TOF-simulator

Release 3.00

Jan 17, 2023

Contents

1 Contents **3**
- 1.1 Installation . . . 3
- 1.2 Graphical interface . . . 3
- 1.3 Examples . . . 6
- 1.4 Implementation of time-of-flight calculations . . . 13
- 1.5 Package API . . . 14

2 Indices y tablas **23**

Python Module Index **25**

Index **27**

Esta documentación en castellano

CHAPTER 1

Contents

1.1 Installation

Install using pip, either system-wide (administrator rights are needed)

```
pip install tof-simulator-X.Y.tar.gz
```

or (usually preferred) for the current user:

```
pip install --user tof-simulator-X.Y.tar.gz
```

1.1.1 Dependencies

- In order to run the simulator programmatically:
  - Only Numpy is required.
  - Optionally:
    - Package tabulate for pretty-printing information on TOF parameters and masses.
    - matplotlib is needed to use included capabilities for plotting.
- To use the GUI, the additional requirements are:
  - matplotlib
  - PyGobject.

    Information on how to install PyGobject on different platforms may be found in the documentation

1.2 Graphical interface

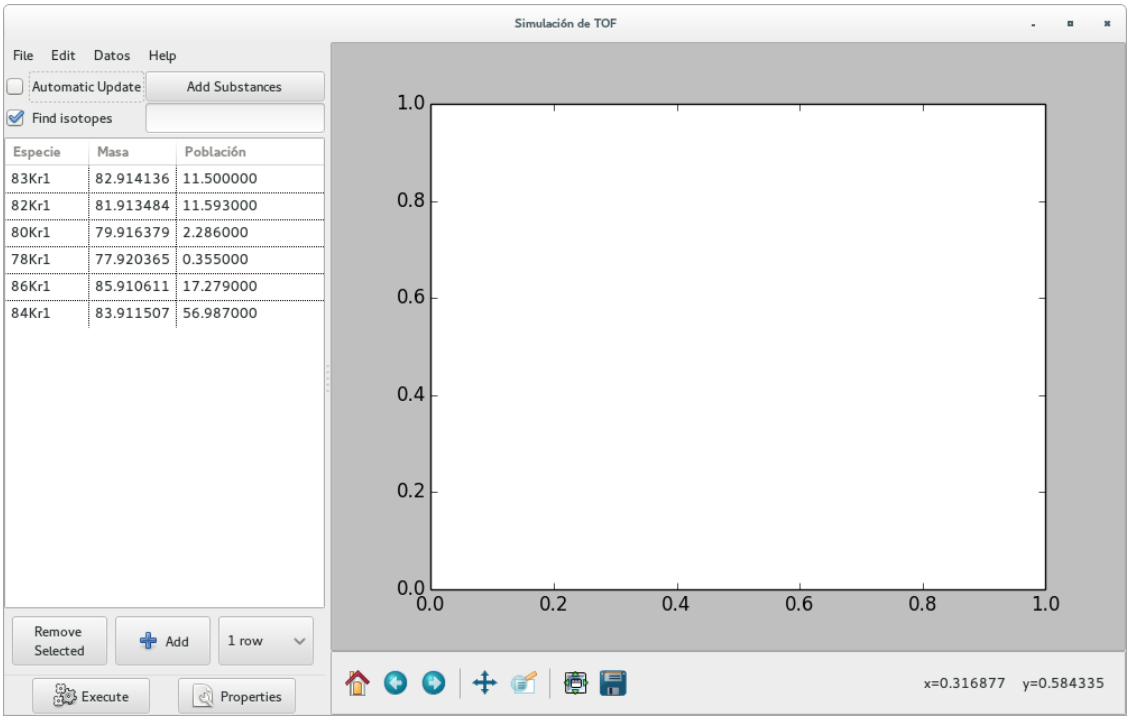
Start the program from a command line with either:

```
$> tof_gtk
```

or:

```
$> tof_gtk.py
```

The graphical interface is intended to be intuitively obvious. At startup you will find a window similar to the following:



Where, in this case, all isotopes of Kripton are defined. The button "Execute" produces the time-of-flight signal for the included masses.

1.2.1 Mass selection

- Each mass may be selected with the *mouse*
- Once selected, they may be removed by using the button *Remove Selected* in the bottom left corner
- The simplest form to add new substances is:
  - Add substances or elements in the box below the *Add Substances* button
  - The notation for (one or more) substances is:
    - Kr, Al, Au (for elements)
    - N2, O2, H2O, CO2, C7H8 (molecules, in this case N_2, O_2, CO_2, C_7H_8)
  - Then, press the *Add Substances* button (or the shortcut key *Alt s*)
  - The box *Find isotopes* (enabled by default) allows to include all isotopes for all atoms, and all combinations of isotopes for molecule. Only isotopic composition with non-negligible abundance will be included (value configurable by *Isotope threshold*)
  - In this way, all isotopes of a given substance will be included with the correct relative abundances.

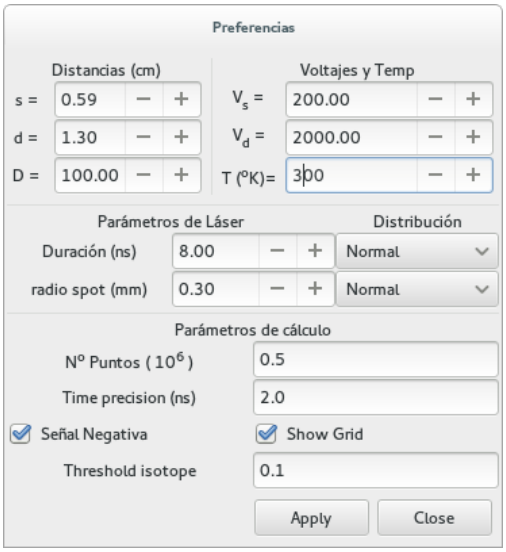
  - Convention for *label* is: a list of isotopes, separated by a dash. Each isotope has the form: Mass number + Symbol + Number of atoms. For instance,
    - For common water isotope (H_2O) we will have: **1H2-16O1**.
    - For heavy water, with oxygen-18 (^{18}O) we will have: **1H2-18O1**
- Also, substances may be added manually in the table, by:
  - Add rows (clicking the *Add* button)
  - Fill in the three boxes in the row
    - Especie: Label of identification
    - Masa: The mass of the substance in AMU. It will determine the position and width of the peak
    - Población: Abundance of the substance. Determines the heigth of the simulated peak.

