The Astrophysical Journal, 947:65 (24pp), 2023 April 20 https://doi.org/10.3847/1538-4357/acbaff
© 2023. The Author(s). Published by the American Astronomical Society.
OPEN ACCESS

pynucastro: A Python Library for Nuclear Astrophysics

Alexander I. Smith[1], Eric T. Johnson[1], Zhi Chen[1], Kiran Eiden[2], Donald E. Willcox[3], Brendan Boyd[1], Lyra Cao[4], Christopher J. DeGrendele[5], and Michael Zingale[1]

[1] Department of Physics and Astronomy, Stony Brook University, Stony Brook, NY 11794-3800, USA; alexander.smithclark@stonybrook.edu
[2] Department of Astronomy, University of California, Berkeley, Berkeley, CA 94720-3411, USA
[3] Center for Computational Sciences and Engineering, Lawrence Berkeley National Lab, USA
[4] Department of Astronomy, Ohio State University, Columbus, OH 43210-1173, USA
[5] Department of Applied Mathematics, University of California, Santa Cruz, Santa Cruz, CA 95064-1077, USA

Received 2022 October 18; revised 2023 February 3; accepted 2023 February 8; published 2023 April 21

Abstract

We describe pynucastro 2.0, an open-source `library` for interactively creating and exploring astrophysical nuclear reaction networks. We demonstrate new methods for approximating rates and use detailed balance to create reverse rates, show how to build networks and determine whether they are appropriate for a particular science application, and discuss the changes made to the `library` over the past few years. Finally, we demonstrate the validity of the networks produced and share how we use pynucastro networks in simulation codes.

Unified Astronomy Thesaurus concepts: Nuclear astrophysics (1129); Nucleosynthesis (1131); Nuclear fusion (2324); Astrophysical processes (104); Reaction rates (2081); Astronomy software (1855); Computational astronomy (293)

1. Introduction

We present pynucastro 2.0, the latest version of our open-source Python `library` for nuclear astrophysics.[6] pynucastro began as a simple Python `library` that could interpret the reaction rate format used by REACLIB (Cyburt et al. 2010) and output the Python code to integrate a network linked by reaction rates from that `library`. The initial intent was for use in a classroom, to make it easier for students to work with, and interactively explore reaction networks. It quickly grew beyond that role, as we recognized the need to explore reaction rates, determine which reactions are important in different environments, and generate networks for multidimensional hydrodynamics simulation codes. Nuclear experiments are constantly refining our understanding of nuclei and reactions (Schatz et al. 2022), and we want to be able to use the most up-to-date reaction rates in our simulations without having to manually update the simulation codes. pynucastro meets this need by interfacing with compilations of nuclear reaction rates and nuclear properties generating the Python or C++ code needed to integrate a reaction network in a simulation code. pynucastro's object-oriented design makes it easy to expand its capabilities by building on existing rate or network classes. The development process is fully opened, managed on Github through pull requests and issues.

Since the the 1.0 release of pynucastro (Willcox & Zingale 2018), a lot of new features were added, including C++ code generation, the ability to modify and approximate rates, nuclear partition functions and the derivation of reverse rates via detailed balance, support for weak rate tables, nuclear statistical equilibrium state determination, electron screening support, many new plot types, and Numba acceleration of Python networks. Finally, pynucastro can generate the reaction networks needed for the AMReX-Astrophysics suite of simulation codes (Zingale et al. 2018) and leverage their ability to offload onto GPUs for accelerated computing. In this paper, we describe these new features, show examples of how we use the `library`, and demonstrate its validity.

The general evolution of a species involved in a two-body reaction rate, $A + B \rightarrow C + D$, takes the form

$$\frac{dn_A}{dt} = \frac{dn_B}{dt} = -(1 + \delta_{AB}) n_A n_B \frac{\langle \sigma v \rangle}{1 + \delta_{AB}}, \tag{1}$$

where n_A and n_B are the number densities of species A and B, respectively, and $\langle \sigma v \rangle$ is the reaction rate, expressed as an average of the product of the cross section, σ, and the relative velocity, v (assumed to follow a Maxwellian distribution), and the denominator corrects for double-counting if particles A and B are identical (see, e.g., Clayton 1968; Arnett 1996). The minus sign here signifies that species A and B are destroyed in this reaction, and the factor of $(1 + \delta_{AB})$ in the numerator indicates that if A and B are the same, then two of the nuclei are destroyed in the reaction. A corresponding equation for products C and D will have a plus sign:

$$\frac{dn_C}{dt} = \frac{dn_D}{dt} = +n_A n_B \frac{\langle \sigma v \rangle}{1 + \delta_{AB}}. \tag{2}$$

In terms of molar fraction, $Y_i \equiv n_i m_u / \rho$, we have, for example, for species A

$$\frac{dY_A}{dt} = -(1 + \delta_{AB}) \rho Y_A Y_B \frac{N_A \langle \sigma v \rangle}{1 + \delta_{AB}}, \tag{3}$$

where N_A is Avogadro's number. The form $N_A \langle \sigma v \rangle$ is the quantity that reaction rate tabulations usually provide (as a fit or a table in terms of temperature, T), for example, as in the classic compilation (Caughlan & Fowler 1988). There are other variations to this, including for three-body reactions and decays, but the general idea is the same—the change in species

[6] https://github.com/pynucastro/pynucastro/

Original content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

molar fraction is given by an ordinary differential equation (ODE) that depends on the molar fraction of reactants, the reaction rate, and density. Reaction networks usually work in terms of molar fractions, since it is easy to match the coefficients in the ODEs to the stoichiometric values from the reaction balance (see Hix & Meyer 2006 for a review of reaction rates).

A reaction network is the collection of nuclei and the rates that link them together, and it is expressed as a system of ODEs, $\{dY_k/dt\}$. Each term on the right-hand side of the ODE represents a link between the kth nucleus (with mass fraction Y_k) and another nucleus in the network. Astrophysical simulations use networks ranging from a few nuclei to several thousand, and managing and integrating the reaction network can be computationally challenging (see, e.g., Timmes 1999). Nuclear timescales can be short, and often the reaction network is integrated using implicit methods, requiring a Jacobian to solve the linearized system representing a single step. A network can evolve the species alone, at fixed temperature and density; include a temperature/internal energy equation under the assumption of constant density or pressure (a self-heating network, as in, e.g., Chamulak et al. 2008); evolve according to a prescribed thermodynamic trajectory (as is often done with tracer particles; e.g., Seitenzahl et al. 2010); be used in an operator split fashion with a hydrodynamics code, with or without an energy equation integrated together with the species (e.g., Müller 1986); or be fully coupled to the hydrodynamics equations and include the velocity-dependent advective terms in addition to energy evolution (as, e.g., done in Zingale et al. 2022).

pynucastro is designed to aid in the creation and management of reaction networks. It knows how to compute the reaction rates, $N_A\langle\sigma v\rangle$, and assemble the terms in Equation (3) that describe a reaction's contribution to the evolution of a nucleus. The goal is to produce a network that can be used in a simulation with just a few lines of Python, with the `library` pulling in the up-to-date reaction rates, finding all of the links, and writing out the code that represents the right-hand side of the system of ODEs. Additionally, since we usually want to use a minimally sized network for multidimensional simulations, pynucastro understands rate approximations and has methods for determining whether specific rates are important for a particular application.

There are already a number of freely available nuclear reaction networks, including the set of approximate (`aprox13` and `aprox19`) and general (`torch`) networks (Timmes 1999), BRUSLIB/NETGEN (Aikawa et al. 2005; Xu et al. 2013), and SkyNet (Lippuner & Roberts 2017). pynucastro has different goals than these, and as such, it provides complementary benefits to the community. pynucastro encourages interactive exploration of rates and networks in Jupyter notebooks and is designed to output fixed-sized reaction networks (potentially with approximate rates) that can be used in simulation codes. Being written completely in Python, it has a low barrier of entry for new contributions and encourages the addition of new features through an open development model. By writing out the code explicitly for a particular network, and not for a general net that can be configured at runtime, we are able to produce networks that can run efficiently on GPUs as part of large simulation codes (Katz et al. 2020). Additionally, when coupling to hydrodynamics, we need to integrate the network with an energy equation and possibly advective terms, which requires more flexibility in how the integration is done than other libraries permit.

This paper is organized as follows. In Section 2 we describe the overall design of the `library`. In Section 3 we walk through the process of creating a reaction network and integrating it. We describe the ability to approximate rates in Section 4, discuss tabular rates in Section 5, and show how to compute inverse rates via detailed balance in Section 6. In Sections 7 and 8 we describe screening and establishing nuclear statistical equilibrium, respectively. Finally, in Section 9 we compare a Python and a C++ generated network. A set of Jupyter notebooks is provided as supplementary material archived on Zenodo[7] that reproduces all the figures shown in the paper. We will refer to individual notebooks in the figure captions.

2. Library Structure

pynucastro uses an extensible, object-oriented design and is written completely in Python. It leverages Numba just-in-time compilation for acceleration of performance-critical parts, SymPy for C++ code generation, SciPy for integrating Python networks, and NetworkX and matplotlib for plotting. We use pytest for unit testing and have extensive documentation prepared with Sphinx and Jupyter notebooks. pynucastro is available on PyPI and can easily be installed via `pip`.

The `library` is built from a few core classes that represent nuclei, rates, and other concepts. A user works directly with objects of these classes to build a network and explore it. The core classes in pynucastro are as follows:

1. `Nucleus`: a `Nucleus` represents a single nuclear isotope and stores the nuclear properties, including proton number, atomic weight, and binding energy. It also holds `SpinNuclide` and `PartitionFunction` objects that can express the nuclear spin and partition function, respectively. Finally, a `Nucleus` knows how to print the nucleus to a screen / Jupyter notebook in both plain text and MathJax formatting.
2. `Composition`: a `Composition` object stores the mass fractions of a collection of nuclei, with methods that enable us to initialize them in various fashions (e.g., solar) and convert them to molar fractions. A `Composition` object is needed when evaluating a reaction rate in Python.
3. `Rate`: a `Rate` is a base class that implements functions common to all rate types. Two main classes are derived from this: `ReacLibRate` for rates from the REACLIB `library` and `TabularRate` for rates that are specified as a table of $(\rho Y_e, T)$. These classes can evaluate the rate in Python and output the source code needed to construct the rate in both C++ and Python.

   Several other classes described below are constructed from these, including `ApproximateRate`, which groups rates together into a single effective rate approximation, and `DerivedRate`, which recomputes an inverse rate from a forward rate using detailed balance.
4. `Library`: a `Library` is a collection of `Rate` objects, which can be from a particular source (e.g., the REACLIB database can be read in as a `Library`). A `Library` provides many methods for filtering the rates, allowing one to select based on the nuclei involved.

[7] https://doi.org/10.5281/zenodo.7202413

A reaction network is built by selecting one or more rates from a `Library`.

5. `RateCollection`: a `RateCollection` is the base class for representing a network—a set of `Nucleus` objects and the list of `Rate` objects that link them together. A `RateCollection` has a number of methods that help manage a network. For example, it is at the `RateCollection` level that one can approximate the network or recompute reverse rates via detailed balance. `RateCollection` also has many ways to visualize a network, including interactively with Jupyter notebook widgets. The `PythonNetwork` and `AmrexAstroCxxNetwork` classes build off of `RateCollection` to produce the source code for a Python or C++ network in the AMReX-Astrophysics suite framework.

2.1. Data Sources

pynucastro uses the REACLIB rate `library` (Cyburt et al. 2010) for most rates. REACLIB aims to be a single source for thermonuclear reaction rates, aggregating different community contributions into a single, uniform format. This provides fits to the $N_A\langle\sigma v\rangle$ portion of the rate in terms of a standard function of temperature. REACLIB does not provide rates for density-dependent electron capture reactions. For these, we currently include the tables of Suzuki et al. (2016), and other sources can easily be added.

A REACLIB `Rate` is described in terms of sets, each of which has seven coefficients, making up a fit of the form

$$N_A\langle\sigma v\rangle = \exp\left(a_0 + \sum_{i=1}^{5} a_i T_9^{(2i-5)/3} + a_6 \log T_9\right) \quad (4)$$

with $T_9 = T/(10^9\ \mathrm{K})$. A single rate can consist of multiple sets, representing, for instance, resonances and the smooth part of the rate, and these are added together to give the total rate. A `ReacLibRate` stores a Python list of `SingleSet` objects, each representing a set that contributes to the total rate. A `SingleSet` knows how to numerically evaluate the set and output the Python and C++ code to compute the set as part of a reaction network. The `ReacLibLibrary` class reads in the latest stored version of the entire REACLIB rate `library`, and `Library` provides the starting point for filtering out the rates of interest.

