Eur. Phys. J. D (2024)78:93
https://doi.org/10.1140/epjd/s10053-024-00879-7

THE EUROPEAN PHYSICAL JOURNAL D

Regular Article - Atomic Physics

# Xstar atomic database: the PyXstar package

Claudio Mendoza[1,2,a], Timothy R. Kallman[1,b], Ralf Ballhausen[1,3,c], Anna Ogorzałek[1,3,d], Randall Dannen[4,e], and Javier A. García[1,f]

[1] X-Ray Astrophysics Laboratory, Code 662, NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA
[2] Southeastern Universities Research Association (SURA), Washington, DC 20005, USA
[3] Department of Astronomy, University of Maryland, College Park, MD 20742, USA
[4] Department of Physics and Astronomy, University of Nevada, Las Vegas, Las Vegas, NV 90154, USA

Received 26 March 2024 / Accepted 3 June 2024


**Abstract.** We present a progress report on the development of `PyXstar`, a Python package to manage the data (input, output, intermediate, atomic database, and model-grids) associated with the XSTAR code for treating photoionized and collisionally ionized plasmas. The `PyXstar` modular structure and database retrieval scheme are described, and its functionality is illustrated with Python functions and classes for performing database searches. We briefly compare `PyXstar` with two other Python spectrum modeling tools: `PyNeb` and `PyAtomDB`.

## 1 Introduction

XSTAR [1,2] is a plasma modeling code widely used by the astronomical X-ray community. It generates high-resolution synthetic spectra by deriving from comprehensive application of relevant physical processes the ionic charge states and level populations of the plasma atomic constituents assuming steady-state equilibrium in a photoionized or collisionally ionized gas. The scientific accuracy of XSTAR thus relies on an extensive atomic database meticulously compiled in the past 20 years from in-house computations (see [3,4] and list of references therein) and a variety of other sources (e.g., [5–7]). Due to its extensive treatment of photoionization, particularly at the high energies associated with the K edges of ions with $Z \leq 30$, and of high-density effects [8], XSTAR is useful in the modeling of luminous compact objects such as accreting black holes and neutron stars.

With the establishment of the big data era [9–11], astronomical spectrum modeling has evolved from running monolithic codes with well-prescribed data outputs to custom workflows, often deployed in high-performance-computing (HPC) environments, which utilize different tools put together with Python. Furthermore, Python is often used beyond model fitting to understand the details of physics of the best-fitting model to constrain with greater versatility the physical mechanisms in hand or discover new ones. We are following this trend by upgrading XSTAR into a general-purpose calculator of non-LTE plasmas (in particular photoionized and collisionally ionized gases) based on Python tools referred to collectively as `PyXstar`. Similar initiatives have already been pursued by other spectrum modeling codes such as `PyAtomDB`[1] [12], `PyNeb`[2] [13,14], and `ChiantiPy`[3] [6], which the astronomical community has well received.

`PyXstar` currently comprises four independent modules allowing the user to: map and exploit the XSTAR FITS[4] output files; get "under the hood" to decipher the intricate intermediate steps of synthetic spectrum generation; run model grids; and interact with the atomic database. Regarding the latter, the user is encouraged to visualize, revise, and modify the current datasets if custom-made versions are desired. Moreover, the curation, provenance, and evaluation (accuracy and completeness) of the database are of prime importance. Steps are being carried out to facilitate database main-

Timothy R. Kallman, Ralf Ballhausen, Anna Ogorzałek, Randall Dannen, Javier A. García have contributed equally to this work.

Dr. Claudio Mendoza passed away during the final stages of the preparation of this manuscript. His obituary will appear separately in the journal Atoms.

[a] e-mail: claudiom07@gmail.com
[b] e-mail: timothy.r.kallman@nasa.gov (corresponding author)
[c] e-mail: ballhaus@umd.edu
[d] e-mail: ogoann@umd.edu
[e] e-mail: randall.dannen@unlv.edu
[f] e-mail: javier.a.garciamartinez@nasa.gov

[1] https://atomdb.readthedocs.io/en/master/.
[2] https://pypi.org/project/PyNeb/.
[3] https://pypi.org/project/ChiantiPy/.
[4] https://fits.gsfc.nasa.gov/.

Published online: 09 July 2024

tenance by storing the master version in a relational model accessible through the SQLite[5] library. The ultimate goal of `PyXstar` is to enable scientists to visualize easily the atomic species and specific spectral features that the plasma imprints on the observed spectrum, and additionally, to understand the assumed equilibrium by including, for example, which processes contribute to the heating and cooling.

The present report gives an overview of the work in progress. In Sect. 2, we describe the technical profile of the XSTAR database, its data curation policies, and the `PyXstar` database retrieval scheme. We go over the package modular structure in Sect. 3, illustrating its functionality in Sects. 4–5 with database retrieval functions and objects. Brief comparisons with other Python modules are given in Sect. 6 followed by our conclusions in Sect. 7.

