition with X.

    The second sequence will have its C replaced by X.

    Parameters
    -------
    sequence_l : str or pyteomics.mass.Composition
        Sequence or composition with labelled amino acids.
    sequence_nl : str or pyteomics.mass.Composition
        Sequence or composition where amino acids are not labelled.

    Returns
    -----
    pyteomics.mass.Composition
        Composition with unlabelled carbon as element X.

    Notes
    ---
    | The function assumes the second sequence has no terminii (H-, -OH).
    | Supports pyteomics.mass.Composition as argument (0.5.1).
    | If mass.Composition objects are provided, the function assumes
      the terminii of the second composition were already removed.
```

[10]:
```
# Get the chemical formula with unlabelled carbon as element X.
chem_formula = stfi.seq_to_xcomp(labelled_sequence, unlabelled_sequence)
print(f"Composition of {peptide_seq} with {unlabelled_amino_acids} unlabelled:\n{chem_
↪formula}")
```

```
Composition of YAQEISRAR with ['A', 'R'] unlabelled:
Composition({'H': 76, 'C': 28, 'O': 15, 'N': 16, 'X': 18})
```

[11]:
```
# If all amino acids are labelled, you can pass an empty string.
labelled_formula = stfi.seq_to_xcomp(labelled_sequence, "")
print(f"Composition of {peptide_seq} with {unlabelled_amino_acids} unlabelled:\n
↪{labelled_formula}")
```

```
Composition of YAQEISRAR with ['A', 'R'] unlabelled:
Composition({'H': 42, 'C': 28, 'O': 11, 'N': 6})
```

[12]:
```
# You can also provide pyteomics.mass.Composition objects.
# To have a coherent result, H- -OH terminii should only be on one of the sequences.
labelled_composition = mass.Composition(labelled_sequence)
# parsed_sequence does not add the terminii.
unlabelled_composition = mass.Composition(parsed_sequence=unlabelled_sequence)

chem_formula = stfi.seq_to_xcomp(labelled_sequence, unlabelled_composition)
print(f"Composition of {peptide_seq} with {unlabelled_amino_acids} unlabelled:\n{chem_
↪formula}")
```

```
Composition of YAQEISRAR with ['A', 'R'] unlabelled:
Composition({'H': 76, 'C': 28, 'O': 15, 'N': 16, 'X': 18})
```

2.1.4 Compute isotopologue intensity

[13]:
```
help(stfi.compute_M0_nl)
print("-" * 79)
help(stfi.compute_M1_nl)
```

```
Help on function compute_M0_nl in module seq_to_first_iso.seq_to_first_iso:

compute_M0_nl(f, a)
    Return the monoisotopic abundance M0 of a formula with mixed labels.

    Parameters
    -------
    f : pyteomics.mass.Composition
        Chemical formula, as a dict of counts for each element:
        {element_name: count_of_element_in_sequence, ...}.
    a : dict
        Dictionary of abundances of isotopes, in the format:
        {element_name[isotope_number]: relative abundance, ..}.

    Returns
    -----
    float
        Value of M0.

    Notes
    ---
    X represents C with default isotopic abundance.

-------------------------------------------------------
Help on function compute_M1_nl in module seq_to_first_iso.seq_to_first_iso:

compute_M1_nl(f, a)
    Compute abundance of second isotopologue M1 from its formula.
```

(continues on next page)

(continued from previous page)

```
    Parameters
    -------
    f : pyteomics.mass.Composition
        Chemical formula, as a dict of counts for each element:
        {element_name: count_of_element_in_sequence, ...}.
    a : dict
        Dictionary of abundances of isotopes, in the format:
        {element_name[isotope_number]: relative abundance, ..}.

    Returns
    -----
    float
        Value of M1.