Electron capture and beta-decay rates are taken from Suzuki et al. (2016) and stored as a two-dimensional table (electron density and temperature) as part of the `TabularRate` class. Linear interpolation is used to return the needed data at an arbitrary thermodynamic point. The `TabularLibrary` class reads in all of the tabular weak rates known to pynucastro as a `library`. It is important to note that there can be overlap with tabular weak rates and approximate weak rates in REACLIB. Usually in these cases, one should use the tabular version of the rate (since it captures the density dependence).

Several nuclear properties are needed to derive inverse rates. In our implementation we use the ground-state spin and the mass excesses from Huang et al. (2021) and Wang et al. (2021) under strong experimental arguments. Each rate's Q-value is obtained from REACLIB; however, we have implemented an alternative option to compute it directly from the measured mass excesses. The spin measurements may be different from the NuDat database used in Cyburt et al. (2010); therefore, we may expect significant differences in heavy or neutron-poor/rich nuclei, where systematic measurements dominate. The partition function values are tabulated in Rauscher (2003) and Rauscher et al. (1997) and merged in the case where each nucleus is present in both tables. We linearly interpolate the logarithm of the partition function from the table data.

2.2. Development Model

pynucastro is openly developed on github, with all the changes done via issues and pull requests. Issues includes new feature requests, bug reports, and questions. Pull requests require a review from a developer, along with all tests and workflows passing before they can be merged. Currently, there are over 400 issues + pull requests filed. Developers who have contributed regularly in the past are granted permission to approve and merge pull requests. Contributors who have made significant contributions were invited to be coauthors of this paper and also appear in the Zenodo record created for each new `library` release. Code checkers (pylint and flake8) are run on every pull request, as well as a suite of unit tests (managed through pytest); a test compilation and run of a C++ network with the AMReX-Astro Microphysics framework, comparing to a previously stored answer; and a build of the docs to ensure that changes do not break existing notebook examples. This is all managed via github actions. The unit tests currently cover 70% of the code, with plotting and Numba-wrapped routines as the main exceptions.

3. Designing a Network in Python

3.1. Selecting Nuclei and Rates

Creating an astrophysical nuclear reaction network is an art form that requires understanding which nuclei are important for your application and which rates need to be included to accurately capture the energy generation rate. In pynucastro this translates into assembling the rates that are important into one or more `Library` objects and then creating a `RateCollection` (or one of its derived classes). We often begin by using a network that is described elsewhere in the literature as inspiration. As a result of the complexity of ensuring that a network accurately captures all of the processes important to your application, a few networks (like the venerable `aprox13`) dominate most of the multidimensional simulation work done today. pynucastro has tools to help understand what rates and nuclei may be important.

There are a number of ways to select rates for your network:

1. pass in a list of just the individual rate files you want to use (e.g., downloaded from the REACLIB website);
2. read in a large `library` (like the entire REACLIB database) and filter the rates based on which nuclei are involved.

This will result in a `Library` containing just the rates you want to use. But now there are two potential worries:

1. Some key nuclei or rates might be missing.
2. The network may be larger than need be if it includes unimportant nuclei and/or rates.

A `RateCollection` provides methods to help with both of these issues.

The easiest way to assess a network is to simply plot it and look to make sure that all of the connections you expect are present. The `validate()` function also helps here—it takes a larger `Library` as an argument and checks the rates in the

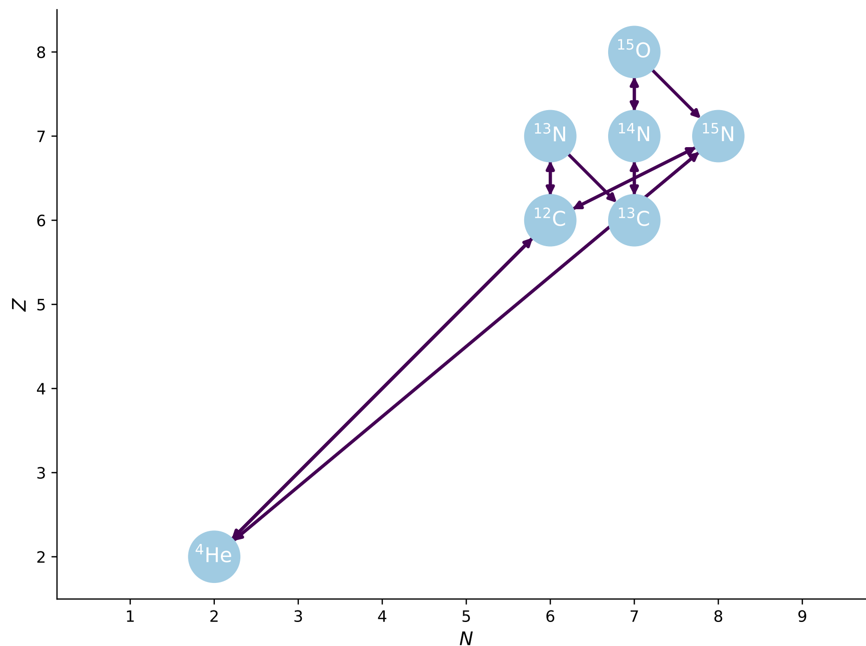


Figure 1. A simple network made by finding all rates that connect the input nuclei. (supplemental notebook: `cno-network.ipynb`)

`RateCollection` against all of those present in the comparison `library`, and report if any products are not consumed by other rates (signifying an endpoint in the network) or if you are missing any other rates with the exact same reactants (because, for instance, you included protons but not neutrons in your initial rate filtering). An example of the latter is that you might have ^{12}C(^{12}C, p)^{23}Na but not ^{12}C(^{12}C, n)^{23}Mg, because you did not include neutrons and ^{23}Mg in the initial list of nuclei you wanted to consider.

The `find_unimportant_rates()` method can help trim down a network by computing the relative magnitude of each rate in a network. Given a list of thermodynamic states (as a tuple: `(density, temperature, Composition)`) and a threshold, ϵ, it returns a list of all the rates that are always smaller than $\epsilon \times |\text{fastest rate}|$. This can then be used to automatically remove rates.

To illustrate the process of creating a network, we will show how to make a simple CNO network and discuss the above considerations. In the examples that follow we import the pynucastro `library` as:

```
import pynucastro as pyna
```

We will start with the REACLIB `library`. The class `ReacLibLibrary` (derived from `Library`) reads the entire `library` (currently >80,000 rates).

```
rl = pyna.ReacLibLibrary()
```

There are a number of ways to filter rates from the `library`. For instance, if we want just the ^{12}C(p, γ)^{13}N rate, then we can do:

```
c12pg = rl.find_rate_by_name("c12(p,g)n13")
```

We can then interact with this `Rate` object directly, for example, to plot it or evaluate it at some temperature.

We will take a different approach for our CNO network. We can start with a list of nuclei we are interested in and then use `linking_nuclei()` to select only rates that involve these input nuclei.

```
lib = rl.linking_nuclei(["p", "he4",
                         "c12", "c13",
                         "n13", "n14", "n15",
                         "o15"])
pynet = pyna.PythonNetwork(libraries=[lib])
```

Optional arguments to `linking_nuclei` allow one to consider just the reactants or products. More sophisticated filtering can also be done. Figure 1 shows a plot of the network created in this manner (created simply via `pynet.plot()`). The arrows indicate the direction of the rate, with double-ended arrows meaning that both the forward and reverse rates are present (the two can be optionally plotted as arcs so they do not overlap). We see that this network has more rates than we really needed if we just want to consider H burning via CNO. In conditions where CNO is important, the triple-alpha rate probably is not. In addition, it is unlikely that we will need the reverse rates for the CNO sequence (p + p is probably important, but we omit that here to simplify the example).

To simplify this network, we evaluate the rates at a thermodynamic state slightly hotter than the Sun's core temperature and remove the small rates. We need to build a `Composition` object that stores the mass fractions for this network—we will assume a solar-like distribution. First, we

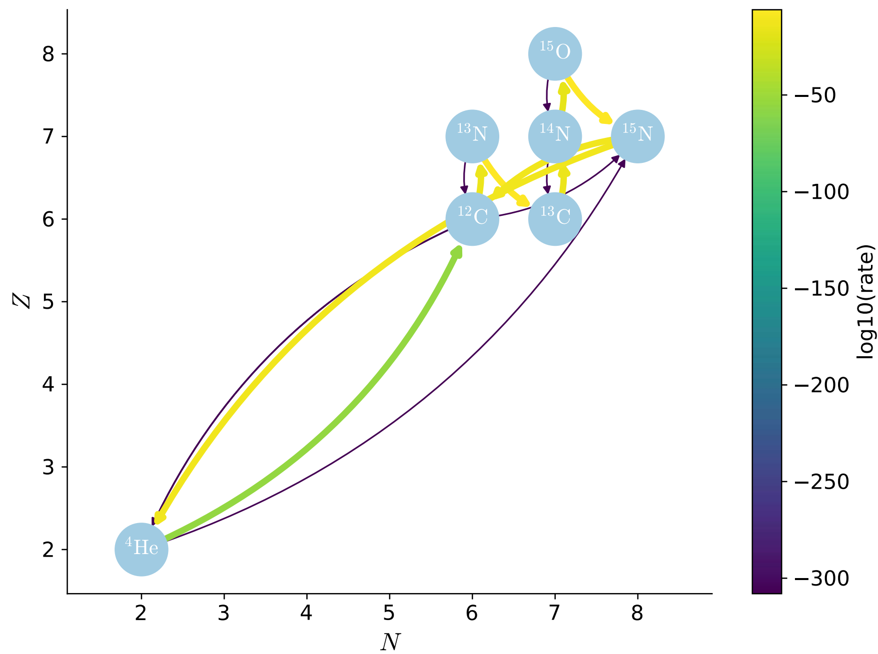


Figure 2. The CNO network with the links colored by reaction rate strength. (supplemental notebook: `cno-network.ipynb`)

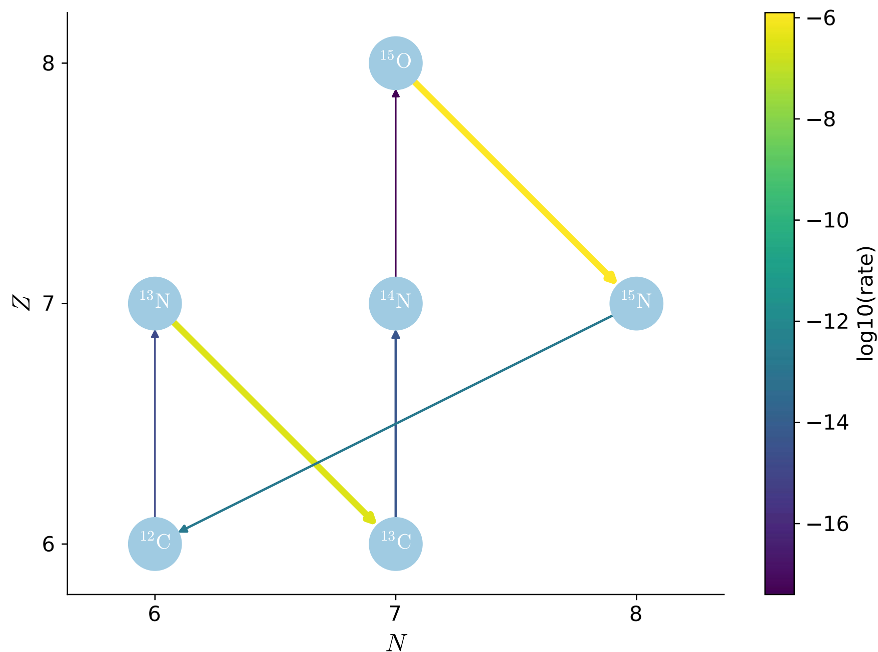


Figure 3. Filtering the `RateCollection` from Figure 1 by removing unimportant rates. (supplemental notebook: `cno-network.ipynb`)

can look at a plot with the links colored by the reaction rate strength for this thermodynamic state—this is shown in Figure 2. As we suspected, the reverse rates are all small, as is the triple-α rate, compared to the main CNO rates. In fact, on this scale, the rates span 300 orders of magnitude.