## 2 The XSTAR database

XSTAR contains an atomic database of around 870 MB of radiative and collisional data to empower the modeling of plasmas of light chemical elements ($Z \leq 30$) at electron temperatures $T > 10^4$ K and densities $n_e \lesssim 10^{24}$ cm$^{-3}$. The master database currently consists of flat ASCII files that are transcribed to a FITS file before each public version is released. It is based on four long arrays of integers, floats, and characters that are read into main memory when the code is invoked. A pointer structure is derived after this initial step to ensure fast direct database access during the complete plasma modeling process.

The XSTAR database curation policies regarding new data and maintenance specify that the tabulations and units of the original sources are maintained. As a result, the database spans an extensive variety of rate and data types that have been inventoried in [2,4]. In the face of this complexity, `PyXstar` aims to give the user-expedited data access and manipulation capabilities through a series of easy-to-use Python functions and object classes.

The `PyXstar` database retrieval scheme follows the *view–table* framework implemented in the development of `TOPbase`,[6] the Opacity Project atomic database [15] (see Fig. 1). In this approach, a disk data search is performed through a single command portrayed by a series of positional arguments referred to as the *descriptor* that leads to a *view* of the database in main memory. The view can then be further manipulated (e.g., row selection, exclusion, and sorting) in main memory through the *table* logical data structure attending the user's ultimate requirements. Views and tables can be displayed on the screen, printed, or disk stored and retrieved. Table graphic processing is also considered. The original `TOPbase` user interface was command-based and accessed remotely from the database host at the Centre de Données astronomiques de Strasbourg (CDS[7]) through the `SSH` (`telnet` at the time) network protocol. However, with the advent of the World Wide Web in the early 1990s, the user interface was rapidly transcribed to the `HTML` markup language reducing the functionality of the table structure [16].

A key difference of the `PyXstar` data retrieval scheme is that database searches are performed in main memory. Views are procured through a Python function by means of descriptors specified in terms of non-positional keyword arguments. The table data structure is fully exploited with Pandas dataframes[8] or Astropy data tables[9] for comprehensive, easy-to-use big-data analysis. Further data manipulation capabilities are brought about by the introduction of data *objects* implemented through Python classes. A central class is `Ion` bearing the basic attributes of an ionic species such as its atomic number, electron number, charge, ionization potential, and ground-level configuration. This class also activates two subclasses, `Level` and `Transition`, with a second layer of attributes and methods: for the former, the level configuration, spin multiplicity, total orbital angular momentum, statistical weight, energy, radiative and Auger widths, and for the latter, the transition wavelength, $A$-value, Auger rate, and effective collision strengths. Inter-class methods are also implemented; for instance, the `Level` subclass also includes the level radiative and Auger widths, both derived through the `Transition` subclass.

## 3 PyXstar

The `PyXstar` blueprint currently consists of four modules that concentrate on well-defined tasks to enhance the user experience and plasma modeling scope of the XSTAR code. We briefly describe them in Sects. 3.1–3.4.

### 3.1 PyXstar_model

This module streamlines the running of XSTAR models from a Jupyter[10] notebook in different computing environments and provides Python functions to access and manipulate the data contained in the FITS output files. Ample use is made of the `astropy.io.fits`[11] Python module.

The first version of this module addresses three running environments: a local HEASoft[12] installation; a local Docker[13] container; and a remote Docker container through SSH tunneling.[14] Input parameters can be specified through a Python dictionary or an

[5] https://www.sqlite.org/.
[6] https://cds.unistra.fr//topbase/topbase.html.
[7] https://cdsweb.u-strasbg.fr/.
[8] https://pandas.pydata.org/.
[9] https://docs.astropy.org/en/stable/table/.
[10] https://jupyter.org/.
[11] https://docs.astropy.org/en/stable/io/fits.
[12] https://heasarc.gsfc.nasa.gov/docs/software/heasoft/.
[13] https://www.docker.com/.
[14] https://www.ssh.com/academy/ssh/tunneling.