    Notes
    ---
    X represents C with default isotopic abundance.
```

[14]:
```
# Compute M0 with natural carbon.
first_isotopologue = stfi.compute_M0_nl(chem_formula, isotopic_abundance)
print(f"M0 in Normal Condition: {first_isotopologue}")

first_isotopologue = stfi.compute_M0_nl(chem_formula, C12_abundance)
print(f"M0 in 12C condition: {first_isotopologue}")
```

```
M0 in Normal Condition: 0.5493191520383802
M0 in 12C condition: 0.7403283857401063
```

[15]:
```
# Compute M1 with natural carbon.
first_isotopologue = stfi.compute_M1_nl(chem_formula, isotopic_abundance)
print(f"M1 in Normal Condition: {first_isotopologue}")

first_isotopologue = stfi.compute_M1_nl(chem_formula, C12_abundance)
print(f"M1 in 12C condition: {first_isotopologue}")
```

```
M1 in Normal Condition: 0.313702912736476
M1 in 12C condition: 0.200655465179031
```

2.1.5 Get the composition of a list of modifications

[16]:
```
help(stfi.get_mods_composition)
```

```
Help on function get_mods_composition in module seq_to_first_iso.seq_to_first_iso:

get_mods_composition(modifications)
    Return the composition of a list of modifications.

    Parameters
    -------
    modifications: list of str
        List of modifications string (corresponding to Unimod titles).

    Returns
    -----
```

(continues on next page)

(continued from previous page)

```
    pyteomics.mass.Composition
        The total composition change.
```

[17]:
```
# Modifications must be strict Unimod entries title.
modification_list = ["Acetyl", "Phospho", "phospho"]  # phospho does not correspond␣
↪to a title, it will be ignored
total_composition = stfi.get_mods_composition(modification_list)
print(f"Total composition shift for {modification_list} is {total_composition}")
```

```
[2019-07-10, 10:38:33] WARNING : Unimod entry not found for : phospho
```

```
Total composition shift for ['Acetyl', 'Phospho', 'phospho'] is Composition({'H': 3,␣
↪'C': 2, 'O': 4, 'P': 1})
```

2.1.6 Get human-readable chemical formula

[18]:
```
help(stfi.formula_to_str)
```

```
Help on function formula_to_str in module seq_to_first_iso.seq_to_first_iso:

formula_to_str(composition)
    Return formula from Composition as a string.

    Parameters
    -------
    composition : pyteomics.mass.Composition
        Chemical formula.

    Returns
    -----
    str
        Human-readable string of the formula.

    Warnings
    -----
    If the composition has elements not in USED_ELEMS, they will not
    be added to the output.
```

[19]:
```
# This is the function used to get the formulas in the output.
formula_str = stfi.formula_to_str(total_composition)
print(f"{total_composition} becomes {formula_str}")
```

```
Composition({'H': 3, 'C': 2, 'O': 4, 'P': 1}) becomes C2H3O4P1
```

[20]:
```
# !!! Warning: if the Composition has elements not in "CHONPSX", they will not be in␣
↪the final string.
bad_composition = mass.Composition("U")
formula_str = stfi.formula_to_str(bad_composition)
print(f"{bad_composition} becomes {formula_str}")
```

```
Composition({'H': 7, 'C': 3, 'O': 2, 'N': 1, 'Se': 1}) becomes C3H7O2N1
```

Here, "non-CHONPSX" element **Se (Selenium) is ignored**.

2.1.7 Parse a file with peptide sequences

seq-to-first-iso accepts files with 1 sequence per line.
Optionally, annotations/sequence IDs can be placed in the same line before sequences if separated by a separator (default: “ ⸮↼▷*Theprogramdeclaresthatthefilehasannotationsifthesepratoris foundonthefirstline*▷ *Theparserwillignorelineswheresequenceshaveincorrectcharacters*↽⸮*notin* “ACDEFGHIKLMNPQRSTVWY”) unless it corresponds to XTandem’s Post-Translational Modification notation.

[21]:
```
help(stfi.sequence_parser)
```

```
Help on function sequence_parser in module seq_to_first_iso.seq_to_first_iso:

sequence_parser(file, sep='\t')
    Return information on sequences parsed from a file.