We can now remove the slow rates. We will use an automated method and filter the rates that are less than 10^{-20} compared to the fastest rate. We do not worry about screening under these conditions, but this check can be done for screened rates as well.

```
comp = pyna.Composition(pynet.get_nuclei())
comp.set_solar_like()
density = 150          # gram/(cm^3)
temperature = 2.e7     # Kelvin
state = (density, temperature, comp)
srates = pynet.find_unimportant_rates([state],
                                       1.e-20)
pynet.remove_rates(srates)
```

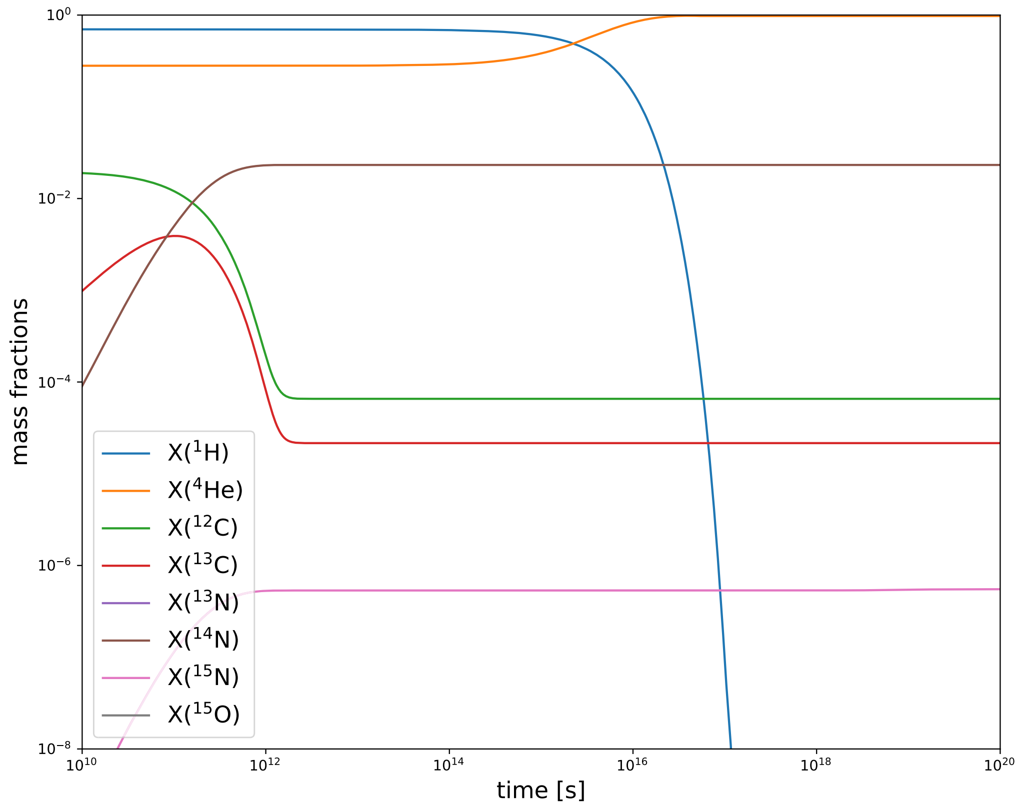


Figure 4. Example integration of the CNO network. (supplemental notebook: `cno-network.ipynb`)

Figure 3 shows the result. We note that ^{4}He and p are not shown by default, unless p + p or triple-α are present in the network. We see that now the range of rates is much more reasonable. The reverse rates are gone, as is the triple-α rate, and we see that the CNO cycle is all that remains. It is also clear from the plot that the ^{14}N(p, γ)^{15}O rate is the limiting rate for the cycle.

This was a simple example, and we could have just specified the exact rates that make up the CNO cycle from the start. But for larger networks and applications to new problem domains, we might not know what rates are needed. Furthermore, `find_unimportant_rates()` can take a list of thermodynamic states, for instance, sampled from a real simulation at different times and positions in the domain, and remove only the rates that are never important under any of those conditions.

3.2. Integrating

Once we are happy with the network, we can write out the Python code that implements all of the functions needed to integrate the network via `pynet.write_network()`. In particular, this will define a module that contains a function `rhs()` that fills the right-hand side of the vector of ODEs that describe the network ($\dot{Y}_k$), a function `jac()` that creates a Jacobian array ($J_{km} = d\dot{Y}_k/dY_m$), and the nuclear properties like proton number, Z_k, and atomic mass, A_k. Reaction networks are often stiff, requiring implicit methods, and hence a Jacobian. Each rate also knows how to write the function that computes its temperature-dependent $N_A\langle\sigma v\rangle$.

The following code shows how to write the network and integrate it using SciPy `solve_ivp()`. We create and initialize an array of mass fractions and then convert to molar fractions, using the nuclear properties in the network module. This becomes the initial conditions for the integration:

```
pynet.write_network("cno.py")
import cno
rho = 150    # gram/(cm^3)
T = 2.e7     # Kelvins

X0 = np.zeros(cno.nnuc)
X0[cno.jp] = 0.7       # unitless mass fraction
X0[cno.jhe4] = 0.28    # unitless mass fraction
X0[cno.jc12] = 0.02    # unitless mass fraction

Y0 = X0/cno.A

tmax = 1.e20    # seconds

sol = solve_ivp(cno.rhs, [0, tmax], Y0,
                method="BDF",
                jac=cno.jacobian,
                dense_output=True,
                args=(rho, T),
                rtol=1.e-6, atol=1.e-8)
```

We use the BDF integration method, since that works well with stiff ODEs and we explicitly pass in the Jacobian function. The density and temperature are passed into the function via the `args` keyword argument—if we wished, we could easily make these time dependent, computing them on the fly from the current simulation time. Finally, we specify both a relative tolerance and an absolute tolerance, which are combined into a single error of the form $\epsilon_i = \mathtt{rtol}|X_i| + \mathtt{atol}$. Figure 4 shows the evolution of the nuclei. For these conditions—slightly hotter than the central temperature of the Sun—we see that H is depleted in about 10^{17} s.

4. Approximate Rates

Reaction networks with a lot of nuclei can be computationally expensive—both because of the size of the linear system that needs to be solved during the implicit integration and when used in a hydrodynamics code owing to the memory needed to carry and advect each species separately. As a result, networks are often approximated—removing rates and nuclei deemed not important and replacing a collection of rates with an effective rate.

One of the most common rate approximations is to group the sequence $A(\alpha, \mathrm{p})X(\mathrm{p}, \gamma)B$ and $A(\alpha, \gamma)B$ into a single effective rate. This is done by assuming that nucleus X and protons are in equilibrium. The evolution equations for these two reaction sequences, including forward and reverse rates, are given as

$$\frac{dY_A}{dt} = \frac{dY_\alpha}{dt} = -\rho Y_A Y_\alpha \lambda_{\alpha,\gamma} + Y_B \lambda_{\gamma,\alpha} - \rho Y_A Y_\alpha \lambda_{\alpha,\mathrm{p}} + \rho Y_X Y_\mathrm{p} \lambda_{\mathrm{p},\alpha} \tag{5a}$$

$$\frac{dY_B}{dt} = +\rho Y_A Y_\alpha \lambda_{\alpha,\gamma} - Y_B \lambda_{\gamma,\alpha} + \rho Y_X Y_\mathrm{p} \lambda_{\mathrm{p},\gamma} - Y_B \lambda_{\gamma,\mathrm{p}} \tag{5b}$$

$$\frac{dY_X}{dt} = \frac{dY_\mathrm{p}}{dt} = +\rho Y_A Y_\alpha \lambda_{\alpha,\mathrm{p}} - \rho Y_X Y_\mathrm{p} \lambda_{\mathrm{p},\gamma} + Y_B \lambda_{\gamma,\mathrm{p}} - \rho Y_X Y_\mathrm{p} \lambda_{\mathrm{p},\alpha}. \tag{5c}$$

Here we use the shorthand notation for the rates, with the forward rates

$$\lambda_{\alpha,\gamma} = N_A \langle \sigma v \rangle_{A(\alpha,\gamma)B} \tag{6a}$$

$$\lambda_{\alpha,\mathrm{p}} = N_A \langle \sigma v \rangle_{A(\alpha,\mathrm{p})X} \tag{6b}$$

$$\lambda_{\mathrm{p},\gamma} = N_A \langle \sigma v \rangle_{X(\mathrm{p},\gamma)B} \tag{6c}$$

and reverse rates

$$\lambda_{\gamma,\alpha} = N_A \langle \sigma v \rangle_{B(\gamma,\alpha)A} \tag{7a}$$

$$\lambda_{\gamma,\mathrm{p}} = N_A \langle \sigma v \rangle_{B(\gamma,\mathrm{p})X} \tag{7b}$$

$$\lambda_{\mathrm{p},\alpha} = N_A \langle \sigma v \rangle_{X(\mathrm{p},\alpha)A}. \tag{7c}$$

Finally, notice that the evolution equations for Y_X and Y_p are identical and the evolution equations for Y_A and Y_α are identical.

If we assume an equilibrium in the proton flow, $dY_p/dt = 0$, then we can solve for the product $\rho Y_X Y_p$:

$$\rho Y_X Y_p = \frac{\rho Y_A Y_\alpha \lambda_{\alpha,p} + Y_B \lambda_{\gamma,p}}{\lambda_{p,\gamma} + \lambda_{p,\alpha}}. \tag{8}$$

Substituting this into the dY_A/dt expression and grouping the forward and reverse terms together, we have

$$\frac{dY_A}{dt} = -\rho Y_A Y_\alpha \left[\lambda_{\alpha,\gamma} + \lambda_{\alpha,p}\left(1 - \frac{\lambda_{p,\alpha}}{\lambda_{p,\gamma} + \lambda_{p,\alpha}}\right)\right] + Y_B \left[\lambda_{\gamma,\alpha} + \frac{\lambda_{p_\alpha} \lambda_{\gamma,p}}{\lambda_{p,\gamma} + \lambda_{p,\alpha}}\right]. \tag{9}$$

This allows us to identify the effective forward and reverse rates:

$$\lambda'_{\alpha,\gamma} = \lambda_{\alpha,\gamma} + \frac{\lambda_{\alpha,p} \lambda_{p,\gamma}}{\lambda_{p,\gamma} + \lambda_{p,\alpha}} \tag{10a}$$

$$\lambda'_{\gamma,\alpha} = \lambda_{\gamma,\alpha} + \frac{\lambda_{p,\alpha} \lambda_{\gamma,p}}{\lambda_{p,\gamma} + \lambda_{p,\alpha}}, \tag{10b}$$

and the evolution equations reduce to

$$\frac{dY_A}{dt} = -\rho Y_A Y_\alpha \lambda'_{\alpha,\gamma} + Y_B \lambda'_{\gamma,\alpha} \tag{11a}$$

$$\frac{dY_B}{dt} = +\rho Y_A Y_\alpha \lambda'_{\alpha,\gamma} - Y_B \lambda'_{\gamma,\alpha}. \tag{11b}$$

This approximation is the basis of the popular "aprox" networks (Timmes 1999) and traces its origins at least as far back as Weaver et al. (1978). We note that as an effect of this approximation, the protons produced by (α, p) are assumed to only react with the nucleus X and not with any other nuclei in the network. pynucastro can do this rate approximation automatically.

Consider a simple network that links ^{24}Mg, ^{28}Si, and ^{32}S via $(\alpha, p)(p, \gamma)$ and (α, γ). Fully connecting these nuclei requires that we include ^{27}Al and ^{31}P as the intermediate nuclei from the (α, p) reactions. For the approximate version, we would like to remove those two nuclei from the network but approximate the flow through them using the effective rates derived above. The code to make this approximation is:

```
rl = pyna.ReacLibLibrary()
lib = rl.linking_nuclei(["mg24", "al27",
                         "si28", "p31", "s32",
                         "he4", "p"])
pynet = pyna.PythonNetwork(libraries=[lib])
pynet.make_ap_pg_approx()
pynet.remove_nuclei(["al27", "p31"])
```

The method `make_ap_pg_approx()` looks to see whether all of the necessary (α, γ), (α, p), and (p, γ) rates (and their inverses) exist in the `RateCollection` and, if so, creates an `ApproximateRate` object that replaces the rates with the effective approximation (while storing the original rates internally to evaluate them for computing the approximation). We then explicitly remove the intermediate nuclei from the network. The original and approximate network is visualized in Figure 5. The gray links in the approximate version indicate that those rates are still computed and used as part of the approximation. We note one other feature of this approximation: there was a link between ^{27}Al and ^{31}P, but since we removed these nuclei, that link is also removed.

To see how well this approximation works, we can integrate it along with the full network (no approximate rates) and a reduced network with no (α, p) links at all (approximate or explicit). We pick $\rho = 10^7$ g cm^{-3}, $T = 3 \times 10^9$ K, and the initial composition $X(^4\mathrm{He}) = X(^{24}\mathrm{Mg}) = 0.5$. The temperature and density are kept fixed. Figure 6 shows the comparison. We see that the network without any $(\alpha, p)(p, \gamma)$ links (dashed lines) takes the longest to burn the ^{24}Mg and the trajectories of all of the isotopes are very different from the full network (solid lines). In contrast, the approximate network (dotted lines) captures the timescale for ^{24}Mg depletion and ^{32}S creation well, agreeing with the full network after a brief transient. There is also good agreement in the intermediate ^{28}Si abundance, with a departure only at very low abundances, as it is almost depleted. This shows that the approximate rate infrastructure in pynucastro works as intended.