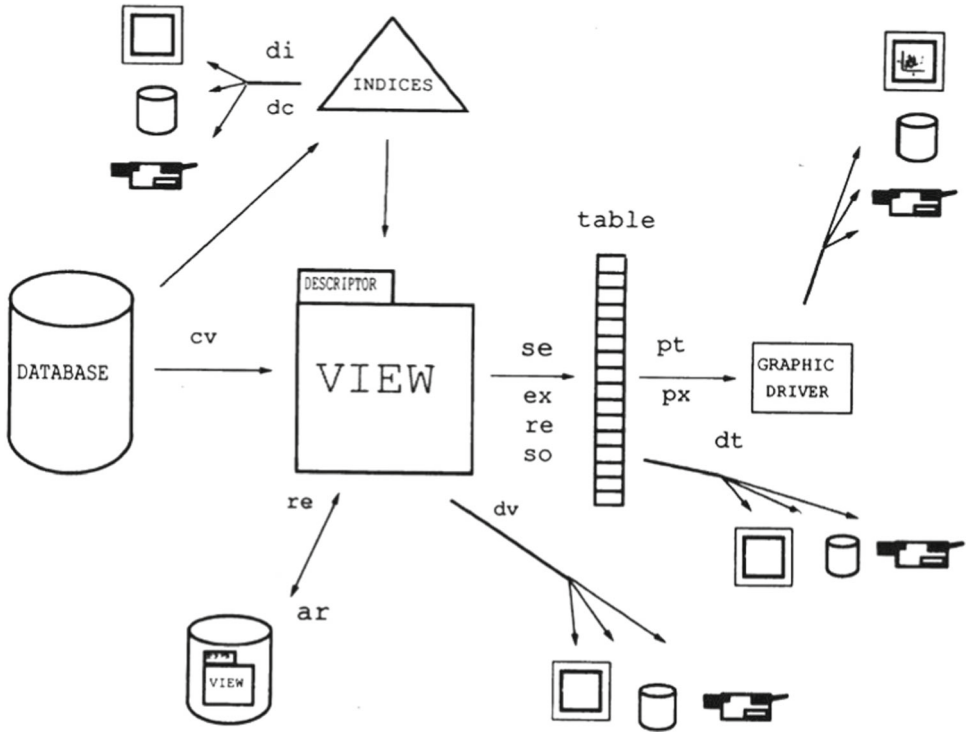



**Fig. 1** Data manipulation scheme of `TOPbase`, the Opacity Project atomic database, based on two data structures: the *view* and the *table*. Retrieved datasets can be viewed or plotted in a terminal, printed, or stored on disk. Figure is reproduced from Fig. 2 of [15] with permission from Revista Mexicana de Astronomía y Astrofísica (http://www.astroscu.unam.mx/RMxAA)

interactive Jupyter widget (`IPyWidgets`[15]). Datasets include among others: plasma parameters; ionic abundances and column densities; heating and cooling rates; and line, radiative recombination, and continuum spectra.

### 3.2 `PyXstar_uth`

We show in Fig. 2 the flowchart of an XSTAR model consisting of different plasma spatial zones. From a set of input parameters that include, among others, the elemental abundances, temperature, density, ionization parameter, luminosity, and turbulence velocity, the code reads from disk the atomic database, works out a set of pointers to access its components, and determines the radiation flux. It then proceeds to compute the ionization balance and level populations by imposing thermal equilibrium that results in a temperature and tabulations of the line and continuum opacities and emissivities. The inter-zone heat transfer is worked out, and the code proceeds with the following iteration. Several passes of the zonal cycle can be prescribed to ensure the desired convergence.

The aim behind `PyXstar_uth` is to compartmentalize this bicycle into Python functions and classes to give the user interactive and scripting potential at every stage and to foster access of the intermediary data unavailable when running the code in the `PyXstar_model` mode. The Python functions have been coded as wrappers of the Fortran XSTAR routines using the `F2PY`[16] interface generator. We are also rewriting these Python wrappers in `Cython`[17] to compare performance and with the intention to port XSTAR to a more modern programming language.

Two types of data are addressed by `PyXstar_uth`: (i) the basic atomic parameters encompassing the database and (ii) derived data computed in plasma models such as transition rates, heating–cooling rates, level populations, ionization fractions, opacities, and emissivities. In the present report, we are mainly concerned with the initial setup stage (see Fig. 2) when the atomic database is loaded into main memory; thus, the chosen Python functions and classes are those destined to display information about its components and methods, which are further described in Sect. 4.

### 3.3 `PyXstar_grid`

Within the context of the HEASoft spectral fitting code XSPEC,[18] grids of XSTAR models can be implemented with the XSTAR2XSPEC Perl script. From user specifications, the FTOOLS XSTINITABLE and XSTAR2TABLE create a job list that is run and consolidated into table models. A parallel version of this procedure

[15] https://ipywidgets.readthedocs.io/en/stable/.

[16] https://numpy.org/doc/stable/f2py/.

[17] https://cython.org/.

[18] https://heasarc.gsfc.nasa.gov/xanadu/xspec/.