    Parameters
    -------
    file : str
        Filename, the file can either just have sequences for each line or
        can have have annotations and sequences with a separator in-between.
    sep : str, optional
        Separator for files with annotations (default is ``\t``).

    Returns
    -----
    dict
        | Parsed output with "key: values" :
        |     - "annotations": a list of annotations if any.
        |     - "raw_sequences": a list of unmodified peptide sequences.
        |     - "sequences": a list of uppercase peptide sequences.
        |     - "modifications": a list of lists of PTMs.
        |     - "ignored_lines": the number of ignored lines.

    Warnings
    -----
    The function uses the first line to evaluate if the file has
    annotations or not, hence a file should have a consistent format.

    Notes
    ---
    | Supports Xtandem's Post-Translational Modification notation (0.4.0).
    | Supports annotations (0.3.0).
```

[22]:
```
parsed_output = stfi.sequence_parser("peptides.txt")
pprint(parsed_output)
```

```
{'annotations': [],
 'ignored_lines': 0,
 'modifications': [[], [], []],
 'raw_sequences': ['YAQEISR', 'VGFPVLSVKEHK', 'LAMVIIKEFVDDLK'],
 'sequences': ['YAQEISR', 'VGFPVLSVKEHK', 'LAMVIIKEFVDDLK']}
```

For *peptides.txt*, the list of annotations is empty (there are no annotations), no lines are ignored and no modifications were found so raw_sequences (with modifications) are the same as sequences (without modifications)

[23]:
```
# Get the values in the returned dict.
annotations = parsed_output["annotations"]
ignored_lines = parsed_output["ignored_lines"]
modifications = parsed_output["modifications"]
sequences = parsed_output["sequences"]
raw_sequences = parsed_output["raw_sequences"]
```

[24]:
```
# Parsing a file with annotations and modifications following XTandem notation.
parsed_output = stfi.sequence_parser("peptides_mod.tsv")
pprint(parsed_output)
```

```
{'annotations': ['0', '1', '2', '4', '7', '11', '13', '14', '16', '24', '27'],
 'ignored_lines': 1,
 'modifications': [['Oxidation'],
                   ['Phospho'],
                   [],
                   ['Glutathione'],
                   ['Acetyl', 'Oxidation'],
                   ['Heme'],
                   ['Pro->Val'],
                   [],
                   ['Glutathione'],
                   [],
                   ['Acetyl', 'Oxidation', 'Acetyl', 'Acetyl']],
 'raw_sequences': ['VPK(Oxidation)ER',
                   'VLLIDLRIPQR(Phospho)SAINHIVAPNLVNVDPNLLWDK',
                   'QRTTFFVLGINTVNYPDIYEHILER',
                   'AELFL(Glutathione)LNR',
                   '.(Acetyl)VGEVFINYIQRQNELFQGKLAYLII(Oxidation)DTCLSIVRPNDSKPLDNR',
                   'YKTMNTFDPD(Heme)EKFEWFQVWQAVK',
                   'HKSASSPAV(Pro->Val)NADTDIQDSSTPSTSPSGRR',
                   'FHNK',
                   '.(Glutathione)MDLEIK',
                   'LANEKPEDVFER',
                   '.
↪(Acetyl)SDTPLR(Oxidation)D(Acetyl)EDG(Acetyl)LDFWETLRSLATTNPNPPVEK'],
 'sequences': ['VPKER',
               'VLLIDLRIPQRSAINHIVAPNLVNVDPNLLWDK',
               'QRTTFFVLGINTVNYPDIYEHILER',
               'AELFLLNR',
               'VGEVFINYIQRQNELFQGKLAYLIIDTCLSIVRPNDSKPLDNR',
               'YKTMNTFDPDEKFEWFQVWQAVK',
               'HKSASSPAVNADTDIQDSSTPSTSPSGRR',
               'FHNK',
               'MDLEIK',
               'LANEKPEDVFER',
               'SDTPLRDEDGLDFWETLRSLATTNPNPPVEK']}
```

2.1.8 Create a dataframe from a list of sequences

[25]:
```
help(stfi.seq_to_df)
```