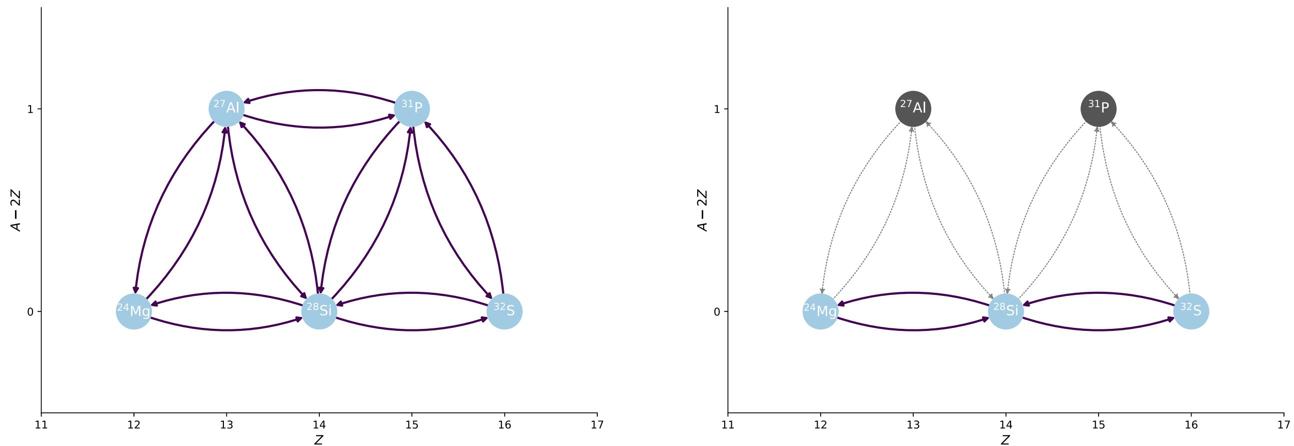


Figure 5. The full ^{24}Mg-burning network (left), and the approximate version (right). Rates that are approximated are shown as dotted lines, and nuclei that are not part of the actual network are shown in gray. (supplemental notebook: `approximate-rates.ipynb`)

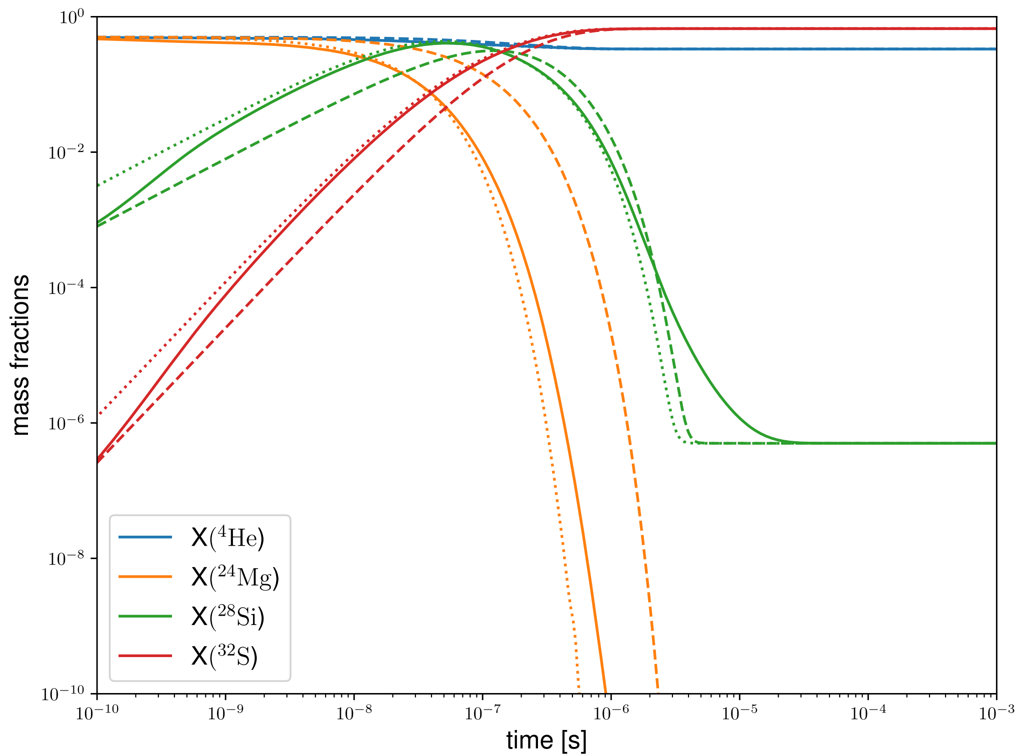


Figure 6. Comparison of a full network with p, ^{4}He, ^{24}Mg, ^{27}Al, ^{28}Si, ^{31}P, ^{32}S (solid), an approximate version that removes ^{27}Al and ^{31}P, by combining the $(\alpha, p)(p, \gamma)$ and (α, γ) rates (dotted) and a reduced network that has no links (explicit or approximate) to ^{27}Al and ^{31}P (dashed). Only the nuclei in common to all three networks are shown. We see that the approximate network follows the full network closely, but the reduced network takes longer to burn the initial ^{24}Mg because it is missing the additional pathway through ^{27}Al. (supplemental notebook: `approximate-rates.ipynb`)

Approximate networks can be exported both to Python and to C++. In this case, pynucastro writes the source code of the functions needed to evaluate each of the rates, as well as the code, needed to arrange them together in the approximation, to a file. While this is one of the most commonly used approximations in nuclear reaction networks, it is straightforward to create other approximations using the same framework in pynucastro.

Another easy approximation that pynucastro supports is modifying a rate. In this case, the rate itself is not changed, but the endpoint nuclei are, and the Q-value for the rate is recomputed. An example use case for this is if you want to capture carbon burning in a network. REACLIB has three rates that involve ^{12}C + ^{12}C, but if you created your network without neutrons, then you will not have pulled in ^{12}C(^{12}C, n)^{23}Mg(n, γ)^{24}Mg. The neutron capture here tends to be very fast, so we could simply approximate this rate as ^{12}C(^{12}C, γ)^{24}Mg. This can be done using the `modify_products()` method on the rate. The end result is that we approximate this branch of carbon burning without having to pull two more nuclei into our

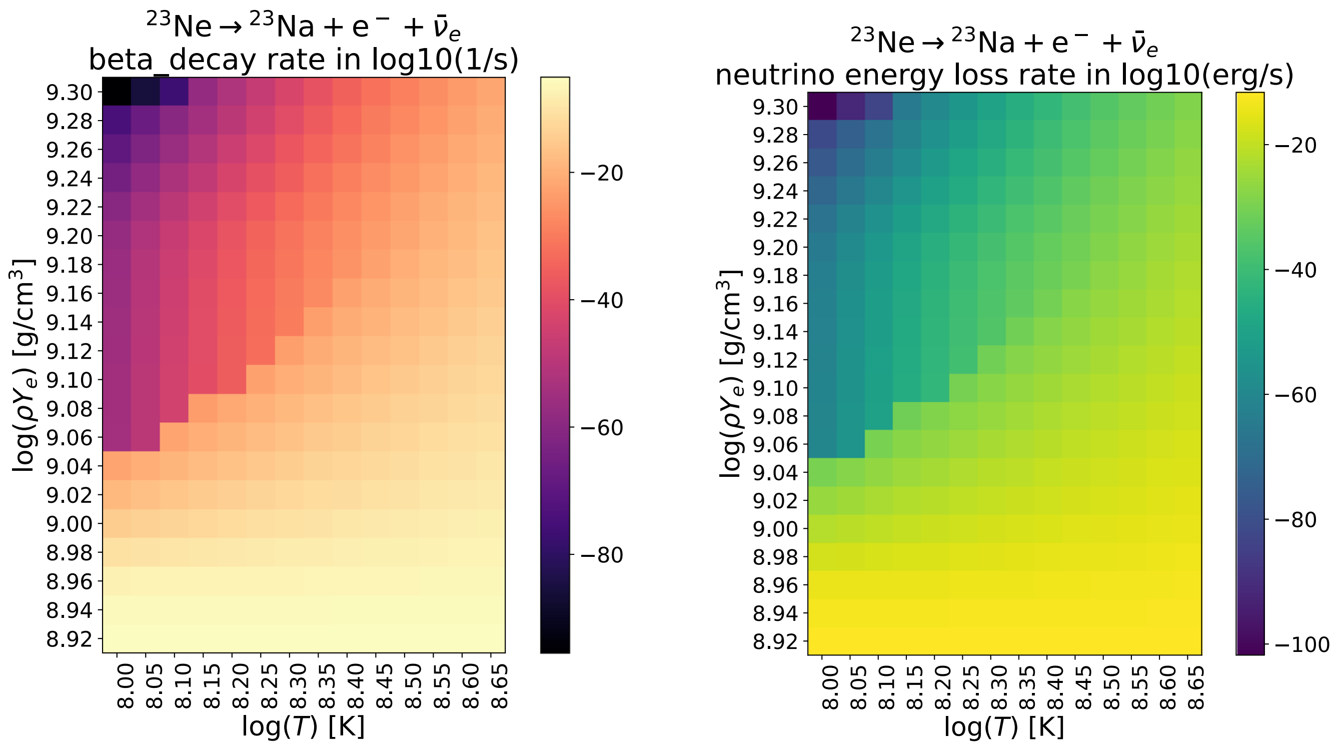


Figure 7. The left panel shows the reaction rate for the ^{23}Ne beta decay for a range of electron density and temperature. The right panel shows the energy-loss rate due to neutrinos for the ^{23}Ne beta decay. (supplemental notebook: `urca_network.ipynb`)

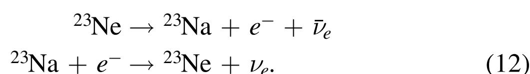
network (n and ^{23}Mg). This was used in the network described in Zingale et al. (2022) and is illustrated in the network described in Appendix E.

5. Tabular Weak Rates

Weak rates are of great interest to many applications, especially in stellar environments. These rates often depend on both temperature and density, requiring the rates to be tabulated prior to integration of a network. pynucastro natively includes tabular rates for sd-shell nuclei pairs with A of 17–28 from Suzuki et al. (2016). These rates are accessed through the `TabularRate` class and can be included in networks with the more standard REACLIB rates.

`TabularRate` reads a file given in a (ρY_e, T) format and performs linear interpolation to calculate the appropriate rate and the energy lost to neutrinos. The energy loss presupposes that the neutrinos free stream out of the environment. Figure 7 demonstrates this capability over a range of electron density and temperature for the ^{23}Ne beta-decay reaction. These two plots are both easily made with the `TabularRate` class and allow for exploration of the various weak rates in pynucastro.

A nice example of a full network that incorporates `TabularRate` is the $A = 23$ Urca reactions with simple carbon burning. The $A = 23$ Urca reactions consist of a beta decay and electron capture, linking the isotopes ^{23}Na and ^{23}Ne:



$$^{23}\mathrm{Ne} \rightarrow {}^{23}\mathrm{Na} + e^- + \bar{\nu}_e$$
$$^{23}\mathrm{Na} + e^- \rightarrow {}^{23}\mathrm{Ne} + \nu_e. \quad (12)$$

Both weak reactions were tabulated rates by Suzuki et al. (2016). Rates related to carbon burning are from REACLIB. Here we see how to create this network with pynucastro. First, we search the `TabularLibrary` for rates linking ^{23}Na and ^{23}Ne. Then, we specify rates from the `ReacLibLibrary` by using each rate's name:

```
rl = pyna.ReacLibLibrary()
tl = pyna.TabularLibrary()

# get Tabular rates
urca23_nuc = ["na23", "ne23"]
urca23_lib = tl.linking_nuclei(urca23_nuc)
urca23_rates = urca23_lib.get_rates()

# get ReacLib rates
rl_names = ["c12(a,g)o16",
            "c12(c12,a)ne20",
            "c12(c12,p)na23",
            "c12(c12,n)mg23",
            "n(,e)p",]
rl_rates = rl.get_rate_by_name(rl_names)

#combine rates in one network
rates = urca23_rates + rl_rates
urca23_net = pyna.PythonNetwork(rates=rates)
```

The resulting network is shown in Figure 8. This demonstrates how we can easily incorporate both `TabularRate` and `ReacLibRate` rates into a single network. For C++ networks, `AmrexAstroCxxNetwork` automatically adds the calls to interpolate from the rate tables when filling the reaction rates. This network was used for convective Urca simulations (Calder et al. 2019) with the MAESTROeX code (Fan et al. 2019).