1.2.2 Parameters configuration

- The box *Automatic Update* allows to see modifications on real time when parameters are changed.
- The button *Properties* (*Alt p*) opens a window with options for construction and operation parameters

1.2.3 Configuration window

The properties window is shown in the following figure



Here, all construction and operation parameters of the TOF may be configured:

TOF dimensions

- Extraction distance
- Lenght of the acceleration zone d
- Free flight distance D

Applied voltages

- Extraction voltage V_s
- Acceleration voltage V_d
- Working temperature T

Laser

- Pulse duration and temporal dependence (normal or uniform)
- Ionization spot radius and spatial profile (normal or uniform)

Plotting

- Negative signal (as shown in osciloscopes)
- Add a grid

Isotope threshold

The box *Threshold isotope* allows to set the minimal abundance (percentage) such that the substance is included

1.3 Examples

Some simple examples and applications:

1.3.1 Example of use of tofsim

Start the session

Import the package, or simply the object `ToF` For instance, either:

```
import tofsim
T = tofsim.ToF()
```

or

```
from tofsim import ToF
T = ToF()
```

The object `ToF` keeps the information on all the construction and operation parameters

```
print ('Parámetros del TOF')
print (T.tof_parameters)
print ('Valor original: Vs =', T.Vs)
T.Vs = T.Vs /2.
print ('Valor cambiado: Vs =', T.Vs)
for k in T.tof_parameters:
  print ('{0} = {1}'.format(k, T.__dict__[k]))
```

```
Parámetros del TOF
['s', 'd', 'D', 'Vs', 'Vd', 'ds', 'dt', 't0', 'r0']
Valor original: Vs = 200.0
Valor cambiado: Vs = 100.0
s = 0.59
d = 1.3
D = 100.0
Vs = 100.0
Vd = 2000.0
ds = 0.03
dt = 0.008
t0 = 0.0
r0 = 1000.0
```

Calculations

We add some substances, for simulation of time-of-fligth signals

```
T.add_substances('Li,H2O')
```

```
# Veamos que fragmentos hemos agregado:
print(T.fragments.to_text())
```

```
| Sustancia   | Fórmula       |     Masa |   Abundancia |
| Li          | 6Li^{0+}      |  6.01512 |      7.59    |
| Li          | 7Li^{0+}      |  7.016   |     92.41    |
| H2O         | 1H2-16O^{0+}  | 18.0106  |     99.7341  |
| H2O         | 1H2-18O^{0+}  | 20.0148  |      0.204953 |
```

The method `ToF.signal()` returns the simulated signal of the TOF.

```
s = T.signal()
print ('type(s):', type(s),'\n')
print ('keys:', s.keys())
```

```
type(s): <class 'dict'>

keys: dict_keys(['signal', '6Li^{0+}', '7Li^{0+}', '1H2-16O^{0+}', '1H2-18O^{0+}',
↪'time'])
```

The result is a dictionary, where each element is a data array:

- s['time'] the x-axis, contains the time-window
- s['signal'] has the resultant signal, summing the individual signal for all the species.
- All other elements, of the form `s[substance]` have the values of the signal for each substance.

```
# Cada elemento es un numpy array
print (type(s['signal']))
```

```
<class 'numpy.ndarray'>
```

Note that it is not necessary to keep the value of `s` in the above example, it is kept in the `ToF` object as `T.times`

