ciaaw

Release 1.1.1

Milan Skocic

Dec 26, 2025

CONTENTS

Standard and abridged atomic weights, isotopic abundances and nuclides' standard atomic weights according to CIAAW.

The latest standard atomic weights were released in 2021 by the [ciaaw](https://www.ciaaw.org). All the values for the atomic weights are provided as double precision reals.

The standard atomic weights (or realtive atomic mass),:math:*A_r(E)*, are extracted from table 1 from Prohaska *et al.* [1]. For the elements that feature an interval for the standard atomic weight, the mean value and the uncertainty are computed using formulas defined in [2].

$$A_r(E) = \frac{a+b}{2}$$
$$u(A_r(E)) = \frac{b-a}{2\sqrt{3}}$$

The standard atomic weights are a dimensionless quantity and thus they need to be multiplied by the molar mass constant $M_u = 1.00000000105 \pm 0.00000000031 g.mol^{-1}$ in order to get the value in $g.mol^{-1}$. See codata for physical constants.

The latest isotopic compositions were released in 2013 by the [ciaaw](https://www.ciaaw.org). All the values for the compositions are provided as double precision reals. The isotopic compositions of the element, are extracted from table 1 from Meija *et al.* [3].

The latest atomic weights for nuclides were released in 2020 by [ciaaw](https://www.ciaaw.org) from Huang *et al.* [4]. All the values for the nuclide atomic weights are provided as double precision reals.

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

ciaaw is a Fortran library providing the standard and abridged atomic weights, the isotopic abundance and the isotopes' standard atomic weights. The data are taken from http://ciaaw.org. C API allows usage from C, or can be used as a basis for other wrappers. Python wrapper allows easy usage from Python.

What have been implemented:

- SAW: Standard Atomic Weights.
- ICE: Isotopic Composition of the Element
- NAW: Nuclides Atomic Weight.

To use *ciaaw* within your fpm project, add the following to your *fpm.toml* file:

```
[dependencies]
iapws = { git="https://github.com/MilanSkocic/ciaaw.git" }
```

1.2 Dependencies

```
gcc>=10.0
gfortran>=10.0
fpm>=0.8
stdlib>=0.5
```

1.3 Installation

A Makefile is provided, which uses fpm, for building the library.

- On windows, msys2 needs to be installed. Add the msys2 binary (usually C:\msys64\usr\bin) to the path in order to be able to use make.
- On Darwin, the gcc toolchain needs to be installed.

Build: the configuration file will set all the environment variables necessary for the compilation

```
chmod +x configure.sh
./configure.sh
make
make test
```

(continues on next page)

(continued from previous page)

```
make install
make uninstall
```

1.4 License

MIT

CHAPTER

TWO

EXAMPLES

2.1 Fortran

```
! EXAMPLE IN FORTRAN
program example_in_f
    use ciaaw
    implicit none

    character(len=8) :: s

    ! ASAW = Abridged Standard Atomic Weight
    ! SAW  = Standard Atomic Weight
    ! ICE  = Isotopic Composition of the Element
    ! NAW  = Nuclide Atomic Weight
    ! U    = Uncertainty

    print '(A)', '########### CIAAW VERSION ##########'
    print *, "version ", get_version()

    print '(A)', '########### CIAAW SAW ##########'
    print '(A10, F10.5)', 'ASAW H   = ', get_saw("H", abridged=.true.)
    print '(A10, F10.5)', 'U ASAW H = ', get_saw("H", uncertainty=.true.)
    print '(A10, F10.5)', 'SAW H    = ', get_saw("H", abridged = .false.)
    print '(A10, F10.5)', 'U SAW H  =  ', get_saw("H", abridged = .false., uncertainty =␣
↪.true.)
    print '(A10, F10.5)', 'ASAW T   = ', get_saw("Tc", abridged=.true.)

    print '(A)', '########### CIAAW ICE ##########'
    print '(A, I3)',        'N ICE H    = ', get_nice("H")
    print '(A, F12.6)',   'ICE H 1    = ', get_ice("H", A=1)
    print '(A, ES23.16)', 'U ICE H 1  = ', get_ice("H", A=1, uncertainty=.true.)
    print '(A, F12.6)',   'ICE H 2    = ', get_ice("H", A=2)
    print '(A, ES23.16)', 'U ICE H 2  = ', get_ice("H", A=2, uncertainty=.true.)
    print '(A, I3)',        'N ICE Tc   = ', get_nice("Tc")
    print '(A, I3)',        'N ICE C    = ', get_nice("C")

    print '(A)', '########### CIAAW NAW ##########'
    print '(A, ES23.16)',   'NAW H 2     = ', get_naw("H", A=2)
    print '(A, ES23.16)',   'U NAW H 2   = ', get_naw("H", A=2, uncertainty=.true.)
    print '(A, I3)',          'N NAW Tc    = ', get_nnaw("Tc")
```

(continues on next page)