6. Reverse Rates and Nuclear Partition Functions

The relationship between the inverse and the forward reaction may be obtained by detailed balance calculations—

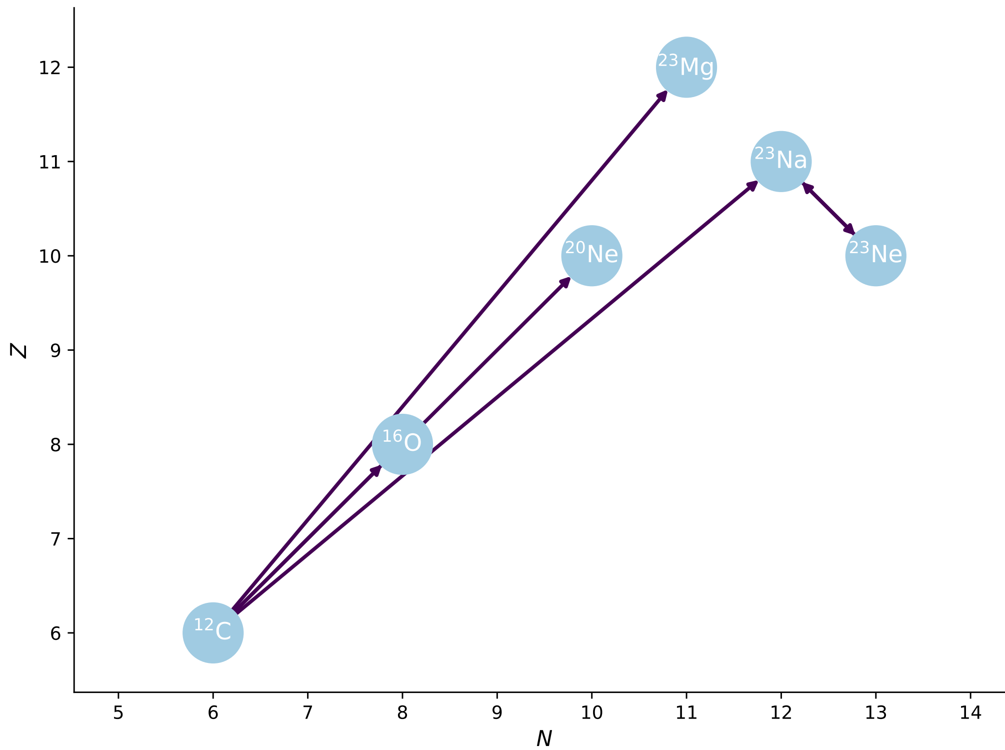


Figure 8. A network with simple carbon burning and the $A = 23$ Urca reactions. (supplemental notebook: `urca_network.ipynb`)

under the assumption of equilibrium in the reaction, we can derive the inverse reaction rate $N_a^{p-1}\langle\sigma v\rangle_{B_1,B_2,\cdots,B_p}$ in terms of the forward rate $N_a^{r-1}\langle\sigma v\rangle_{C_1,C_2,\cdots,C_r}$ (see the derivation in Appendix B). Each detailed balance inverse rate can be decomposed as a multiplication between a component that satisfies fit (4), as in REACLIB, and an additional component that involves partition functions:

$$N_a^{p-1}\langle\sigma v\rangle_{B_1,B_2\cdots,B_p} = N_a^{p-1}\langle\sigma v\rangle'_{C_1,C_2\cdots,C_p} \times \frac{G_{C_1}\cdots G_{C_r}(T)}{G_{B_1}\cdots G_{B_p}(T)}, \tag{13}$$

where $N_a^{p-1}\langle\sigma v\rangle_{B_1,B_2\cdots,B_p}$ is defined by Equation (B9) and $N_a^{p-1}\langle\sigma v\rangle'_{B_1,B_2\cdots,B_p}$ is the rate parameterization (4) with the constants $a_{0,\mathrm{rev}}$, $a_{1,\mathrm{rev}}$, $\cdots$, $a_{6,\mathrm{rev}}$ given by Equations (B10a–B10g), and each G_{n_i} is the nuclear partition function of the n_i-nuclei defined in Equation (A8).

The nuclear partition function information tables are stored in a suitable format to be read by pynucastro, switching between merging the low- and high-temperature tables in Rauscher et al. (1997) and Rauscher (2003), or to use the low-temperature table only, in Rauscher et al. (1997). Furthermore, we can decide between using the finite-range droplet model (FDRM) and using the Thomas-Fermi approach with Strutinski integral (ETFSI-Q) mass models to compute the partition function values. In our implementation of the partition functions in pynucastro, we have considered merging both temperature tables and the FDRM mass model as our default options.

Similarly, the tabulated spin of the ground-state nuclei from Huang et al. (2021) and Wang et al. (2021) is also stored in a suitable format to be read by pynucastro. From this point, we can choose between using all the unique-value ground-state spin measurements, regardless of the experimental or theoretical arguments, or to keep only the strong experimental observed measurements. In our implementation for the spin of the ground-state nuclei in pynucastro, we have considered the latter choice.

In order to compute the inverse rate by detailed balance (13), we have implemented the `DerivedRate` class. The `DerivedRate` class inherits from `Rate` and computes the REACLIB component that satisfies fit (4) $N_a^{p-1}\langle\sigma v\rangle'_{B_1,B_2\cdots,B_p}$ from the chosen forward rate. However, the `DerivedRate` class may go further and compute Equation (13) by using interpolations of the tabulated values of the partition functions as requested. It is important to note that `DerivedRate` does not support constructing inverse weak rate reactions. This decision has been made because weak rates are not required to be in equilibrium to occur (see, e.g., Clifford & Tayler 1965; Seitenzahl et al. 2009), breaking the necessary assumptions to compute Equation (B9).

To show the effects of the partition function correction and the accuracy of the `DerivedRate` rates, we have defined the `iron56_end` network with ^{48,49,55,56}Cr, $^{52–56}$Mn, $^{52–56}$Fe, ^{55,56}Co, and ^{56}Ni species, along with n, p, and α. This network is inspired by the high-Z portion of the popular `aprox21` network, as illustrated in Figure 9.

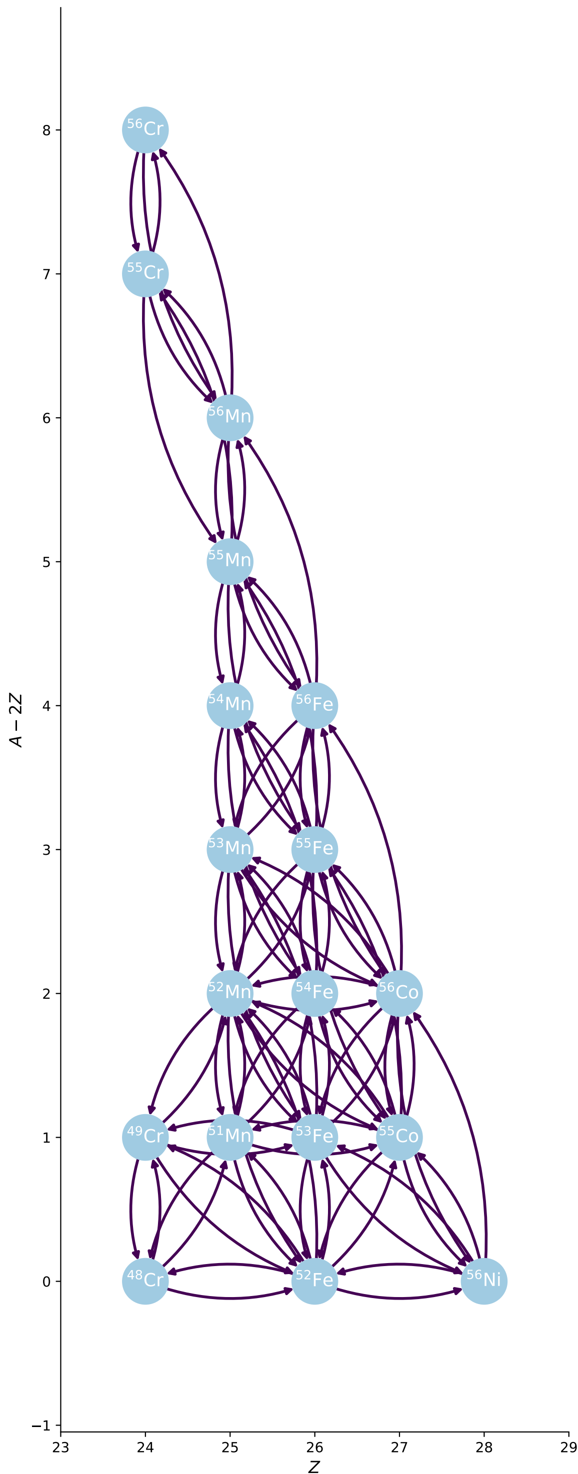


Figure 9. The integrated network `iron56_end` inspired by the high-Z portion of the popular `aprox21` network by replacing the REACLIB detailed balance reverse rates with the `DerivedRate` rates under `use_pf=True`. (supplemental notebook: `derived_network.ipynb`)

The required code to create `iron56_end` is:

```
rl = pyna.ReacLibLibrary()
fwd = rl.derived_forward()

nuclei = ["n", "p", "he4",
          "cr48", "cr49", "cr55",
          "cr56", "mn51", "mn52",
          "mn53", "mn54", "mn55",
          "mn56", "fe52", "fe53",
          "fe54", "fe55", "fe56",
          "co55", "co56", "ni56"]

frates = fwd.linking_nuclei(nuclist=nuclei,
                            with_reverse=False)
derived = []
for r in frates.get_rates():
    d = DerivedRate(rate=r, compute_Q=False,
                    use_pf=True)
    derived.append(d)

der_rates = Library(rates=derived)
full_library = frates + der_rates

pynet = PythonNetwork(libraries=full_library,
                      do_screening=True)
```

The screening and inverse screening factors of the `iron56_end` network are computed from Equations (19) and (C7), respectively. It is important to mention that Equation (C7) is computed under the assumption of the linear mixing rule, but not all the screening routines may satisfy this requirement (see, e.g., Wallace et al. 1982; Chugunov 2021). In order to cover these cases, we have implemented symmetric screening to the forward and inverse rates, i.e., the same enhancement factor assigned to the forward rate is assigned also to the inverse rate. We can explore the partition function temperature behavior by plotting $G_{n_i} = G_{n_i}(T)$ for different nuclei n_i of the network, as depicted in Figure 10. Notice that the nuclei with the same atomic number have a more similar behavior than the nuclei with the same atomic mass number.

To validate our results, we have compared the results obtained from the `iron56_end` network by replacing the REACLIB detailed balance rates with `DerivedRate` rates and replacing the results obtained by integrating the `iron56_end` network with just REACLIB detailed balance inverse rates. Setting $\rho = 10^8$ g cm^{-3}, $T = 7.0 \times 10^9$ K, and the composition

$$X(\mathrm{p}) = X(\mathrm{n}) = X(\alpha) = X(^{56}\mathrm{Ni}) = 5/37, \tag{14a}$$

$$X(^{48}\mathrm{Cr}) = 17/37, \tag{14b}$$

with all the remaining mass fractions set to zero as our initial conditions, we can show the comparison of these two reaction network calculations and introduce the role of the partition function in the previous reaction network under the same initial conditions (Figure 11). The convergence of the network with

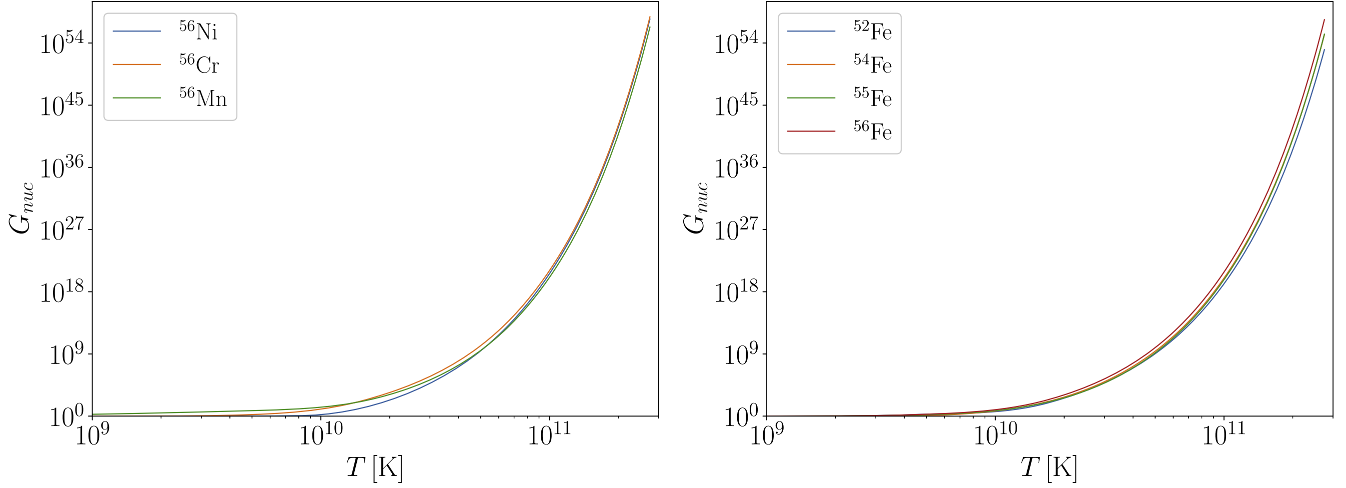


Figure 10. An example of the partition function temperature behavior of ^{56}Ni, ^{56}Cr, ^{56}Mn (left), and ^{52}Fe, ^{54}Fe, ^{55}Fe, ^{56}Fe (right). The partition function value of each nucleus, with identical atomic mass number A but different atomic number Z, changes significantly around 10^{10} K. This behavior is different in nuclei with different A and the same Z, where all the nuclei tend to have similar values around 10^{10} K. Although we should not generalize this behavior to all existing nuclei from this particular case, this behavior dominates in the nuclei of our network. (supplemental notebook: `plot_pf.ipynb`)