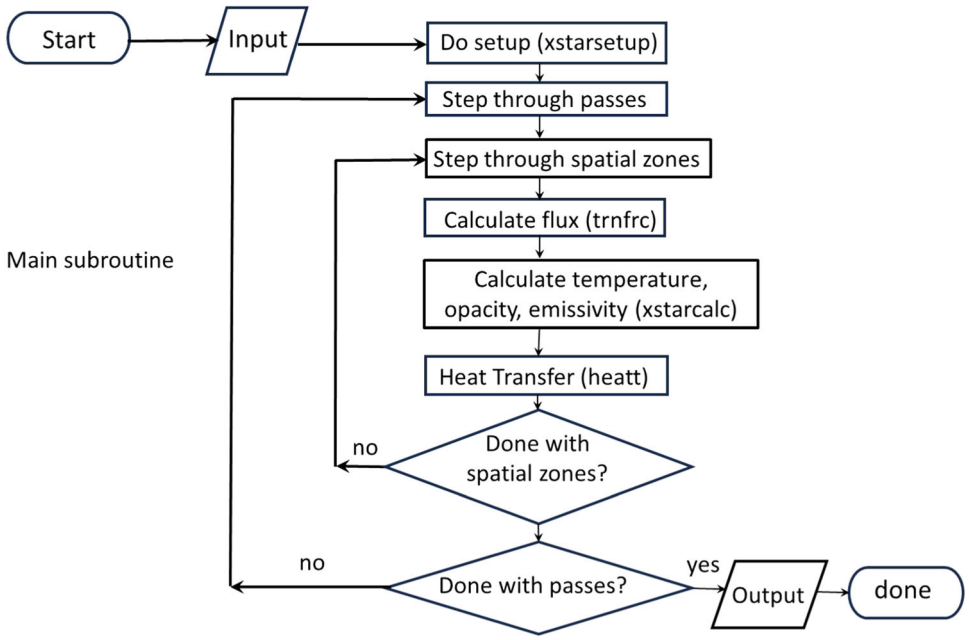



**Fig. 2** XSTAR model flowchart showing the main stages of the computation and the spatial zones and passes in the iterative cycles

has been devised independently in the C++ programming language with the Message Passing Interface (MPI) [17]. In the present work, we have developed a Python version of the XSTAR2XSPEC script based on the FTOOLS Python wrappers of `HeasSoftPy`,[19] which runs in parallel in a multi-core processor through the `multiprocessing`[20] module.

### 3.4 `PyXstar_DB`

As mentioned in Sect. 2, the master XSTAR database is structured as a collection of flat ASCII files that do not conform to a relational model, thus making maintenance and updating time-consuming. `PyXstar_DB` remaps the database onto an `SQLite` engine diminishing record indexing and reinforcing table relationships through keys. Dataset provenance and completeness are high-priority issues. Implementation of the new database is leading to a more comprehensive and simplified inventory of the XSTAR data and rate types.

## 4 Database function

The function in `PyXstar_uth` to display components of the atomic database

$$\texttt{get\_data}(\texttt{ion}, \texttt{rtype}, \texttt{llo}, \texttt{lup}) \quad (1)$$

requires the following keyword arguments: `ion` identifying of the ionic species in XSTAR notation (e.g., 'o_iii') or the tuple $(Z, N)$ with its atomic $(Z)$ and electron $(N)$ numbers; `rtype` the dataset of interest specified by a character acronym or the corresponding XSTAR rate-type integer [3,4]; and the integers `llo` and `lup` denoting, respectively, the lower- and upper-level indices of the ion. The default for the lower level (`llo` $= 0$) returns all the transitions with `llo` $<$ `lup` while that for the upper level (`lup` $= 0$) returns all the transitions with `lup` $>$ `llo`. The option `llo` $=$ `lup` $= 0$ returns all the transitions of the ion.

`PyXstar_uth` is invoked in the usual Python manner uploading the atomic database to main memory where it will reside for the rest of the user interactions. The graphic library `Matplotlib` and the module of mathematical functions `math` are also conveniently imported in this initial step:

```
import pyxstar as px
import matplotlib.pyplot as plt
import math
```

```
Xstar database has been loaded. PyXstar is ready
to go.
```

We have adopted here and in Sects. 4.1–4.3 the cell-based format of the Jupyter notebook interface. We illustrate the functionality of `get_data`() with database searches of energy levels, radiative bound–bound transitions, and photoionization cross sections.newpage

[19] https://heasarc.gsfc.nasa.gov/lheasoft/heasoftpy/.
[20] https://docs.python.org/3/library/multiprocessing.html.