(continued from previous page)

```
end program
```

2.2 C

```
/* EXAMPLE IN C */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include "ciaaw.h"


    // ASAW = Abridged Standard Atomic Weight
    // SAW  = Standard Atomic Weight
    // ICE  = Isotopic Composition of the Element
    // NAW  = Nuclide Atomic Weight
    // U    = Uncertainty

int main(void){


    printf("%s\n", "########## CIAAW VERSION ##########");
    printf("version %s\n", ciaaw_get_version());

    printf("%s\n", "########## CIAAW SAW ##########");
    printf("%s %10.5f\n", "ASAW H   = ", ciaaw_get_saw("H", 1, true, false));
    printf("%s %10.5f\n", "U ASAW H = ", ciaaw_get_saw("H", 1, true, true));
    printf("%s %10.5f\n", "SAW H    = ", ciaaw_get_saw("H", 1, false, false));
    printf("%s %10.5f\n", "U SAW H  = ", ciaaw_get_saw("H", 1, false, true));
    printf("%s %10.5f\n", "ASAW Tc  = ", ciaaw_get_saw("Tc", 2, true, false));

    printf("%s\n", "########## CIAAW ICE ##########");
    printf("%s %d\n",      "N ICE H      = ", ciaaw_get_nice("H", 1));
    printf("%s %12.6f\n", "ICE H 1      = ", ciaaw_get_ice("H", 1, 1, false));
    printf("%s %23.16e\n","U ICE H 1    = ", ciaaw_get_ice("H", 1, 1, true));
    printf("%s %12.6f\n", "ICE H 2      = ", ciaaw_get_ice("H", 1, 2, false));
    printf("%s %23.16e\n","U ICE H 2    = ", ciaaw_get_ice("H", 1, 2, true));
    printf("%s %d\n",      "N ICE Tc     = ", ciaaw_get_nice("Tc", 2));
    printf("%s %d\n",      "N ICE C      = ", ciaaw_get_nice("C", 1));

    printf("%s\n", "########## CIAAW NAW ##########");
    printf("%s %23.16f\n", "NAW H 2      = ", ciaaw_get_naw("H", 1, 2, false));
    printf("%s %23.16e\n", "U NAW H 2    = ", ciaaw_get_naw("H", 1, 2, true));
    printf("%s %d\n",       "N NAW Tc     = ", ciaaw_get_nnaw("Tc", 2));

    return EXIT_SUCCESS;
    }
```

2.3 Python

```python
sys.path.insert(0, "../py/src/")
import pyciaaw

# ASAW = Abridged Standard Atomic Weight
# SAW  = Standard Atomic Weight
# ICE  = Isotopic Composition of the Element
# NAW  = Nuclide Atomic Weight
# U    = Uncertainty

print("########## CIAAW VERSION ##########")
print("version ", pyciaaw.__version__)

print("########## CIAAW SAW  ##########")
print("ASAW H   = ", pyciaaw.get_saw("H"))
print("U ASAW H = ", pyciaaw.get_saw("H", uncertainty=True))
print("SAW H    = ", pyciaaw.get_saw("H", abridged=False, uncertainty=False))
print("U SAW H  = ", pyciaaw.get_saw("H", abridged=False, uncertainty=True))
print("ASAW Tc  = ", pyciaaw.get_saw("Tc"))


print("########## CIAAW ICE  ##########")
print("N ICE H    = ", pyciaaw.get_nice("H"))
print('ICE H 1    = ', pyciaaw.get_ice("H", A=1))
print('U ICE H 1 = ', pyciaaw.get_ice("H", A=1, uncertainty=True))
print('ICE H 2    = ', pyciaaw.get_ice("H", A=2))
print('U ICE H 2 = ', pyciaaw.get_ice("H", A=2, uncertainty=True))
print("N ICE Tc   = ", pyciaaw.get_nice("Tc"))
print("N ICE C    = ", pyciaaw.get_nice("C"))

print("########## CIAAW NAW  ##########")
print('NAW H 2    = ', pyciaaw.get_naw("H", A=2))
print('U NAW H 2 = ', pyciaaw.get_naw("H", A=2, uncertainty=True))
print("N NAW Tc   = ", pyciaaw.get_nnaw("Tc"))
```

CHAPTER THREE

APIS

3.1 Fortran

https://milanskocic.github.io/ciaaw/ford/index.html

3.2 C API

```c
#ifndef CIAAW_H
#define CIAAW_H
#if _MSC_VER
#define ADD_IMPORT __declspec(dllimport)
#else
#define ADD_IMPORT
#endif
#include <stdbool.h>

extern char* ciaaw_get_version(void);
extern double ciaaw_get_saw(char *s, int n, bool abridged, bool uncertainty);
extern double ciaaw_get_ice(char *s, int n, int A, bool uncertainty);
extern int ciaaw_get_nice(char *s, int n);
extern double ciaaw_get_naw(char *s, int n, int A, bool uncertaintuy);
extern int ciaaw_get_nnaw(char *s, int n);

#endif
```

3.3 Python

SAW: Standard Atomic Weights.

ICE: Isotopic Compositions of the element.

NAW: Nuclide's Atomic Weights.

`get_ice`(*s: str*, *A: int*, *uncertainty: bool = False*)

Get the isotopic composition of the element s for the mass number A.

Parameters

s: str
Element symbol.