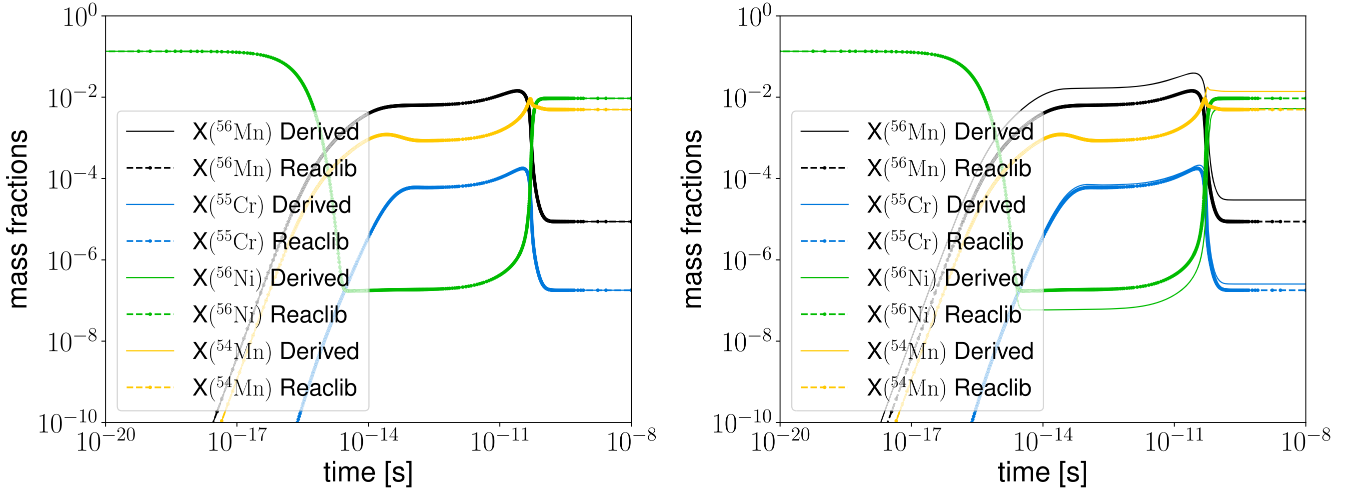


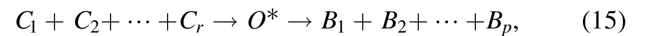
Figure 11. Comparing the nuclei ^{55}Cr, ^{54,56}Mn, and ^{56}Ni of the `iron56_end` network with the REACLIB detailed balanced inverse rates replaced by `DerivedRate` objects under the effects of the partition functions (right) and the `iron56_end` network comparison without any modification from the partition functions (left). The maximum molar fraction absolute error in the left case is $\sim 10^{-7}$. The density and temperature in both comparisons are $\rho = 10^8$ g cm^{-3} and $T = 7.0 \times 10^9$ K, respectively. (supplemental notebook: `iron56_end_networks.ipynb`)

the inclusion of partition function may differ significantly as we increase the temperature around $\sim 10^{10}$ K. This temperature regime is common in stellar atmospheres of neutron stars or inside the cores of white dwarfs, where the nuclear statistical equilibrium regime takes place.

7. Electron Screening

So far we have been treating the nuclei as if they were decoupled from the surrounding sea of electrons, but inside a star densities are sufficiently high to invalidate this approximation. Accurate reaction rate calculation requires us to account for Coulomb screening, where the free electrons surrounding each nucleus are preferentially attracted to the positively charged nuclei. This reduces the nucleus effective charge, thus lowering the Coulomb barrier to fusion and greatly enhancing reaction rates.

From Dewitt et al. (1973) and imposing the linear mixing rule, any reaction rate with a set of reactants $\{C_i\}$ and a set of products $\{B_j\}$,



$$C_1 + C_2 + \cdots + C_r \to O^* \to B_1 + B_2 + \cdots + B_p, \tag{15}$$

is enhanced owing to the Coulomb screening of the reactants, by a factor $f^{\rm cl}$ given by

$$f^{\rm cl}_{C_1 \cdots C_r \to O^*} = \exp\left[\frac{\mu^C(Z_{C_1}) + \cdots + \mu^C(Z_{C_r})}{k_B T} - \frac{\mu^C(Z_{C_1} + \cdots Z_{C_r})}{k_B T}\right], \tag{16}$$

where O^* is the composite nuclei of the reactants. We need to highlight that Equation (16) only describes the classical enhancement corrections. If we introduce quantum effects, the tunneling probability and the total width of each O^* channel modify the total enhancement factor (see more details in

Kushnir et al. 2019; Chugunov 2021) as follows:

$$f_{C_1\cdots C_r\to B_1\cdots B_p} = f^{\rm cl}_{C_1\cdots C_r\to O^*} f^q_{C_1\cdots C_r\to B_1\cdots B_p}, \tag{17}$$

where $f_{C_1\cdots C_r\to B_1\cdots B_p}$ is the total screening factor and $f^q_{C_1\cdots C_r\to B_1\cdots B_p}$ is the quantum enhancement sector given by

$$f^q_{C_1\cdots C_r\to B_1\cdots B_p} = f^q_{C_1\cdots C_r\to O^*} f^q_{B_1\cdots B_p\to O^*}\frac{\Gamma}{\tilde{\Gamma}}. \tag{18}$$

Γ and $\tilde{\Gamma}$ are the total width in vacuum and in the plasma, respectively. Note that $f^q_{C_1\cdots C_r\to B_1\cdots B_p} = f^q_{B_1\cdots B_p\to C_1\cdots C_r}$, i.e., the quantum enhancement screening sector is symmetric with respect to the reactants and the products. To implement these corrections, in principle, we need to sum over all the $N_a^{r-1}(\sigma v)^{\rm sc}_{C_1\cdots C_r}$ reaction channels in order to accurately compute $N_a^{r-1}\langle\sigma v\rangle^{\rm sc}_{C_1\cdots C_r}$.

However, in pynucastro we fix $f^q_{C_1\cdots C_r\to B_1\cdots B_p} = 1$; thus, the total enhancement factor used in our implementation is given by

$$f_{C_1\cdots C_r} = f^{\rm cl}_{C_1\cdots C_r\to O^*}. \tag{19}$$

This assumption may introduce an uncertainty of ≈10% in the screening factor, as in the pp-reaction (Kushnir et al. 2019). For our purposes, it is out of the scope to perform calculations on each state of the compound nuclei. However, an exceptional case discussed in Chugunov (2021) can be easily implemented in a future version of pynucastro.

From Equations (16) and (19), we need to understand the functional form of μ_i^C, which appears as a part of the free energy $F^{\rm sc} = \mu^{\rm C}/k_{\rm B}T$. In pynucastro we have implemented three different fits that construct F and perform the enhancement factor calculations in a reaction network: `chugunov_2007` from Chugunov et al. (2007), `chugunov_2009` from Chugunov & DeWitt (2009), and `potekhin_1998` from Chabrier & Potekhin (1998). Note that only Chabrier & Potekhin (1998) is available to be used in the nuclear statistical equilibrium (NSE) state that we discuss in the next section. Since this fit is implemented in both reaction networks and NSE, we will center our discussion on it.

We start by defining the ith nuclei Coulomb-coupling constant Γ_i of a multicomponent plasma (MCP) by

$$\Gamma_i = \Gamma_e Z_i^{5/3},\ \Gamma_e = \frac{e^2}{a_e k_{\rm B}T},\ a_e = \left(\frac{4\pi n_e}{3}\right)^{-1/3}. \tag{20}$$

The approximation $n_e \sim \rho Y_e/m_u$ is used in some references, for example, in Seitenzahl et al. (2009); however, in pynucastro we always update Y_e to be consistent with the current estimate of the composition. Two important container classes store the necessary information to compute the screening factor: `PlasmaState` and `ScreenFactors`. A `PlasmaState` object holds the temperature, the density, the molar fraction, and the atomic number values of each nucleus inside the reaction network to be screened. Similarly, a `ScreenFactors` object holds the reactant nucleus information of the rate to be screened. Given the two nuclei contained in a `ScreeningPair` object, a `ScreenFactors` object is initialized to hold the two-reactant nucleus information.

According to Chabrier & Potekhin (1998) and the definitions provided in Equation (20),

$$\begin{aligned} F_i^{\rm sc} = \frac{\mu_i^C}{k_{\rm B}T} = A_1\Big[&\sqrt{\Gamma_i(A_2+\Gamma_i)} \\ &- A_2\ln\left(\sqrt{\frac{\Gamma_i}{A_2}}\sqrt{1+\frac{\Gamma_i}{A_2}}\right)\Big] \\ &+ 2A_3[\sqrt{\Gamma_i} - \arctan(\sqrt{\Gamma_i})], \end{aligned} \tag{21}$$

where $A_1 = -0.9052$, $A_2 = 0.6322$, and $A_3 = -\sqrt{3}/2 - A_1/\sqrt{A_2}$ is valid within a range $0.01 < \Gamma_i \lesssim 170$. The `iron56_end` network, under the initial conditions (14a)–(14b), at $\rho = 10^8$ g cm^{-3} and $T = 7.0\times10^9$ K, defines a range $0.01 < \Gamma_i < 3.09$, which agrees within the boundaries of the fit. This fit is provided in pynucastro by the function `potekhin_1998`.

Once the enhancement calculations are completed and returned to the `RateCollection` network, we rely on the `ScreeningPair` class and the `evaluate_screening` method to hold the necessary information to compute the screening factor. The `ScreeningPair` class is a container of all the same two-nucleus reactant rates. The core idea of this class is to contain the required information, relative to the reactants. For REACLIB we use only one `ScreeningPair` object for each reaction, with the exception of the triple-α reaction. The method `evaluate_screening` constructs a `PlasmaState` object from the density, temperature, and molar composition of the network to compute the n_e factor, Γ_e, and Γ_i, respectively. It then creates a list with all the `ScreeningPair` objects required to screen the network, holding their information inside `ScreenFactors`. Finally, it uses these objects to compute the screening factor for each rate using the specified fit method. It is important to recall that we have described only the use of the `potekhin_1998` method, but `chugunov_2007` and `chugunov_2009` are also available in our implementation.

Finally, we note that it is easy to use one of the screening implementations when integrating a Python network—simply add the name of the screening function to the `args` tuple discussed in Section 3.2, and it will be called when evaluating the rates. For C++ networks, we have C++ ports of the screening routines that are automatically hooked in when using `AmrexAstroCxxNetwork`.

8. Nuclear Statistical Equilibrium

One additional feature we have implemented in pynucastro is the ability to compute the NSE mass fractions using several iterations of the hybrid Powell's method (Powell 1964, 1970). In Python, this is implemented by the SciPy `fsolve()` function. In C++, we converted the FORTRAN 77 implementation of Powell's method from MINPACK (More et al. 1980) to C++ as a templated header `library`. This solver is more robust than the traditional Newton–Raphson method in two ways. From our tests, Powell's method can handle a larger range of thermodynamic conditions than simple Newton–Raphson iterations. It is also capable of handling the problem of the Jacobian becoming singular during Newton–Raphson iterations. As pointed out in Lippuner & Roberts (2017), we will follow the same strategy to accelerate the solution convergence, by updating Y_e from the computed composition of the previous iteration.

We begin setting the NSE variables (ρ, T, Y_e) and an initial guess for ($\mu_p^{\rm id}$, $\mu_n^{\rm id}$)—we have chosen ($\mu_p^{\rm id}$, $\mu_n^{\rm id}$) to be (−3.5, −15.0); however, a different pair may be chosen if the solution does not converge uniformly. We start the first iteration with $\mu_i^C = 0$. The solution is then obtained by iterating over the following:

1. First, we compute the NSE mass fractions (see the derivation in Appendix D):

$$X_i = \frac{m_i}{\rho}(2J_i + 1)G(T)\left(\frac{m_i k_{\rm B} T}{2\pi\hbar^2}\right)^{3/2} \times \exp\left[\frac{Z_i\mu_p^{\rm id} + N_i\mu_n^{\rm id} + Q_i}{k_{\rm B}T} + \frac{Z_i\mu_p^C - \mu_i^C}{k_{\rm B}T}\right]. \quad (22)$$

2. Next, we use the predicted mass fractions in the constraint equations,

$$\sum_i X_i - 1 = 0 \quad (23a)$$

$$Y_e - \sum_i \frac{Z_i X_i}{A_i} = 0, \quad (23b)$$

and solve for $\mu_p^{\rm id}$ and $\mu_n^{\rm id}$. The first constraint equation ensures that the predicted mass fractions sum to 1, and the second enforces that the electron fraction of the state agrees with the electron fraction input.