### 4.1 Energy levels

The function displays the attributes of energy level `llo` in `ion`. The default, `llo` = 0, displays all the energy levels, while `llo` = −1 does so for the continuum; i.e., the ionization potential of the species. As an example in the cell below, we search for all the levels of C-like O III:

```
y = px.get_data(ion='o_iii',
    rtype='LV')
print(y)
```

```
      ion  Z  N  index           level  n  (2S+1)  L  stat_wt      energy
0   o_iii  8  6      1        2p2.3P_0  2       3  1      1.0    0.000000
1   o_iii  8  6      2        2p2.3P_1  2       3  1      3.0    0.014054
2   o_iii  8  6      3        2p2.3P_2  2       3  1      5.0    0.038022
3   o_iii  8  6      4        2p2.1D_2  2       1  2      5.0    2.512230
4   o_iii  8  6      5        2p2.1S_0  2       1  0      1.0    5.351830
..    ... .. ..    ...             ... ..     ... ..      ...         ...
74  o_iii  8  6     75    1s1.2p5.3P_1  2       3  1      3.0  575.461000
75  o_iii  8  6     76    1s1.2p5.3P_0  2       3  1      1.0  575.501000
76  o_iii  8  6     77    1s1.2p5.1P_1  2       1  1      3.0  578.680000
77  o_iii  8  6     78  superlevel_[K]  2       1  1      3.0  594.140000
78  o_iii  8  6     79       continuum  2       2  1      2.0   54.934000

[79 rows x 10 columns]
```

The search output is displayed in a Pandas dataframe whose concise and comprehensive data visualization profile is remarkable. It may also be noted that the level electron configuration in the XSTAR database abides by the Witthoeft notation delineated in "Appendix A."

### 4.2 Bound–bound radiative transitions

Radiative attributes (wavelength, $A$-value, and $gf$-value) for a bound–bound transition between levels `llo` and `lup` of `ion` are also listed with this function. In the cell below, we retrieve the respective data for the transition between levels 7 and 3 in O III:

```
y = px.get_data(ion='o_iii',
    rtype='LA',llo=3,lup=7)
print(y)
```

```
   lrtyp  ltyp    ion  lup  llo     upper_lev lower_lev  wavelength  \
0      4    50  o_iii    7    3  2s1.2p3.3D_2  2p2.3P_2   835.10199

       a_value  gf_value
0  173000000.0    0.0887
```

It may be seen that the output dataframe can conveniently list records longer than the monitor width by using the continuation character '/'.

### 4.3 Photoionization cross sections

The function retrieves fits or tabulations of the photoionization cross sections $\sigma(E)$ as a function of energy for the transition between level `llo` of the parent `ion` leaving the daughter ion in level `lup`. We search below for the photoionization cross section of the ground level of O III leaving O IV in its ground level:

```
y = px.get_data(ion='o_iii',
    rtype='PI',llo=1,lup=1)
```

```
Data products computed = 2
[0] Photoionization transition identifiers
[1] Photoionization cross sections
```

Two data products are returned, $y[0]$ listing the transition identifiers

```
print(y[0])
```

```
   ltyp  lrtyp  Z parent  llo parent_lev daughter  lup daughter_lev
0    49      7  8  o_iii    1   2p2.3P_0      o_iv    1   2p1.2P_1/2
1    53      7  8  o_iii    1   2p2.3P_0      o_iv    1   2p1.2P_1/2
```

showing that the cross section has two contributions (data types 49 and 53):

```
print(y[1])
```

```
[        energy  x_section
0     36.669495   0.001375
1     37.757496   0.000977
2     37.815495   0.000950
3     37.832497   0.000950
4     37.841496   0.000911
..          ...        ...
475   41.252495   0.000843
476   61.952496   0.000250
477   82.652496   0.000105
478  103.353490   0.000054
479  124.053500   0.000031

[480 rows x 2 columns],        energy  x_section
0     0.000000   1.428000
1     0.225300   1.754000
2     0.365200   1.629000
3     0.605100   1.315000
4     0.888600   1.166000
5     1.896000   0.821300
6     2.619000   0.655700
7     3.158000   0.561300
8     6.038690   0.093351
9     9.458380   0.032764
10   12.878100   0.015945
11   16.297800   0.009204
12   19.717400   0.005901
13   23.137100   0.004063
14   26.556801   0.002946
15   29.976500   0.002220
16   33.396198   0.001726
17   36.669399   0.001375]
```

The correspondence between the `y[0]` transition identifiers and `y[1]` tabulations is

$$\texttt{y[0].iloc}[i] \rightleftharpoons \texttt{y[1]}[i] \tag{2}$$

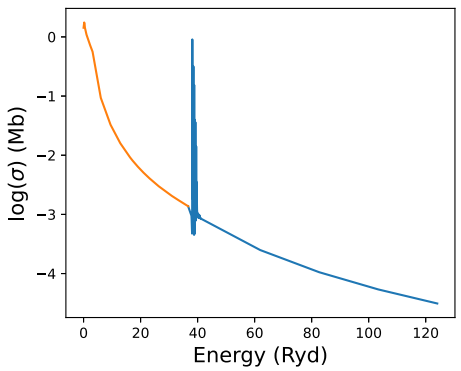
We can then plot the cross section using the `Matplotlib` graphic library,

```
for i in range(2):
    plt.plot(y[1][i]['energy'],y[1][i]['x_section'].\
        apply(lambda x: math.log10(x)))
    plt.xlabel('Energy (Ryd)',fontsize=15)
    plt.ylabel(r'$\log(\sigma)$ (Mb)',fontsize=15)
plt.show()
```

showing the two contributions. This plot is useful to check matching accuracy in a multi-component method frequently used in the XSTAR database to represent cross sections over a wide energy range. The figure also illustrates how data in Pandas dataframes are readily plotted with `Matplotlib`; e.g., applying the anonymous Python lambda function to process column values.