```
Help on function seq_to_df in module seq_to_first_iso.seq_to_first_iso:

seq_to_df(sequences, unlabelled_aa, **kwargs)
    Create a dataframe from sequences and return its name.
```

(continues on next page)

(continued from previous page)

```
    Parameters
    -------
    sequences : list of str
        List of pure peptide sequences string.
    unlabelled_aa : container object
        Container of unlabelled amino acids.
    annotations : list of str, optional
        List of IDs for the sequences.
    raw_sequences : list of str, optional
        List of sequences with Xtandem PTMs.
    modifications : list of str, optional
        List of modifications for raw_sequences.

    Returns
    -----
    pandas.Dataframe
        | Dataframe with :
        |                  annotation (optional), sequence, mass,
                           formula, formula_X, M0_NC, M1_NC, M0_12C, M1_12C.

    Warnings
    -----
    If raw_sequence is provided, modifications must also be provided
    and vice-versa.
```

[26]:
```
# Dataframe from a list of sequences, give an empty list to unlabelled_aa have all␣
↪amino acids labelled.
df_peptides = stfi.seq_to_df(sequences, unlabelled_aa=[])
df_peptides.head()
```

```
[2019-07-10, 10:38:33] INFO    : Computing formula
[2019-07-10, 10:38:33] INFO    : Computing mass
[2019-07-10, 10:38:33] INFO    : Computing M0 and M1
```

[26]:
```
         sequence         mass          formula        formula_X     M0_NC  \
0         YAQEISR   865.429381     C37H59O13N11     C37H59O13N11  0.620641
1    VGFPVLSVKEHK  1338.765971    C63H102O16N16    C63H102O16N16  0.455036
2  LAMVIIKEFVDDLK  1632.916066  C76H128O21N16S1  C76H128O21N16S1  0.369940

      M1_NC    M0_12C    M1_12C
0  0.280871  0.920656  0.051619
1  0.345060  0.890522  0.074113
2  0.337319  0.831576  0.081017
```

[27]:
```
# The dict returned by sequence_parser can be unpacked in the function (remove
↪"ignored_lines" against warning).
parsed_output.pop("ignored_lines")
df_peptides = stfi.seq_to_df(unlabelled_aa=unlabelled_amino_acids, **parsed_output)
# Dataframe with annotations.
df_peptides.head()
```

```
[2019-07-10, 10:38:34] INFO    : Computing formula
[2019-07-10, 10:38:34] INFO    : Computing composition of modifications
[2019-07-10, 10:38:34] WARNING : Fe in (Heme) is not supported in the computation of␣
↪M0 and M1
```

(continues on next page)

(continued from previous page)

```
[2019-07-10, 10:38:34] INFO    : Computing mass
[2019-07-10, 10:38:34] INFO    : Computing M0 and M1
```

[27]:
```
  annotation                                          sequence         mass  \
0          0                                 VPK(Oxidation)ER   643.365324
1          1         VLLIDLRIPQR(Phospho)SAINHIVAPNLVNVDPNLLWDK  3838.102264
2          2                           QRTTFFVLGINTVNYPDIYEHILER  3037.566156
3          4                               AELFL(Glutathione)LNR  1279.623072
4          7  .(Acetyl)VGEVFINYIQRQNELFQGKLAYLII(Oxidation)D...  5049.638616

             formula            formula_X     M0_NC     M1_NC    M0_12C  \
0        C27H49O9N9         C21H49O9N9X6  0.703864  0.235324  0.880417
1  C172H285O49N48P1  C154H285O49N48P1X18  0.113124  0.236320  0.583917
2    C140H212O40N36    C128H212O40N36X12  0.171960  0.290060  0.672794
3    C55H89O18N15S1     C46H89O18N15S1X9  0.470936  0.318055  0.768911
4  C226H361O68N61S1  C205H361O68N61S1X21  0.054198  0.148778  0.481767