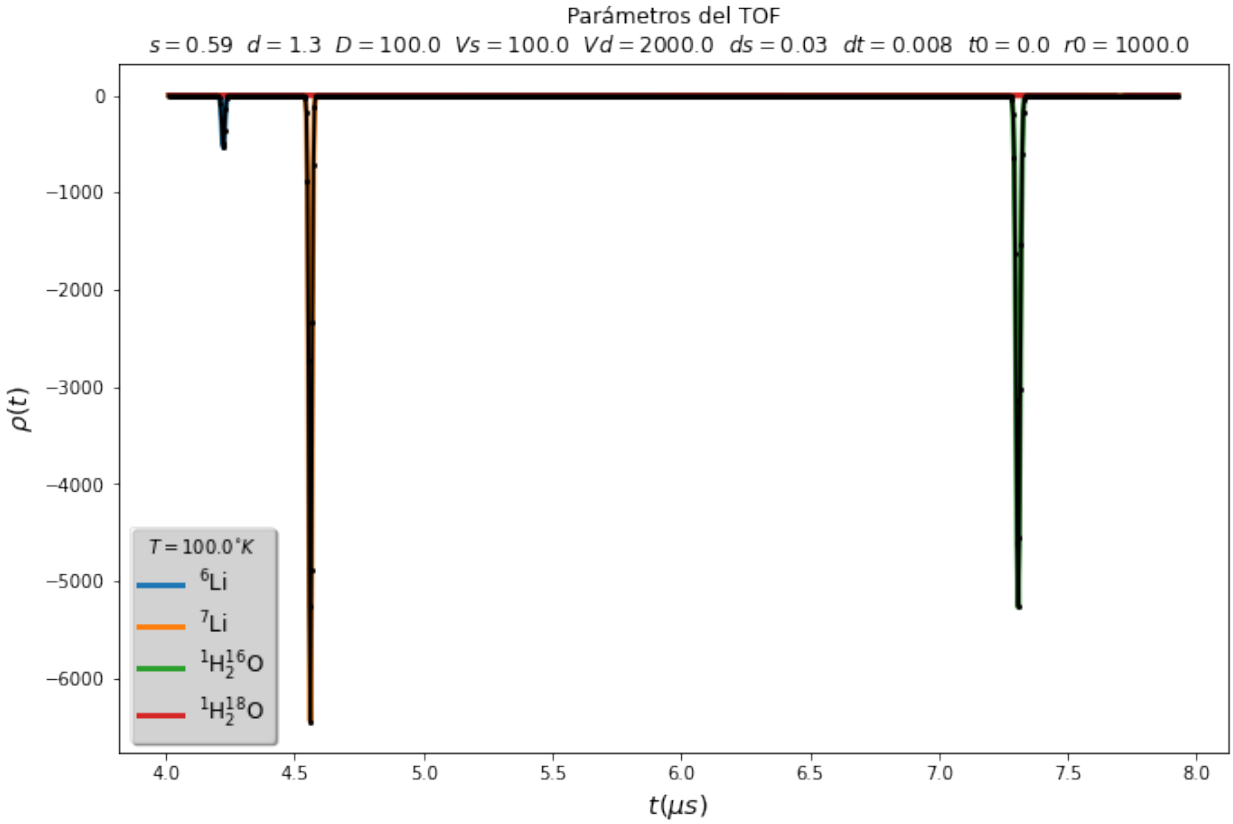
```
T.times == s
```

```
True
```

Plotting

The simplest way of plotting the signals is using the method `make_plot`.

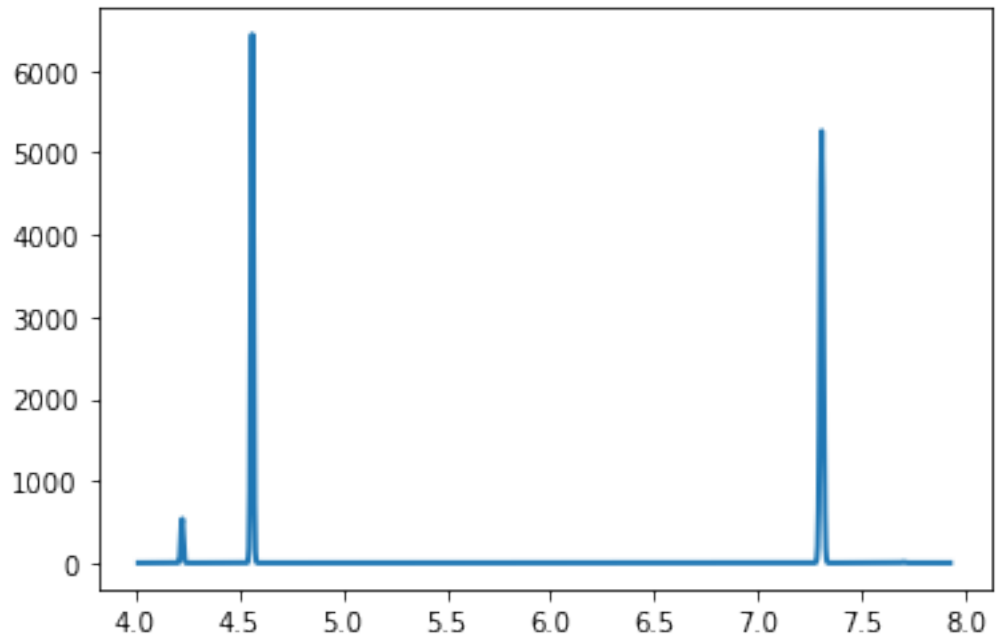
```
f = T.make_plot(T.fragments, negative=True, graph_all=True, hide_legend=False)
```



This method allows for some customization. Also, since the data is kept in numpy arrays it may be plotted independently:

```
import matplotlib.pyplot as plt
x, y = T.times['time'], T.times['signal']
plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x7fe9faa94ee0>]
```



1.3.2 Getting Information on the peaks

We want to get the peaks from the spectra. We start importing and creating the TOF object with all isotopes from Ar and Kr

```
from tofsim import ToF
```

```
T = ToF('Ar,Kr')
T.Vs = 120                          # Extraction voltaje
T.Vd = 3000                         # Acceleration voltaje
T.signal();                         # Make the spectra
```

```
p = T.get_statistics_peaks()     # Get the peaks
```

Here `p` is a dictionary-like object, where each key is a substance and each value is a dictionary containing the information on the corresponding peak

```
p.keys()
```

```
dict_keys(['36Ar^{+}', '38Ar^{+}', '40Ar^{+}', '78Kr^{+}', '80Kr^{+}', '82Kr^{+}',
↪'83Kr^{+}', '84Kr^{+}', '86Kr^{+}'])
```

```
list(p.values())[0]
```

```
{'index': (53, 56, 60),
 'position': 8.519009871520703,
 'height': 6.774956591427326,
 'width': 0.03684000090535555}
```

```
p.headers
```

```
['index', 'position', 'height', 'width']
```