## 5 Data objects

Further programming pathways to make the most of the XSTAR database components can be rendered through data objects in terms of Python classes. We introduce the `Ion` class exhibiting several attributes and two subclasses: `Level` and `Transition`. To examine their possibilities, we instantiate the class with O III and list its attributes



```
y = px.Ion('o_iii')
print('List of class attributes:',y.__dict__)
```

```
List of class attributes: {'atom_num': 8, 'charge': 2, 'electron_num': 6, 'ion
_pot': 54.934, 'ground_conf': '2p2.3P_0', 'level': <pyxstar.Level object at 0x
7fcba236e5c0>, 'transition': <pyxstar.Transition object at 0x7fcba236ec20>}
```

The class attributes of the instantiated ion may be accessed with the usual dot notation; for instance, the ionization potential

```
print('Ion IP =',y.ion_pot)
```

```
Ion IP = 54.934
```

Similarly, the methods of the `Level` and `Transition` subclasses can be inventoried

```
print('List of Level methods:', [m for m in dir(px.Level)\
      if not m.startswith('__')])
```

```
List of Level methods: ['L', 'auger_width', 'conf', 'energy', 'mult', 'rad_wid
th', 'stat_wt']
```

```
print('List of Transition methods:', [m for m in \
     dir(px.Transition) if not m.startswith('__')])
```

```
List of Transition methods: ['auger_rate', 'avalue', 'wavelength']
```

and accessed

```
y.level.conf(llo=3)
```

```
'2p2.3P_2'
```

```
y.transition.wavelength(llo=3,lup=4)
```

```
5008.98
```

The default `llo = lup = 0` gives the whole dataset as a list of tuples.

## 6 Comparison with other spectral modeling tools

We briefly review here two spectral modeling tools that use Python interfaces to display their atomic databases.

### 6.1 PyNeb

PyNeb [13,14] is a Python package widely used in nebular physics for the analysis of emission lines. Relying on an extensive atomic database, it solves the equilibrium equations to obtain the level populations, critical densities, and line emissivities. By comparing the theoretical emissivity ratios with observed line intensity ratios, the electron temperature and density and the chemical abundances may be estimated.

In the object-oriented architecture of PyNeb, the ion object is implemented through the `Atom` and `RecAtom` classes with a stream of methods to tabulate and plot the radiative and collisional data. For instance, to list the effective collision strength for the transition between levels 2 and 1 in O III at $T = 10^4$ K, the atomic data files are fetched, the ion is instantiated, and the parameter is listed:

```
import pyneb
pyneb.atomicData.getDataFile('O3')
O3 = pyneb.Atom('O',3)
O3.getOmega(1.0e4,2,1)
```

```
array(0.5421)
```

A salient aspect of the database curation of this system is its reliance on collections rather than selections of available datasets. The latter are transcribed to a prescribed format and laboriously re-indexed to match the NIST energy-level order [7], which is standard throughout the database. The default datasets are carefully appointed, but their substitution with other listed datasets in any spectral model is straightforward. This option makes PyNeb particularly useful in determining the impact of the atomic data on nebular plasma models [18] and in atomic data assessment [19,20].

Atomic datasets in XSTAR, on the other hand, are mostly custom-computed or selected from other databases, and when updating, those to be replaced

are discarded. Furthermore, it is considerably more difficult to replace datasets (e.g., partial photoionization cross sections) piecemeal for an $N$-electron ion in XSTAR as they can be tightly coupled to those of the $(N-1)$-electron system.

### 6.2 PyAtomDB

AtomDB [12] is an atomic database compiled to underpin the spectral modeling of collisionally excited plasmas mainly in the ultraviolet and X-ray. Spectral modeling is carried out with the PyAtomDB Python package that replaced the previous APEC C code. Although it is mainly tailored to compute derived data from the database, e.g., rate coefficients, level populations, and charge state distributions, PyAtomDB also allows the interactive viewing of the raw atomic parameters.

For instance, to get the $A$-value of transition 2–1 in O III, the respective atomic data file is loaded to list its headings and attributes:

```
import pyatomdb
Z = 8
z1 = 3
b = pyatomdb.atomdb.get_data(Z,z1,'LA')
bb = b[1].data[b[1].data['UPPER_LEV']==2]
print(bb.names)
```

```
['UPPER_LEV', 'LOWER_LEV', 'WAVELEN', 'WAVE_OBS', 'WAVE_ERR', 'EINSTEIN_A', 'E
IN_A_ERR', 'WAVE_REF', 'WV_OBS_REF', 'EIN_A_REF']
```

```
print(bb)
```

```
[(2, 1, 883563.94, nan, nan, 2.597e-05, 0., '2009A&A...498..915D', '', '2009A&
A...498..915D')]
```

It may be noted that the requested data type is indicated by a character string, 'LA' (we have implemented similar acronyms to denote atomic data types in PyXstar) and that the source references are included in the transition attributes.