     M1_12C
0  0.096230
1  0.256235
2  0.212051
3  0.140284
4  0.264187
```

If modifications have "non-CHONPSX" elements, computation of isotopologue intensities will be less accurate.

2.2 Command-line interface of seq-to-first-iso

seq-to-first-iso computes the first two isotopologue intentities (M0 and M1) from peptide sequences with natural carbon and with 99.99 % 12C enriched carbon.

The program can take into account unlabelled amino acids to simulate auxotrophies to amino acids.

seq-to-first-iso is available as a command line tool.

[1]:
```
import pandas as pd  # For output visualisation.
```

Note: the exclamation mark ''!'' is a magic command to run a Linux command within a Jupyter notebook. In a real Linux terminal, you don't need it.

[2]:
```
!seq-to-first-iso -v
```

```
seq-to-first-iso 0.4.3
```

[3]:
```
!seq-to-first-iso -h
```

```
usage: seq-to-first-iso [-h] [-o OUTPUT] [-n amino_a] [-v] input

Read a file of sequences and creates a tsv file

positional arguments:
  input                 file to parse

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

(continues on next page)

(continued from previous page)

```
                        name of output file
  -n amino_a, -non-labelled-aa amino_a
                        amino acids with default abundance
  -v, -version          show program's version number and exit
```

The output file will have columns

sequence	mass	formula	formula_X	M0_NC	M1_NC	M0_12C	M1_12C
original sequence	sequence mass	chemical formula	chemical formula with X	M0 in NC	M1 in NC	M0 in 12C	M1 in 12C

X: Virtual element created to represent unlabelled carbon
NC: Normal Condition (Natural Carbon)
12C: 12C condition (12C enriched carbon)

[4]:
```
# File used.
!cat peptides.txt
```

```
YAQEISR
VGFPVLSVKEHK
LAMVIIKEFVDDLK
```

2.2.1 Minimal command

[5]:
```
!seq-to-first-iso peptides.txt
```

```
[2019-06-28, 14:49:36] INFO    : Parsing file
[2019-06-28, 14:49:36] INFO    : Computing formula
[2019-06-28, 14:49:36] INFO    : Computing composition of modifications
[2019-06-28, 14:49:36] INFO    : Computing mass
[2019-06-28, 14:49:36] INFO    : Computing M0 and M1
```

Running the command above will create a file with tab-separated values : *peptides_stfi.tsv*

[6]:
```
# Read basic output file.
df = pd.read_csv("peptides_stfi.tsv", sep="\t")
df.head()
```

[6]:
```
          sequence         mass          formula        formula_X     M0_NC  \
0         YAQEISR   865.429381     C37H59O13N11     C37H59O13N11  0.620641
1    VGFPVLSVKEHK  1338.765971    C63H102O16N16    C63H102O16N16  0.455036
2  LAMVIIKEFVDDLK  1632.916066  C76H128O21N16S1  C76H128O21N16S1  0.369940

      M1_NC    M0_12C    M1_12C
0  0.280871  0.920656  0.051619
1  0.345060  0.890522  0.074113
2  0.337319  0.831576  0.081017
```

2.2.2 Changing output name

You can also change the name of the output file

[7]:
```
!seq-to-first-iso peptides.txt -o sequence
```

```
[2019-06-28, 14:49:38] INFO    : Parsing file
[2019-06-28, 14:49:38] INFO    : Computing formula
[2019-06-28, 14:49:38] INFO    : Computing composition of modifications
[2019-06-28, 14:49:38] INFO    : Computing mass
[2019-06-28, 14:49:38] INFO    : Computing M0 and M1
```