3. From the computed ($\mu_p^{\rm id}$, $\mu_n^{\rm id}$) pair we evaluate the network composition of the nuclei, using Equation (22) and the electron number density n_e from

$$n_e = \frac{\rho}{m_u}\sum_i \frac{Z_i X_i}{A_i}, \quad (24)$$

where m_u is the atomic unit mass.

4. Using Equations (20) and (21), we compute μ_i^C by replacing the computed electron number density of the previous step (24). Is important to recall that we may take $\mu_i^C = 0$ for any iteration if no Coulomb screening is considered.

5. Finally, we compute $\mu_p^{\rm id}$ and $\mu_n^{\rm id}$, setting each nucleus μ_i^C to the value computed in the previous step and by the use of the hybrid Powell's method again.

We iterate this process until the values of $\mu_p^{\rm id}$ and $\mu_n^{\rm id}$ converge to an absolute and relative tolerance of $\sim 10^{-10}$. These values may be adjusted by hand, as the initial guess, to ensure the convergence of the solution.

The method `get_comp_nse` implemented in the `RateCollection` network follows the previous four steps to compute the solution, ensuring that μ_i^C is constant for each iteration, while the hybrid Powell's method computes the solution. A sample code that solves the NSE state given a set of nuclei with Coulomb correction is shown below:

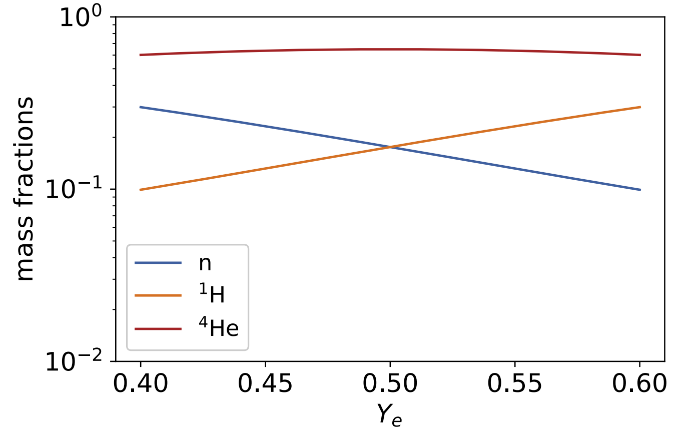


Figure 12. This figure shows the NSE mass fraction at a range of electron fractions from 0.4 to 0.6 at $\rho = 10^7$ g cm^{-3} and at temperature $T = 9.0 \times 10^9$ K. The nuclei p, n, and ^{4}He dominate at high temperature, and there is a symmetry correspondence between the mass fraction of p and n once $Y_e = 0.5$. A similar figure is observed in Figure 1 in Seitenzahl et al. (2008). (supplemental notebook: `nse.ipynb`)

```
rl = pyna.ReacLibLibrary()

nuclei = ["n", "p", "he4",
          "c12", "o16", "ne20",
          "mg24", "si28"]

lib = rl.linking_nuclei(nuclei)
rc = pyna.RateCollection(libraries=lib)

rho = 1.0e7     # gram/(cm^3)
T = 6.0e9       # Kelvin
ye = 0.5        # unitless electron fraction

nse_comp = rc.get_comp_nse(rho, T, ye,
                use_coulomb_corr=True)
```

Figures 12 and 13 are shown to compare the computed NSE distribution at different thermodynamic conditions to the ones obtained from the literature to show the validity of the solver. The nuclei that are not present in the corresponding figures for comparison are in dashed lines and transparent colors for better readability. Both Figures 12 and 13 show how NSE mass fractions change for a range of electron fractions from 0.4 to 0.6 at a fixed density, $\rho = 10^7$ g cm^{-3}, and temperature. Figure 12 is at a relatively high temperature, $T = 9.0 \times 10^9$ K. There is a domination of mass fractions by p, n, and α and a symmetry in mass fractions at $Y_e = 0.5$. Figure 13 is at a relatively low temperature, $T = 3.5 \times 10^9$ K. Mass fractions change drastically before $Y_e = 0.5$ but become steady and stable once $Y_e > 0.5$. Figures 1 and 3 in Seitenzahl et al. (2008) used the same thermodynamic conditions and have a similar result for comparison.

Finally, in order to compare how the reaction network agrees with the NSE calculations, we have considered the `iron56_end` network, described in Figure 9, under the initial conditions (14a–14b). By keeping the default options intact, we obtain the comparison between integrating the reaction network

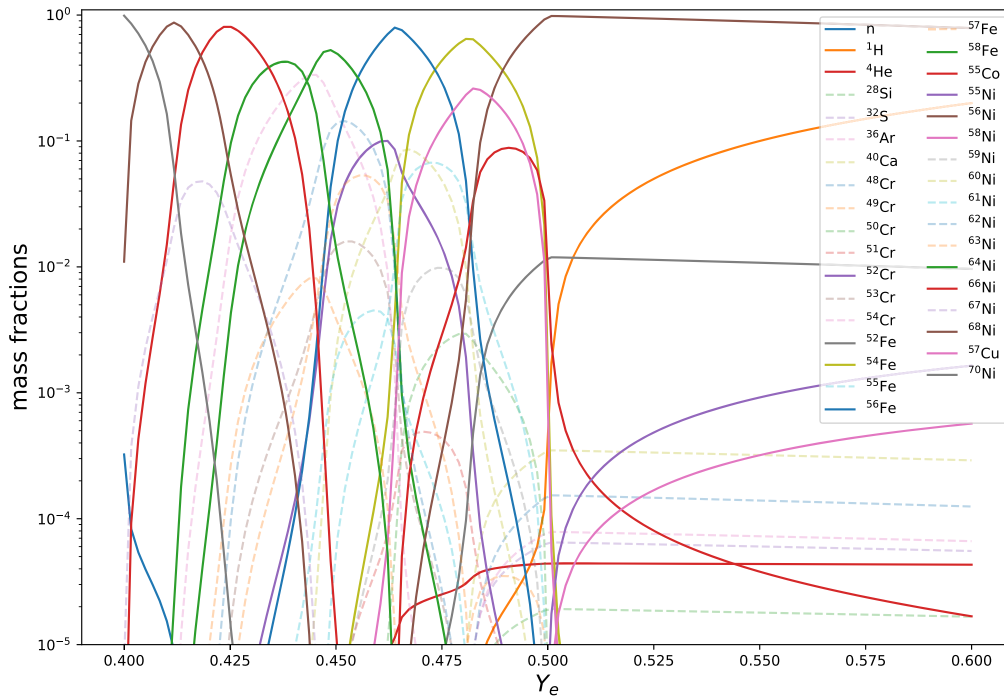


Figure 13. This figure shows the NSE mass fraction at a range of electron fractions from 0.4 to 0.6 at $\rho = 10^7$ g cm^{-3} and at temperature $T = 3.5 \times 10^9$ K. Dashed lines with transparent colors are nuclei that are not present in Figure 3 in Seitenzahl et al. (2008) for easier comparison. This figure shows a distinct behavior once $Y_e = 0.5$, which is observed in Seitenzahl et al. (2008). (supplemental notebook: `nse.ipynb`)

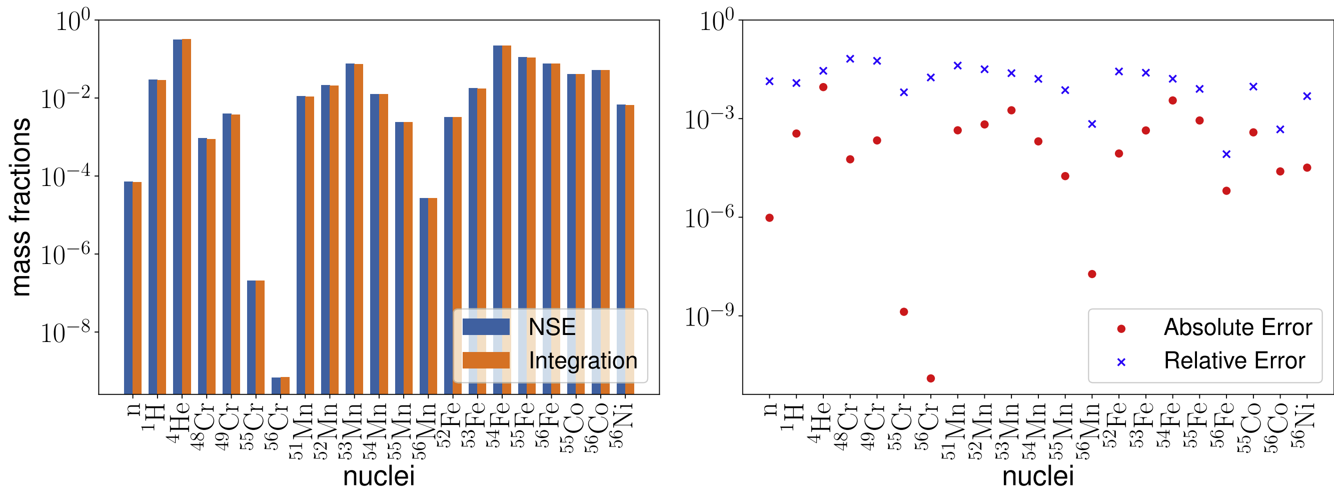


Figure 14. Comparing the mass fractions X_i of each nucleus n_i contained in the reaction network, by integrating the `iron56_end` reaction network and computing its NSE mass fractions (left). In addition, we compute the absolute and relative mass fraction differences with respect to each nucleus (right). The Coulomb correction is included in the NSE state and reaction network calculations. The density and temperature are $\rho = 10^8$ g cm^{-3} and $T = 7.0 \times 10^9$ K, respectively. (supplemental notebook: `iron56_end_integrate.ipynb`)

and computing the NSE state mass fractions, shown in Figure 14. From this comparison we can assert an excellent agreement between the NSE solve and integrating the network to steady state; however, it is important to mention that our initial conditions choice significantly impacts the convergence of the steady state of the network into the NSE state composition.

9. C++ Code Generation

pynucastro works closely with the AMReX-Astro simulation codes, Castro (Almgren et al. 2020) and MAESTROeX (Fan et al. 2019). The codes are written in C++, for performance portability, and therefore we need to output reaction networks in C++. A key concern for the C++ code generation is that it be able to work on both GPUs and CPUs. We do this in the AMReX-Astro framework, utilizing the AMReX (Zhang et al. 2019) data structures and lambda capturing of compute kernels to offload the entire network integration to GPUs. pynucastro generates the right-hand side and Jacobian of the network, as well as the metadata needed to interpret it, and this can be directly used by our codes. The C++ networks differ from the simple Python networks described above owing to the need to couple with hydrodynamics. Castro supports three different coupling strategies: an operator split approach where reactions and advection are independent of one another, a traditional spectral deferred corrections (SDC)

approach that supports second- and fourth-order space-and-time integration (Zingale et al. 2019), and a simplified SDC approach (Zingale et al. 2022). The C++ network generated by pynucastro needs to be compatible with each of these approaches. As part of this approach, we integrate mass fractions, X_k, or partial densities, (ρX_k), in the reaction network, instead of the molar fractions used internally in pynucastro.

Integrating astrophysical reaction networks requires stiff integrators, and a lot of work has been done on assessing the performance of different algorithms (Timmes 1999; Longland et al. 2014). The AMReX-Microphysics suite uses the VODE integrator (Brown et al. 1989) to integrate the reaction network. VODE is a fifth-order implicit backward difference integrator that handles stiff reaction networks well. VODE was originally written in FORTRAN 77, but we completely ported it to C++ to work in our GPU framework and added a number of integration step checks to help preserve $\sum_k X_k = 1$ without having to resort to renormalization of the species. The details of our integrator are given in Zingale et al. (2022).

The AMReX-Astro networks are implemented in header files to enable function inlining. pynucastro contains templates with these headers with keyword placeholders for the network-specific code to be injected. The `AmrexAstroCxxNetwork write_network()` function reads the templates and at each keyword encountered calls an appropriate function to write out the appropriate piece of the network, for instance, functions to evaluate reaction rates, ydot terms, and Jacobian. We represent the ydot terms as SymPy objects and use the SymPy C++ code generator to write the expressions to compute them. Since we evolve energy in the C++ networks, we include temperature derivatives in the Jacobian, and the rate functions use templating to compile this out when not needed (for instance, when using a numerical Jacobian). For our application codes, we generate the networks once and keep them under version control in our Microphysics project repository (AMReX-Astro Microphysics Development Team et al. 2022). Microphysics contains unit tests to exercise any network, which are run nightly. The C++ networks have the same screening functions as in pynucastro. Additionally, the C++ networks have a plasma neutrino loss term, which has not yet been ported to Python. This augments the energy from the network.