Output the data of the peaks

```
print(p)
```

```
Substance    index              position      height      width
-----------  ---------------  ----------  ----------  ---------
36Ar^{+}     (53, 56, 60)        8.51901     6.77496  0.03684
38Ar^{+}     (97, 101, 104)      8.75419     1.28081  0.03684
40Ar^{+}     (140, 144, 147)     8.98108  2010.2      0.03684
78Kr^{+}     (816, 820, 825)    12.5402      6.76633  0.0473657
80Kr^{+}     (846, 851, 855)    12.7016     43.8943   0.0473657
82Kr^{+}     (876, 881, 885)    12.8594    222.046    0.0473657
83Kr^{+}     (891, 896, 900)    12.938     220.033    0.0473657
84Kr^{+}     (906, 910, 915)    13.0137   1079.69     0.0473657
86Kr^{+}     (935, 940, 944)    13.1696    329.294    0.0473657
```

If we want to print them in a different way we may use directly the dictionary-like object or the list obtained by using `tolist()` method

```
pl = p.tolist()
pl[0]
```

```
['36Ar^{+}',
 (53, 56, 60),
 8.519009871520703,
 6.774956591427326,
 0.03684000090535555]
```

Importing and using the tabulate package we may output it to any supported format. For instance, “fancy_grid”:

```
from tabulate import tabulate
headers = ['Fragment'] + p.headers
```

```
print(tabulate(p.tolist(), headers=headers, tablefmt='fancy_grid'))
```

```
╒═══════════╤═══════════════╤════════════╤═══════════╤═══════════╕
│ Fragment  │ index         │   position │    height │     width │
╞═══════════╪═══════════════╪════════════╪═══════════╪═══════════╡
│ 36Ar^{+}  │ (53, 56, 60)  │    8.51901 │   6.77496 │ 0.03684   │
├───────────┼───────────────┼────────────┼───────────┼───────────┤
│ 38Ar^{+}  │ (97, 101, 104)│    8.75419 │   1.28081 │ 0.03684   │
├───────────┼───────────────┼────────────┼───────────┼───────────┤
│ 40Ar^{+}  │ (140, 144, 147)│   8.98108 │ 2010.2    │ 0.03684   │
├───────────┼───────────────┼────────────┼───────────┼───────────┤
│ 78Kr^{+}  │ (816, 820, 825)│  12.5402  │   6.76633 │ 0.0473657 │
├───────────┼───────────────┼────────────┼───────────┼───────────┤
│ 80Kr^{+}  │ (846, 851, 855)│  12.7016  │  43.8943  │ 0.0473657 │
├───────────┼───────────────┼────────────┼───────────┼───────────┤
│ 82Kr^{+}  │ (876, 881, 885)│  12.8594  │ 222.046   │ 0.0473657 │
├───────────┼───────────────┼────────────┼───────────┼───────────┤
│ 83Kr^{+}  │ (891, 896, 900)│  12.938   │ 220.033   │ 0.0473657 │
├───────────┼───────────────┼────────────┼───────────┼───────────┤
```

(continues on next page)

(continued from previous page)

```
| 84Kr^{+}   | (906, 910, 915) |   13.0137   | 1079.69     | 0.0473657 |
|------------|-----------------|-------------|-------------|-----------|
| 86Kr^{+}   | (935, 940, 944) |   13.1696   |  329.294    | 0.0473657 |
```

or “latex”:

```
print(tabulate(p.tolist(), headers=headers, tablefmt='latex'))
```

```
begin{tabular}{llrrr}
hline
 Fragment    & index           &    position &     height &     width \
hline
 36Ar^{}{+}    & (53, 56, 60)     &     8.51901 &     6.77496 & 0.03684    \
 38Ar^{}{+}    & (97, 101, 104)   &     8.75419 &     1.28081 & 0.03684    \
 40Ar^{}{+}    & (140, 144, 147) &     8.98108 & 2010.2      & 0.03684    \
 78Kr^{}{+}    & (816, 820, 825) &    12.5402  &     6.76633 & 0.0473657 \
 80Kr^{}{+}    & (846, 851, 855) &    12.7016  &    43.8943  & 0.0473657 \
 82Kr^{}{+}    & (876, 881, 885) &    12.8594  &   222.046   & 0.0473657 \
 83Kr^{}{+}    & (891, 896, 900) &    12.938   &   220.033   & 0.0473657 \
 84Kr^{}{+}    & (906, 910, 915) &    13.0137  & 1079.69     & 0.0473657 \
 86Kr^{}{+}    & (935, 940, 944) &    13.1696  &   329.294   & 0.0473657 \
hline
end{tabular}
```