[8]:
```
# Read output file with different name.
df = pd.read_csv("sequence.tsv", sep="\t")
df.head()
```

[8]:

	sequence	mass	formula	formula_X	M0_NC	M1_NC	M0_12C	M1_12C
0	YAQEISR	865.429381	C37H59O13N11	C37H59O13N11	0.620641	0.280871	0.920656	0.051619
1	VGFPVLSVKEHK	1338.765971	C63H102O16N16	C63H102O16N16	0.455036	0.345060	0.890522	0.074113
2	LAMVIIKEFVDDLK	1632.916066	C76H128O21N16S1	C76H128O21N16S1	0.369940	0.337319	0.831576	0.081017

2.2.3 Choosing unlabelled amino acids

[9]:
```
!seq-to-first-iso peptides.txt -n V,W -o sequence
```

```
[2019-06-28, 14:49:41] INFO    : Amino acid with default abundance: ['V', 'W']
[2019-06-28, 14:49:41] INFO    : Parsing file
[2019-06-28, 14:49:41] INFO    : Computing formula
[2019-06-28, 14:49:41] INFO    : Computing composition of modifications
[2019-06-28, 14:49:41] INFO    : Computing mass
[2019-06-28, 14:49:41] INFO    : Computing M0 and M1
```

[10]:
```
# Read output file with different name and unlabelled amino acids.
df = pd.read_csv("sequence.tsv", sep="\t")
df.head()
```

[10]:

	sequence	mass	formula	formula_X	M0_NC	M1_NC	M0_12C	M1_12C
0	YAQEISR	865.429381	C37H59O13N11	C37H59O13N11	0.620641	0.280871	0.920656	0.051619
1	VGFPVLSVKEHK	1338.765971	C63H102O16N16	C48H102O16N16X15	0.455036	0.345060	0.758956	0.185155
2	LAMVIIKEFVDDLK	1632.916066	C76H128O21N16S1	C66H128O21N16S1X10	0.369940	0.337319	0.747509	0.152927

The carbon of unlabelled amino acids is shown as X in column "formula_X".

We can observe that for sequence "YAQEISR" that has no unlabelled amino acids, M0 and M1 are the same as the previous *sequence.tsv*, regardless of the condition.

In contrast sequence "VGFPVLSVKEHK", in 12C condition, has M0 go down from 0.8905224988642593 to 0.7589558393662944 and M1 go up from 0.07411308335404865 to 0.18515489894512063.

CHAPTER

THREE

REFERENCE MANUAL

3.1 seq-to-first-iso

Compute first two isotopologue intensities from sequences.

The program computes M0 and M1 and differentiate labelled (with a 99.99 % C[12] enrichment) and unlabelled amino acids.

Read a file composed of amino acid sequences on each line and return : sequence, mass, formula, formula_X, M0_NC, M1_NC, M0_12C and M1_12C in a tsv file.

Formula_X is the chemical formula with carbon of unlabelled amino acids marked as X.

NC means Normal Condition, 12C means C[12] enrichment condition.

Example

Running the script after installation

```
$ seq-to-first-iso sequences.txt
```

will provide file 'sequences_stfi.tsv'

Notes

Carbon of unlabelled amino acids keep default isotopic abundance, and are represented as X in formulas. Naming conventions for isotopes follow pyteomics's conventions.

`seq_to_first_iso.`**`sequence_parser`**(*file*, *sep='\t'*)

Return information on sequences parsed from a file.

Parameters

- **`file`** (*`str`*) – Filename, the file can either just have sequences for each line or can have have annotations and sequences with a separator in-between.
- **`sep`** (*`str, optional`*) – Separator for files with annotations (default is `\t`).

Returns

Parsed output with "key: values" :
- "annotations": a list of annotations if any.
- "raw_sequences": a list of unmodified peptide sequences.
- "sequences": a list of uppercase peptide sequences.

- "modifications": a list of lists of PTMs.
- "ignored_lines": the number of ignored lines.