An interesting aspect of the PyAtomDB database views is that the requested dataset is downloaded directly from the host server the first time it is used to be then addressed locally for further local manipulation; thus, the user has the possibility of managing the database files in home file space. In contrast, the complete PyXstar atomic database is on local disk and uploaded to main memory when the module is imported. This leads to better performance but involves larger data volumes. Moreover, the PyAtomDB output data are formatted as Python dictionaries and lists or FITS files, while PyXstar interfaces with Pandas dataframes that certainly uplifts data processing and analysis. Finally, PyAtomDB has a module with a wide variety of useful functions to manipulate atomic parameters to facilitate data processing.

## 7 Conclusions

Within the context of the atomic database of the spectral modeling code XSTAR, we have given an overview of current developments of a Python package for upscaling its data processing capabilities. These broadly involve the input, output, raw, and intermediate data with both plasma modeling and database curation in mind.

For performance, functions dealing with the raw and intermediate data have been coded as Python wrappers of the XSTAR Fortran subroutines. However, Cython versions have also been implemented for comparison and to look for alternatives to ensure long-term sustainability and maintenance of the system. The use of Pandas dataframes for data processing has been emphasized as well as the translation of the database from flat ASCII files to a relational model using the SQLite machinery. The ultimate intent is to allow the user to modify the atomic datasets in plasma modeling.

**Acknowledgements** We gratefully acknowledge private communications with Michael Witthoeft (NASA Goddard Space Flight Center and ADNET Systems Inc.) regarding his electron-configuration notation for atomic levels and with Manuel Bautista (US Department of Energy) for a report on the development of PyXstar_DB.

## Author contributions

All authors contributed to the conceptualization, development, and analysis of the content of the paper and revision of the manuscript.

**Funding** The material is based upon work supported by NASA under Award Number 80GSFC21M0002. CM acknowledges support from the Astrophysics Research and Analysis (APRA) Program under Grant 22-APRA-0101.

**Data Availability Statement** The manuscript has associated data in a data repository. [Author's comment: The xstar atomic database is available when the code is downloaded as a standalone package from the NASA High Energy Astrophysics Science Archive Research Center [HEASARC (https://heasarc.gsfc.nasa.gov/docs/software/lheasoft/xstar/xstar.html)].

**Code Availability Statement** Beta versions of the `PyXstar_model` and `PyXstar_grid` modules are available from the corresponding author upon request.



## Appendix A: The Witthoeft configuration notation

In the XSTAR database, electron configuration assignments for atomic levels abide by the Witthoeft notation. In this notation, a subshell is represented by the integer triplet $nlp$, where $n$ is the principal quantum number and $l$ the orbital angular momentum quantum number of the electron orbital, while $p$ denotes the subshell electron occupancy (number of electrons). The notation begins:

- with the last filled subshell if there are no previous fully empty subshells;
- otherwise, with the first fully empty subshell (zero occupancy) or the first partially filled subshell;
- no other fully empty subshells are included;
- subsequent filled and partially filled subshells are separated by dots;
- and ends with the level spectroscopic term $ML\_J$, where $M = 2S + 1$ is the total spin multiplicity, $L$ the total orbital angular momentum quantum number, and $J$ the total angular quantum number as an integer or half-integer value.

For instance, the ground level of hydrogen is assigned the configuration `1s1.2S_1/2`, while the first excited state is denoted by `1s0.2s1.2S_1/2`. The ground level of O III is assigned `2p2.3P_0`; the first odd-parity excited level `2s1.2p3.5S_2`, and the first K-vacancy level `1s1.2s2.2p3.5S_2`.