Plot the data

```
import numpy as np
import matplotlib.pyplot as plt
```

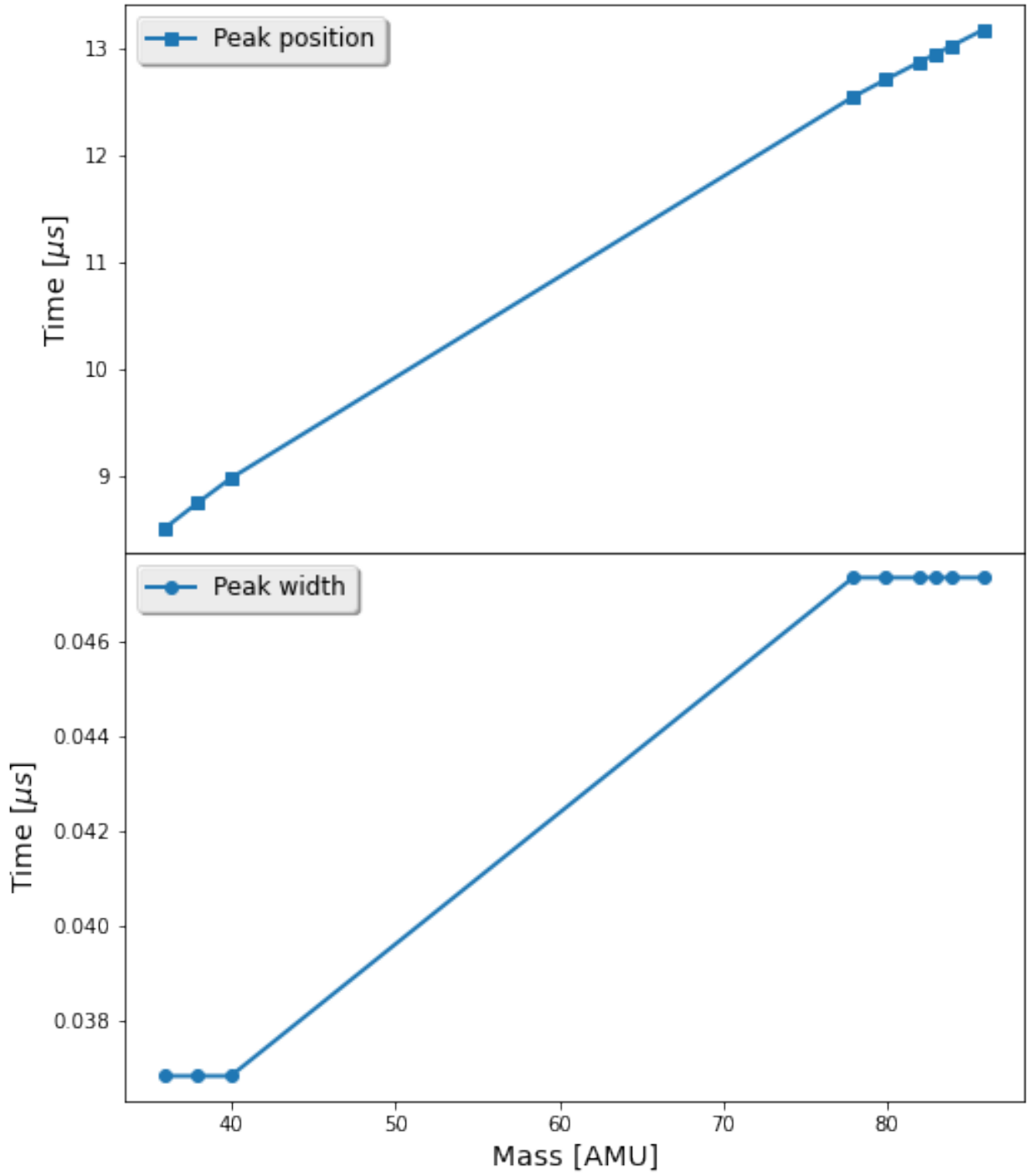
```
pa = np.asarray(pl)
```

We will plot the position and width of each peak as a function of the mass of the fragment:

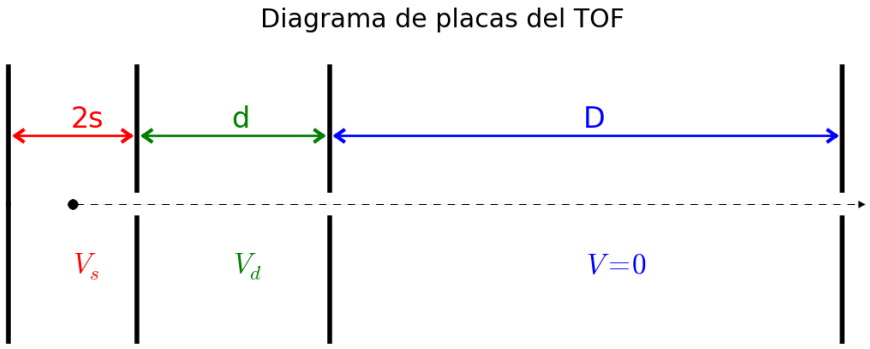
```
x = [T.fragments[k]['M'] for k in pa[:,0]]
ypos = pa[:,2]
ywidth = pa[:,4]
```

```
fig, ax = plt.subplots(nrows=2, sharex=True, figsize=(8, 10))
ax[0].plot(x, ypos, '-s', label='Peak position')
ax[1].plot(x, ywidth, '-o', label='Peak width')
ax[1].set_xlabel(r'Mass [AMU]')

ax[0].set_ylabel(r'Time [$\mu s$]')
ax[1].set_ylabel(r'Time [$\mu s$]')
ax[0].legend(loc='best')
ax[1].legend(loc='best')
plt.subplots_adjust(hspace=0)
```


Peak position
Time [μs]
13
12
11
10
9
Peak width
Time [μs]
0.046
0.044
0.042
0.040
0.038
40
50
60
70
80
Mass [AMU]

1.4 Implementation of time-of-flight calculations



1.4.1 Equations of motion

The equations of motion for each stage of the spectrometer are:

$$v = v_0 + at$$
$$\Delta s = v_0 t + at^2/2 \qquad (a = qE/m)$$

By defining the dimensionless magnitudes

$$u \equiv \sqrt{m} v$$
$$f \equiv a/m = qE\,,$$
$$T \equiv t/\sqrt{m}$$

and solving for time, we get

$$T = \left[\sqrt{u_0^2 + 2\Delta s\, f} - u_0\right] / f$$

Time of flight for each stage is given by ($t_j = \sqrt{m}\, T_j$):

$$T_s = \left[\sqrt{u_0^2 + 2(s_0 - x_0)\, qE_s} - u_0\right] / (qE_s) \qquad u_s = u_0 + q\, E_s T_s$$
$$T_d = \left[\sqrt{u_s^2 + 2\, q\, dE_d} - u_s\right] / (qE_d) \qquad u_d = u_s + q\, E_d\, T_d$$
$$T_D = D/u_d$$

1.4.2 Initial conditions

- Initial velocities are given by the Maxwell-Boltzmann distribution in the direction of the TOF

  $$P(u_0) = \frac{1}{\mathcal{Z}}\, e^{-u_0^2/2k_BT}$$

  with width $\sigma = \sqrt{k_B T}$
- The spatial distribution may be chosen either uniform with width δs, or *normal* with $\sigma = \delta s/2$.
- The time distribution also may be chosen either uniform of width δt, or *normal* with $\sigma = \delta t/2$. The default value is $\delta t = 8$ ns.