A: int
Mass number.

uncertainty: bool, optional
Flag to get the uncertainty instead of the value. Default to False.

Returns

ice: float
Returns NaN if the provided element or the mass number A are incorrect or -1 if the element does not have an ICE.

`get_naw`(*s: str*, *A: int*, *uncertainty: bool = False*)

Get the atomic weight of the nuclide s for the mass number A.

Parameters

s: str
Element symbol.

A: int
Mass number.

uncertainty: bool, optional
Flag to get the uncertainty instead of the value. Default to False.

Returns

naw: float
Returns NaN if the provided element or A are incorrect or -1 if the element does not have an NAW.

`get_nice`(*s*)

Get the number of isotopes in ICE of the element s.

Parameters

s: str
Element symbol.

Returns

nice: int
Returns -1 if the provided element is incorrect.

`get_nnaw`(*s*)

Get the number of nuclides in NAW of the element s.

Parameters

s: str
Element symbol.

Returns

nnaw: int
Returns -1 if the provided element is incorrect.

`get_saw`(*s: str*, *abridged: bool = True*, *uncertainty: bool = False*) → float

Get the standard atomic weight of the element s.

Parameters

s: str
Element symbol.

abridged: bool, optional
Flag for the abridged value. Default to True.

uncertainty: bool, optional
Flag to get the uncertainty instead of the value. Default to False.

CHAPTER

FOUR

CHANGELOG

4.1 1.1.1

- Refactoring the `configure.sh` script.
- Remove support for 3.14t. No official release on python.org.
- If binaries for Python 3.14t are needed you need to compile them by yourself.

4.2 1.1.0

- Remove support for Python 3.9 and add support for Python 3.14(t).

Full changelog

4.3 1.0.0

- Switch to UCRT64 for Windows binaries.
- Switch documentation to `sphinx` with `fspx` (development version).
  - `fspx` uses Ford as a backend.
- Drop Ford documentation output

Full changelog

4.4 0.5.1

- Code refactoring.
- Documentation update.

Full changelog

4.5 0.5.0

- API break:
  - Data for standard atomic weights are no directly accessed.
  - SAW, ICE and NAW are stored in derived types for each elements.
  - Each element is stored in a periodic table.

  - Element properties are now accessed through getters.
- Doc update.

Full changelog available at github

4.6 0.4.3

- Refractoring.
- Documentation update.

Full changelog available at github

4.7 0.4.2

- Refractoring.
- Documentation update.
- Merge back C API and Python wrapper.

Full changelog available at github

4.8 0.4.1

- Fix bug in max value for saw.

Full changelog available at github

4.9 0.4.0

- Fix type error in tests
- Refractoring
- Documentation update.

Full changelog available at github

4.10 0.3.0

- Code refactoring.
- Move C API and Python wrapper to their own repositories.
- Improve documentatinon.

Full changelog available at github

4.11 0.2.0

- Fix error in phosphorus name.
- Possibility to have standard atomic weights for several years.

Full changelog available at github

4.12 0.1.0

- All elements from the periodic table added for the saw module.
- They are implemented as parameter derived type.

Full changelog available at github

BIBLIOGRAPHY

[1] Thomas Prohaska, Johanna Irrgeher, Jacqueline Benefield, John K. Böhlke, Lesley A. Chesson, Tyler B. Coplen, Tiping Ding, Philip J. H. Dunn, Manfred Gröning, Norman E. Holden, Harro A. J. Meijer, Heiko Moossen, Antonio Possolo, Yoshio Takahashi, Jochen Vogl, Thomas Walczyk, Jun Wang, Michael E. Wieser, Shigekazu Yoneda, Xiang-Kun Zhu, and Juris Meija. Standard atomic weights of the elements 2021 (IUPAC Technical Report). *Pure and Applied Chemistry*, 94(5):573–600, 2022. URL: https://doi.org/10.1515/pac-2019-0603 (visited on 2023-02-20), doi:10.1515/pac-2019-0603.

[2] M.H. van der Veen, J. Meia, and D. Brynn Hibbert. Interpretation and use of standard atomic weights (IUPAC Technical Report). *Pure and Applied Chemistry*, 93(5):629–646, 2021.

[3] Juris Meija, Tyler B. Coplen, Michael Berglund, Willi A. Brand, Paul De Bièvre, Manfred Gröning, Norman E. Holden, Johanna Irrgeher, Robert D. Loss, Thomas Walczyk, and Thomas Prohaska. Isotopic compositions of the elements 2013 (IUPAC Technical Report). *Pure and Applied Chemistry*, 88(3):293–306, 2016. URL: https://doi.org/10.1515/pac-2015-0503 (visited on 2024-06-07), doi:10.1515/pac-2015-0503.

[4] W.J. Huang, Meng Wang, F.G. Kondev, G. Audi, and S. Naimi. The ame 2020 atomic mass evaluation (i). evaluation of input data, and adjustment procedures*. *Chinese Physics C*, 45(3):030002, mar 2021. URL: https://dx.doi.org/10.1088/1674-1137/abddb0, doi:10.1088/1674-1137/abddb0.

PYTHON MODULE INDEX

INDEX