## References

1. T. Kallman, M. Bautista, Photoionization and high-density gas. Astrophys. J. Suppl. Ser. **133**(1), 221–253 (2001). https://doi.org/10.1086/319184
2. M.A. Bautista, T.R. Kallman, The XSTAR atomic database. Astrophys. J. Suppl. Ser. **134**(1), 139–149 (2001). https://doi.org/10.1086/320363
3. T.R. Kallman, P. Palmeri, M.A. Bautista, C. Mendoza, J.H. Krolik, Photoionization modeling and the K lines of iron. Astrophys. J. Suppl. Ser. **155**(2), 675–701 (2004). https://doi.org/10.1086/424039. arXiv:astro-ph/0405210 [astro-ph]
4. C. Mendoza, M.A. Bautista, J. Deprince, J.A. García, E. Gatuzz, T.W. Gorczyca, T.R. Kallman, P. Palmeri, P. Quinet, M.C. Witthoeft, The XSTAR atomic database. Atoms **9**(1), 12 (2021). https://doi.org/10.3390/atoms9010012. arXiv:2012.02041 [astro-ph.IM]
5. H.P. Summers, M.G. O'Mullane, A.D. Whiteford, N.R. Badnell, S.D. Loch, ADAS: atomic data, modelling and analysis for fusion, in *Atomic and Molecular Data and Their Applications, AIP Conference Series*, vol. 901, ed. by E. Roueff (2007), pp. 239–248. https://doi.org/10.1063/1.2727374
6. K.P. Dere, G. Del Zanna, P.R. Young, E. Landi, CHIANTI-an atomic database for emission lines. XVII. Version 10.1: revised ionization and recombination rates and other updates. Astrophys. J. Suppl. Ser. **268**(2), 52 (2023). https://doi.org/10.3847/1538-4365/acec79. arXiv:2305.15221 [physics.atom-ph]
7. A. Kramida, Y. Ralchenko, J. Reader, NIST ASD Team, NIST Atomic Spectra Database (ver. 5.11) [2024, February 14]. National Institute of Standards and Technology, Gaithersburg, MD (2023). https://physics.nist.gov/asd
8. T. Kallman, M. Bautista, J. Deprince, J.A. García, C. Mendoza, A. Ogorzalek, P. Palmeri, P. Quinet, Photoionization models for high-density gas. Astrophys. J. **908**(1), 94 (2021). https://doi.org/10.3847/1538-4357/abccd6. arXiv:2011.10603 [astro-ph.HE]
9. T. Hey, S. Tansley, K. Tolle, J. Gray, *The Fourth Paradigm: Data-Intensive Scientific Discovery* (Microsoft Research, Redmond, 2009)
10. C.R. Fierro-Santillán, J. Klapp, L.D.G. Sigalotti, J. Zsargó, M. Hareter, Analysis of spectral lines in large databases of synthetic spectra for massive stars. Astron. J. **161**(3), 121 (2021). https://doi.org/10.3847/1538-3881/abd950
11. G. Duniam, V.V. Kitaeff, A. Wicenec, Data modelling approaches to astronomical data: mapping large spectral line data cubes to dimensional data models. Astron. Comput. **38**, 100539 (2022). https://doi.org/10.1016/j.ascom.2021.100539
12. A.R. Foster, K. Heuer, PyAtomDB: extending the AtomDB atomic database to model new plasma processes and uncertainties. Atoms **8**(3), 49 (2020). https://doi.org/10.3390/atoms8030049
13. V. Luridiana, C. Morisset, R.A. Shaw, PyNeb: a new software for the analysis of emission lines, in *Planetary Nebulae: An Eye to the Future*, vol. 283 (2012), pp. 422–423. https://doi.org/10.1017/S1743921312011738
14. V. Luridiana, C. Morisset, R.A. Shaw, PyNeb: a new tool for analyzing emission lines. I. Code description and validation of results. Astron. Astrophys. **573**, 42

(2015). https://doi.org/10.1051/0004-6361/201323152. arXiv:1410.6662 [astro-ph.IM]
15. W. Cunto, C. Mendoza, The opacity project—the topbase atomic database. Rev. Mex. Astron. Astr. **23**, 107 (1992)
16. W. Cunto, C. Mendoza, F. Ochsenbein, C.J. Zeippen, TOPbase at the CDS. Astron. Astrophys. **275**, 5–8 (1993)
17. A. Danehkar, M.A. Nowak, J.C. Lee, R.K. Smith, MPI_XSTAR: MPI-based parallelization of the XSTAR photoionization program. PASP **130**(984), 024501 (2018). https://doi.org/10.1088/1538-3873/aa9dff. arXiv:1712.00343 [astro-ph.HE]
18. L. Juan de Dios, M. Rodríguez, Atomic data and the density structures of planetary nebulae. Mon. Not. R. Astron. Soc. **507**(4), 5331–5339 (2021). https://doi.org/10.1093/mnras/stab2488. arXiv:2108.13013 [astro-ph.GA]
19. C. Morisset, V. Luridiana, J. García-Rojas, V. Gómez-Llanos, M. Bautista, C. Mendoza, Atomic data assessment with PyNeb. Atoms **8**(4), 66 (2020). https://doi.org/10.3390/atoms8040066. arXiv:2009.10586 [astro-ph.GA]
20. C. Mendoza, J.E. Méndez-Delgado, M. Bautista, J. García-Rojas, C. Morisset, Atomic data assessment with PyNeb: radiative and electron impact excitation rates for [Fe II] and [Fe III]. Atoms **11**(4), 63 (2023). https://doi.org/10.3390/atoms11040063. arXiv:2304.01298 [astro-ph.GA]