1.5 Package API

1.5.1 nist_elem

class tofsim.nist_elem.**Sustancias**(*elementos=''*, *threshold=0.0001*, *isotopes=True*)

Bases: dict

Object describing a list of substances.

Examples

```
>>> elementos = ['N2', 'H2O', 'Ar^{3+}', 'Ar^{+}', 'Ar^2+', 'UF6']
>>> m = Sustancias(elementos, threshold=1.e-3)
>>> print(m)
l                      M          P
-------------  --------  ---------
1H2-16O^{+}     18.0106  99.7341
1H2-18O^{+}     20.0148   0.204953
14N2^{+}        28.0061  99.2733
14N-15N^{+}     29.0032   0.72535
36Ar^{3+}       35.9675   0.3365
36Ar^{+}        35.9675   0.3365
36Ar^{2+}       35.9675   0.3365
40Ar^{3+}       39.9624  99.6003
40Ar^{+}        39.9624  99.6003
40Ar^{2+}       39.9624  99.6003
19F6-235U^{+}  349.034    0.7204
19F6-238U^{+}  352.041   99.2742
```

To get help use:

```
>>> m.show_help()
Create an object Sustancias as:
#
  Sustancias(<elemento o sustancia>)
#
Fields in mass have the meaning:
#
 Z:  str  Atomic Number
 S:  str  Atomic Symbol
 A:  str  Mass Number
 M:  str  Relative Atomic Mass
 P:  str  Isotopic Composition
 W:  str  Standard Atomic Weight
 q:  str  Charge
```

add(*sustancias*)

Returns a dictionary with all the information on the substances added.

Parameters **sustancias** (*str*) – Comma separated list of substances to be added, por instance:

m = add('H20,O2,N2,UF5,UF6')

copy()

Copy the object and returns a new one.

get_lista (*cols=None*)
Return a list with the cols

Parameters **cols** (*tuple/list or None*) – A list of string selecting the fields to included, chosen from *"A,L,l,M,P,S,Z"*

list (*key=None*)
Return a list of elements, optionally filtered with a criteria given by key

Parameters **key** (*None or function*) –

- if key=None returns all elements
- otherwise key has to be a function returning *True* or *False*.

Examples

```
>>> list(key = lambda x:  x['P'] > 0.40)  # returns True if population␣
↪greater than 40%
>>> list(key = lambda x:  x['M'] > 50)    # returns True if Mass smaller than␣
↪than 50 AMU
>>> list(key = lambda x:  x['q'] > 2)     # returns True if charge greater␣
↪than 2
```

loadconf (*fname*)
Load substances from file

Parameters **fname** (*str or Path() object*) – File with definition of substances to add

File should have configuration format as:

```
[34S1]
S=S
L=$^{34}\mathrm{S}$
P=4.25
M=33.9678669
A=34
Z=16

[32S1]
S=S
L=$^{32}\mathrm{S}$
P=94.99
M=31.972071
A=32
Z=16
```

remove (*sustancias*, *criteria='fragment'*)
Remove a fragment

Parameters

- **sustancias** (*str or list of strings*) – substance or fragment to remove
- **criteria** (*str*) – either 'fragment' (the default value) or 'substance':

For the choices of 'criteria', we use:

- 'fragment': e.g: Ar^{2+}, SF6^++, Removes exactly the fragment given
- 'substance': e.g: Ar, SF6, Xe, Remove all fragments with independently of the charge

saveconf (*fname=None*)
Save the substances to a file in a format that may be later read

Parameters

- **self** (*type*) – description
- **fname** (*None or str or pathlib.Path()*) – Name of the file. If *None* prints to screen

show_help ()
Show help on use of Object Sustancias

sort (*order=None*, *reverse=False*)
Sort the substances according to the specified order. The sort is stable.

Parameters

- **self** (*type*) – description
- **order** (*str*) – desired order. One of 'ALMPSZq'
- **reverse** (*bool*) – If *True* will be reversed

to_latex (*cols=None*, *headers=None*, *standalone=False*)
Convert to LaTeX table using tabulate package

Parameters

- **cols** (*list*) – Substance characteristics to include.
- **headers** (*str*) – String to write before data.
- **standalone** (*bool*) – If *True* make a standalone document that may be compiled by LaTeX.

to_pdf (*cols=None*, *headers=None*, *latexcommand='pdflatex'*, *output='tmp.pdf'*)
Convert to pdf format using latex.

Export to a latex file and compiles it.

Parameters

- **self** (*type*) – description
- **cols** (*str*) – Substance characteristics to include
- **headers** (*str*) – Description to include before data
- **latexcommand** (*str*) – command to compile to pdf
- **output** (*str*) – Name of the output file

to_table (*cols=None*, *headers=()*, *tablefmt='simple'*, *floatfmt='g'*, *numalign='decimal'*, *stralign='left'*, *missingval=''*)
Format the list of sustances in Table format.

This is a wrapper around the tabulate function from the tabulate package: https://pypi.python.org/pypi/tabulate. Available formats are: fancy_grid, grid, plain, psql, rst, simple, tsv latex, latex_raw, latex_booktabs, mediawiki, orgtbl, pipe, html

Options *tablefmt*, *floatfmt*, *numalign*, *stralign*, *missingval* are passed to package *tabulate*, if available.