Return type dict

Warning: The function uses the first line to evaluate if the file has annotations or not, hence a file should have a consistent format.

Notes

Supports Xtandem's Post-Translational Modification notation (0.4.0).
Supports annotations (0.3.0).

seq_to_first_iso.**separate_labelled**(*sequence*, *unlabelled_aa*)
Get the sequence of unlabelled amino acids from a sequence.

Parameters

- **sequence** (*str*) – String of amino acids.
- **unlabelled_aa** (*container object*) – Container (list, string...) of unlabelled amino acids.

Returns

The sequences as a tuple of string with:
- the sequence without the unlabelled amino acids
- the unlabelled amino acids in the sequence

Return type tuple(str, str)

seq_to_first_iso.**compute_M0_nl**(*f*, *a*)
Return the monoisotopic abundance M0 of a formula with mixed labels.

Parameters

- **f** (*pyteomics.mass.Composition*) – Chemical formula, as a dict of counts for each element: {element_name: count_of_element_in_sequence, ...}.
- **a** (*dict*) – Dictionary of abundances of isotopes, in the format: {element_name[isotope_number]: relative abundance, ..}.

Returns Value of M0.

Return type float

Notes

X represents C with default isotopic abundance.

seq_to_first_iso.**compute_M1_nl**(*f*, *a*)
Compute abundance of second isotopologue M1 from its formula.

Parameters

- **f** (*pyteomics.mass.Composition*) – Chemical formula, as a dict of counts for each element: {element_name: count_of_element_in_sequence, ... }.
- **a** (*dict*) – Dictionary of abundances of isotopes, in the format: {element_name[isotope_number]: relative abundance, ..}.

Returns Value of M1.

Return type float

Notes

X represents C with default isotopic abundance.

seq_to_first_iso.**formula_to_str**(*composition*)
Return formula from Composition as a string.

Parameters **composition** (*pyteomics.mass.Composition*) – Chemical formula.

Returns Human-readable string of the formula.

Return type str

> **Warning:** If the composition has elements not in USED_ELEMS, they will not be added to the output.

seq_to_first_iso.**seq_to_xcomp**(*sequence_l*, *sequence_nl*)
Take 2 amino acid sequences and return the composition with X.

The second sequence will have its C replaced by X.

Parameters

- **sequence_l** (*str or pyteomics.mass.Composition*) – Sequence or composition with labelled amino acids.
- **sequence_nl** (*str or pyteomics.mass.Composition*) – Sequence or composition where amino acids are not labelled.

Returns Composition with unlabelled carbon as element X.

Return type pyteomics.mass.Composition

Notes

The function assumes the second sequence has no terminii (H-, -OH).

Supports pyteomics.mass.Composition as argument (0.5.1).

If mass.Composition objects are provided, the function assumes the terminii of the second composition were already removed.

seq_to_first_iso.**get_mods_composition**(*modifications*)
Return the composition of a list of modifications.

Parameters **modifications** (*list of str*) – List of modifications string (corresponding to Unimod titles).

Returns The total composition change.

Return type pyteomics.mass.Composition

seq_to_first_iso.**seq_to_df**(*sequences*, *unlabelled_aa*, ***kwargs*)

Create a dataframe from sequences and return its name.

Parameters

- **sequences** (*list of str*) – List of pure peptide sequences string.
- **unlabelled_aa** (*container object*) – Container of unlabelled amino acids.
- **annotations** (*list of str, optional*) – List of IDs for the sequences.
- **raw_sequences** (*list of str, optional*) – List of sequences with Xtandem PTMs.
- **modifications** (*list of str, optional*) – List of modifications for raw_sequences.

Returns

Dataframe with :

annotation (optional), sequence, mass, formula, formula_X, M0_NC, M1_NC, M0_12C, M1_12C.

Return type pandas.Dataframe

Warning: If raw_sequence is provided, modifications must also be provided and vice-versa.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

S

INDEX