Parameters

- **cols** (*list*) – A list of strings with the characteristics to include. Must be in ALMPSZ
- **headers** (*str*) – String to include before data

- **tablefmt** (*str*) – Format of table
- **floatfmt** (*str*) – Format used to write numbers
- **numalign** (*str*) – Alignment for numbers
- **stralign** (*str*) – Alignment for strings
- **missingval** (*str*) – Value to write when values are missing

to_text()
Convenience function to format as a simple table

tofsim.nist_elem.**analyze_substance**(*subst*, *threshold=0.0001*, *fragments=False*, *isotopes=True*)
Devuelve todas las combinaciones con posibles isotopos (y su poblacion) de la sustancia con la formula dada.

Parameters

- **subst** (*str*) – substance to include (H2O, CO, SF6, N2, ...)
- **threshold** (*float*) – Minimum abundance that has to have an isotope to be included
- **fragments** (*bool*) – If True includes all fragments. For instance from 'CO' -> 'CO', 'C', 'O'
- **isotopes** (*bool*) – If True includes all isotope combinations

Examples:

```
analyze_substance('H2O').keys() =  ['1H1-2H1-16O1', '1H2-17O1', '1H2-16O1', '1H2-
↪18O1']
analyze_substance('CO').keys() =
       ['13C1-17O1', '13C1-18O1', '12C1-16O1', '12C1-18O1', '13C1-16O1', '12C1-
↪17O1']
```

tofsim.nist_elem.**format_listmass**(*rows*, *end='\n'*, *sep=' '*, *header=''*, *footer=''*)
Simple formatting of a list of substances. It is intended to use with the output of *get_lista()*

Parameters

- **rows** (*list*) – List where each row is a list with data on a single element
- **end** (*str*) – End of line string
- **sep** (*str*) – string used as separator within a line
- **header** (*str*) – Header to add before the data
- **footer** (*str*) – Footer to add after the data

tofsim.nist_elem.**loadmass**(*fname*)
Load masses from configuration file

Parameters **fname** (*str or Path()*) – File to read masses from

Returns a dictionary with the elements read

Return type dict

tofsim.nist_elem.**make_label**(*ss*, *fmt='key'*)
Construct the label for a given substance

Parameters

- **ss** (*str*) – Array of tuples [(A1,S1), (A2,S2), ...] for all atoms in the substance
- **fmt** (*str*) – Indicates the format of labels:

```
'key'          -> "14N2"
'Latex'        -> "$^{14}N_{2}$"
'latexsimple'  -> "$N_{2}$"
'latexdoc'     -> r"\ce{^{14}N_{2}}"
'mass'         -> "28"
```

tofsim.nist_elem.**mass2conf**(*masses*)
Format masses for a configuration file

Parameters **masses** (*dict*) – Information on substance(s)

Returns string with the format of a configuration file

Return type str

tofsim.nist_elem.**read_nist_data**(*threshold=0*)
Read data from file *datafile* and fills the dictionary *__NIST_ELEMENTS__* with all the data

Parameters **threshold** (*float*) – Add all elements whose abundance is higher than threshold

1.5.2 tof

class tofsim.tof.**Peaks**
Bases: dict

Simple object describing spectra peaks for the TOF.

It is essentially a dictionary, with a few added convenience methods

Each peak is described by a dictionary whose key is the label of the fragment.

The values are:

- **'index': tuple** has the form (im, i0, ip) with the indexes of the peak (i0) and the two positions where the width is obtained.
- **'position': float** is value of the coordinate where the peak is located, in microseconds.
- **'height': float** is the value of the peak.
- **'width': float** is the width of the peak in microseconds.

tolist()
Returns a list with the data describing the peaks. The form is: ['Substance', 'index', 'position', 'height', 'width']

class tofsim.tof.**ToF**(*substances=None*, ***kwds*)
Bases: object

The ToF object defines and handles all aspects of a Time-of-Flight spectrometer

The following parameters related to the construction and operation are included: Distances, voltages, working temperature, particle velocity fields and dispersion, time duration and size of the ionizing beam are also included.

Examples

```
tof_parameters= {
  's': 0.7,                    # Distance from center to first plate in cm
  'd': 2.54,                   # Distance between plates in cm (second stage)
  'D': 100.,                   # Distance of free flight
```

(continues on next page)

(continued from previous page)

```
  'Vs': 500,                    # Potential Es in eV
  'Vd': 1900,                   # Potential Ed in eV
  'ds': 0.05,                   # Radio del spot del laser in cm
  'dt': 0.008,                  # Laser-pulse duration
  't0': 0.,                     # Offset in time of TOF (experimental)
  'r0': 1.e3,                   # Aperture radius after extraction
  }

# We can create the object with the parameters desired
T = ToF(['Ar', 'N2', 'CO2'],**tof_parameters)

# We can also load parameters from a file
T.load_conf_file('tof.conf')

T.signal()      # Calculate the signals

# Plot the signals
T.make_plot(fname='tof2.png', negative=True, show_legend=True, show_all=True)

# Just printing the object gives all the information on construction and
# condition parameters as well as the substances being simulated
print(T)
```

add_substances (*substances*, *threshold=0.001*)
Add substances to be simulated.

Parameters

- **substances** (*str*) – Comma-separated string, or list of strings, each substance may have the form SF5^{+}, SF5^{++}, SF5^+, S^2+
- **threshold** (*float*) – Minimum abundance of a given isotope to be included (default 1.e-3).

calc_signal (*particles*, *Time*)
Calculo de los histogramas para generar la señal de un grupo de partículas

data_to_array (*unpack=False*)
Conversion from signal to "numpy arrays", sorted by time

Keyword Arguments: unpack – (default False)

get_statistics_peaks (*substances='all'*, *fwidth=0.36787944117144233*)
Find peaks

Parameters

- **substances** (*'all' or list*) –
  - if 'all' find the peak for every fragment in the TOF
  - if substance is a list of strings, each element must be a fragment in TOF
- **fwidth** (*float*) – fraction of the maximum at which to evaluate the full width. For instance, fwidth = 0.5 corresponds to FWHM

get_tof_parameters (*explain=False*)
Returns the parameters of TOF in a dict.

Parameters

- **self** (*type*) – description

- **explain** (*bool*) – If *True* add an ascii graphic with the diagram of the TOF

load_conf_file (*fname*)
Load a configuration file, update tof parameters, and also return masses if present.

make_plot (*especies=None*, *fname=None*, ***kvars*)
Plot the signals in a "standard" form.

Parameters

- **especies** (*dict*) – masses to plot. If no present uses simulated fragments
- **fname** (*string or None (default None)*) – if not None -> Save the figure to "fname"
- **kvars** (*Optional*) –
  - negative = *True/False* : if *True* -> Negative signal
  - graph_all='True/False' (or show_all): if *True* -> Plot individual species
  - show_legend='True/False' : if *True* -> Show the legend

mean_times (*x0=0.0*)
Evaluate the 'main' time of each species, given as if generated in ideal conditions:

- At time t=0
- At rest (null initial velocity)
- At a distance x0 from the center of the extraction plates (s = 0)

remove_substances (*sustancias*)
Remove substances from the simulation

Parameters

- **self** (*type*) – description
- **sustancias** (*str or list of strings*) – substances or fragments to remove

save_conf_file (*fname*, *masas={}*)
Save configuration data for tof and masses to a file

save_data (*fname*)
Save the signal to a data file

set_initial_distribution ()
Initial velocity and position distributions

signal (*particles=None*)
Evaluate the signal that would produce the particles in the ToF

Parameters

- **particles** (*dict each item is a dictionary that should have*) –
  - key: is the label of the mass
  - 'M': Mass (in AMU)
  - 'P': (float in range 0 to 1) Abundance
  - 'L': an (optional, possibly formatted) label, otherwise the key is used
- **particles is None => Usa los fragments** (*If*) –

Returns times – Sets the object variable self.times and also returns its value

Return type numpy array

time_of_extract(*x_0*, *v_0*)
Evaluates the time taken to extract a particle of mass M=1 and charge q=1 with initial velocity v_0 and position x_0 from the extraction plates.

time_of_flight(*x_0*, *v_0*)
Evaluates the time-of-flight of a particle of mass M=1 and charge q=1 with initial velocity v_0 and position x_0.

time_of_flight_bibliog(*x_0*, *v_0*)
Alternative evaluatation of time-of-flight of a particle of mass M=1 and charge q=1 with initial velocity v_0 and position x_0 (from bibliography).

tofsim.tof.**get_one_peak**(*x*, *y*, *fwidth=0.36787944117144233*)
Returns features of a single peak.

Returns a list with:

- Position 0: indexes of position of:
  - lower half height
  - center
  - upper half height
- Position 1: values of x
- Position 2: values of y

CHAPTER 2

Indices y tablas

- genindex
- modindex
- search

Python Module Index

t

tofsim.nist_elem, 14
tofsim.tof, 18

Index

A

add() (*tofsim.nist_elem.Sustancias method*), 14
add_substances() (*tofsim.tof.ToF method*), 19
analyze_substance() (*in module tofsim.nist_elem*), 17

C

calc_signal() (*tofsim.tof.ToF method*), 19
copy() (*tofsim.nist_elem.Sustancias method*), 14

D

data_to_array() (*tofsim.tof.ToF method*), 19

F

format_listmass() (*in module tofsim.nist_elem*), 17

G

get_lista() (*tofsim.nist_elem.Sustancias method*), 14
get_one_peak() (*in module tofsim.tof*), 21
get_statistics_peaks() (*tofsim.tof.ToF method*), 19
get_tof_parameters() (*tofsim.tof.ToF method*), 19

L

list() (*tofsim.nist_elem.Sustancias method*), 15
load_conf_file() (*tofsim.tof.ToF method*), 20
loadconf() (*tofsim.nist_elem.Sustancias method*), 15
loadmass() (*in module tofsim.nist_elem*), 17

M

make_label() (*in module tofsim.nist_elem*), 17
make_plot() (*tofsim.tof.ToF method*), 20
mass2conf() (*in module tofsim.nist_elem*), 18
mean_times() (*tofsim.tof.ToF method*), 20

P

Peaks (*class in tofsim.tof*), 18

R

read_nist_data() (*in module tofsim.nist_elem*), 18
remove() (*tofsim.nist_elem.Sustancias method*), 15
remove_substances() (*tofsim.tof.ToF method*), 20

S

save_conf_file() (*tofsim.tof.ToF method*), 20
save_data() (*tofsim.tof.ToF method*), 20
saveconf() (*tofsim.nist_elem.Sustancias method*), 15
set_initial_distribution() (*tofsim.tof.ToF method*), 20
show_help() (*tofsim.nist_elem.Sustancias method*), 16
signal() (*tofsim.tof.ToF method*), 20
sort() (*tofsim.nist_elem.Sustancias method*), 16
Sustancias (*class in tofsim.nist_elem*), 14

T

time_of_extract() (*tofsim.tof.ToF method*), 21
time_of_flight() (*tofsim.tof.ToF method*), 21
time_of_flight_bibliog() (*tofsim.tof.ToF method*), 21
to_latex() (*tofsim.nist_elem.Sustancias method*), 16
to_pdf() (*tofsim.nist_elem.Sustancias method*), 16
to_table() (*tofsim.nist_elem.Sustancias method*), 16
to_text() (*tofsim.nist_elem.Sustancias method*), 17
ToF (*class in tofsim.tof*), 18
tofsim.nist_elem (*module*), 14
tofsim.tof (*module*), 18
tolist() (*tofsim.tof.Peaks method*), 18