9.1. Comparison of Python to C++ Nets

To check for consistency between the Python and C++ networks, we do a simple one-zone burn. Although the C++ networks are often done with hydro evolution, or at least a temperature/energy equation (a self-heating network), we compare without any temperature evolution here. We compare using the `subch_approx` network—this is an approximate version of the network described in detail in Zingale et al. (2022). The code needed to generate it is given in Appendix E.

In our C++ Microphysics `burn_cell` unit test, we set the initial thermodynamic state and then instruct VODE to integrate by a Δt and return the new solution. In order to see the history of the evolution, we evolve to $t = 10^{-3}$ s in 100 increments, logarithmically spaced between 10^{-10} and 10^{-3} s. It is not enough to just disable the energy evolution (by setting $de/dt = 0$), but we also need to skip obtaining the temperature from the equation of state each step, since even with fixed e the composition change will cause T to change. With this set, we can compare the C++-VODE integration to the Python-Scipy `solve_ivp` integration (we use the `"bdf"` option, which is an implementation of Shampine & Reichelt 1997).

Figure 15 shows the comparison. We select $\rho = 10^6$ g cm^{-3}, $T = 2 \times 10^9$ K, and the initial composition 99% ^{4}He and 1% ^{14}N by mass. In the top panel, we show the history of the mass fractions from Python (solid lines) and C++ (points) and see that they lie right on top of one another. For clarity, we only plot mass fractions that go above $X = 10^{-5}$ at some point in their history. In the bottom panel, for those same species, we plot the difference between the Python and C++ abundances as a function of time. We see that this absolute error is always below 10^{-6}. This is despite the integrators being different implementations of a backward difference integrator. This shows that our C++ and Python code generation is remarkably consistent.

10. Summary and Future Work

There are many areas of development to focus on in the future. First, there are a lot of other examples of approximations to networks that could be added. For instance, CNO and hot CNO burning can be reduced to just a few rates, carbon and oxygen burning can likewise be simplified to a few endpoints, and approximations to iron-group equilibrium like those made in the `aprox19` network and others can make lower Y_e accessible with a reduced network. Adding a separate "photoionization" proton, as done in `aprox19`, is also desirable, since it reduces the complexity of the Jacobian of the system. We can build a derived class off of the `Rate` class to implement some of these approximations. Very specialized networks like the `iso7` network (Timmes et al. 2000) or the rp-process approximate network of Wallace & Woosley (1981) might require specialized code paths but are still possible to implement to allow for interactive exploration and exporting the network in different formats with updated rates.

The Python interface to nuclear reaction data and network structure allows for easier analysis of trends in reaction networks, like the work of Zhu et al. (2016) and Meyer (2018), and the development of machine-learning approaches to approximating nuclear kinetics (Fan et al. 2022).

We are also in the process of implementing two methods for reaction network reduction that were originally developed for the chemical reaction networks used in combustion simulations. The first method was originally outlined by Sun et al. (2010), and the second was developed by Pepiot-Desjardins & Pitsch (2008) and explored further in Niemeyer & Sung (2011). The pyMARS `library` (Mestas et al. 2019) implements these methods for chemical networks and inspired their inclusion in pynucastro.

We also want to expand the range of nuclear physics data sources we encompass. Since this is developed openly, on github, the hope is that others in the community will directly contribute new rates as they become available. In the near term, we will increase our coverage of weak rates. At the moment, we include those from Suzuki et al. (2016), since Urca was an initial application area. The tabulated weak rates of Fuller et al. (1985), Langanke & Martínez-Pinedo (2000), and others fit into the `TabularRate` framework well. This coverage will also allow us to generate NSE tables from large networks that include Y_e evolution that can be used within hydro codes, as in Ma et al. (2013), to capture the energetic and Y_e evolution from a large network in tandem to a more reasonably sized network used for advection. In this case, tabulating dY_e/dt in addition to

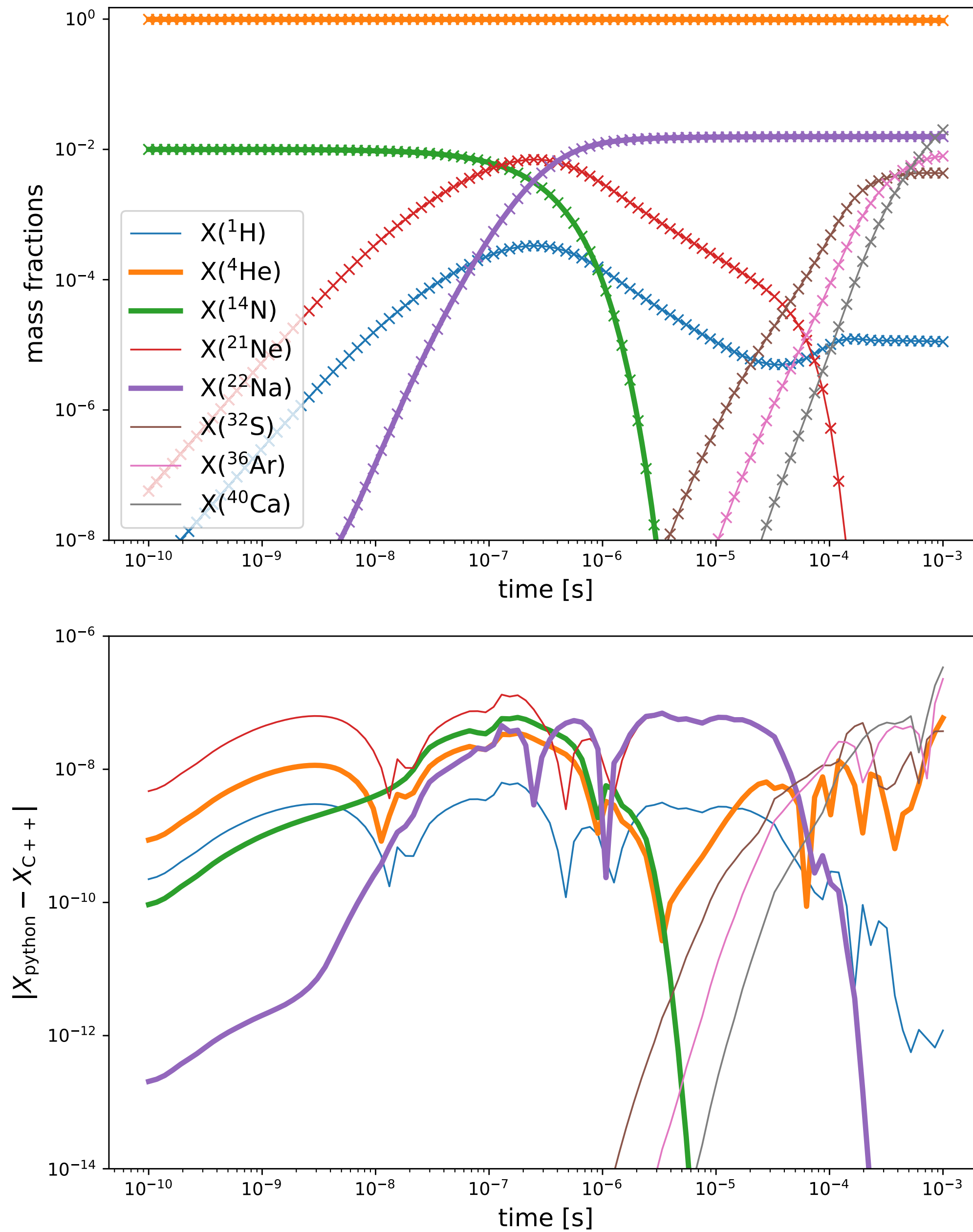


Figure 15. Comparison between the Python and C++ integration of the `subch_approx` network at fixed T and ρ. The top panel shows the mass fractions, with the Python result as lines and the C++ result as points. The bottom panel shows the difference between the two codes. (supplemental notebook: `subch_approx_comparison.ipynb`)

the NSE state would allow for a second-order-accurate predictor-corrector time integration scheme with NSE (especially when combined with the spectral deferred corrections integrator, as in Zingale et al. 2022).

Although our primary source for thermonuclear rates is REACLIB, other rate compilations exist in the community, including STARLIB (Sallaska et al. 2013). STARLIB goes beyond REACLIB in including error information associated with the tabulated rates. Support for STARLIB can be added in a straightforward manner by subclassing either `Rate` or `ReacLibRate` and would be a good project for a new contributor.

It is also straightforward to increase the application codes that we can output networks to. We use a template find-and-replace strategy to insert the functions needed to evaluate the network into the C++ code functions that communicate with the AMReX-Astro simulation codes. Similar templates can be used for other codes with few modifications to the code. This would be a good project for a student using a simulation code in their research.

Finally, the focus of pynucastro so far is on nuclear properties and networks, but support for more areas of nuclear astrophysics can be added as needed, such as wrappers for popular equations of state and neutrinos. Plasma neutrino losses would make it easy to plot ignition curves, for example, and since we keep track of the individual rates, we can handle the neutrino losses from H-burning reactions in a straightforward manner.

We thank Jim Truran for teaching us all about nuclear astrophysics. We thank Luna for her help sniffing out bugs in pynucastro.

The work at Stony Brook was supported by DOE/Office of Nuclear Physics grant DE-FG02-87ER40317. This material is based on work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of Nuclear Physics, Scientific Discovery through Advanced Computing (SciDAC) program under award No. DE-SC0017955. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. A.S.C. is funded by the Chilean Government ANID grant No. 56160017 and the U.S Department of State, Fulbright grant No. PS00280789.

Software: AMReX (Zhang et al. 2019), Jupyter (Kluyver et al. 2016), matplotlib (Hunter 2007), NetworkX (Hagberg et al. 2008), Numba (Lam et al. 2015), NumPy (Oliphant 2007; van der Walt et al. 2011), pytest (Krekel et al. 2004), SymPy (Meurer et al. 2017).

Appendix A
Ideal Boltzmann Gas Approximation

In order to relate the different thermodynamics quantities computed in a reaction network, an equation of state should be provided. In the pynucastro design we assumed the nondegenerate, noninteracting, and nonrelativistic regime for the nuclei, with exceptions discussed in Section 7. As we increase the temperature, keeping the density $\lesssim 10^{11}$ g cm^{-3}, the Boltzmann statistics governs the behavior of the nuclei involved in all the reaction networks. In this section we want to stress the relationship between the number density n, chemical potential μ, and temperature T under this approximation. These calculations are standard and can be found in many references (see, e.g., Hill & Hill 1960).

The grand-canonical partition function $\mathcal{Z}$ for k subsystems is given by

$$\mathcal{Z} = \sum_k (z^k Z_k), \tag{A1}$$

where Z_k is the canonical partition function for the subsystem with k identical particles, $z = e^{\mu/k_{\rm B}T}$ is the fugacity parameter, and $k_{\rm B}$ is the Boltzmann constant. From here

$$Z_k = \frac{Z_{1-p}^k}{k!}, \tag{A2}$$

where $Z_{1\text{-}p}$ is the canonical partition function for only one particle,

$$Z_{1-p} = \sum_l \int d^3\boldsymbol{p}\ d^3\boldsymbol{x}\ e^{-\epsilon(l, p_i, x_i)/k_{\rm B}T}, \tag{A3}$$

where $\boldsymbol{x}$ are the space parameters, $\boldsymbol{p}$ are the momentum parameters, and l labels the internal degrees of freedom. For a spherically symmetric potential bounded particle, the energy of each nucleus is given by

$$\epsilon(l, \boldsymbol{p}, \boldsymbol{x}) = \frac{\boldsymbol{p}^2}{2m} + \Delta_l + mc^2 + \mu^{\rm C}, \tag{A4}$$

with $\boldsymbol{p}^2 = \boldsymbol{p}\cdot\boldsymbol{p}$ as the squared momentum magnitude, Δ_l as the discrete energy levels of each particle, mc^2 as the rest energy, and $\mu^{\rm C}$ as the Coulomb screening constant, which is the consequence of the electron cloud interactions that surround each ionized nucleus.

Combining Equations (A1)–(A4), we may compute $\mathcal{Z}$ by

$$\mathcal{Z} = \sum_k (2J_l + 1) e^{-\Delta_l/k_{\rm B}T} V \left(\frac{mk_{\rm B}T}{2\pi\hbar^2}\right)^{3/2} e^{(\mu - mc^2 - \mu^C)/k_{\rm B}T}, \tag{A5}$$

where V is the volume, J_l is the l-energy level spin of the nuclei, and m is the mass of the nuclei. Thus, the grand-canonical potential Ω is given by

$$\begin{aligned}\Omega &= -k_{\rm B}T \log \mathcal{Z} \\ &= -k_{\rm B}T \sum_l (2J_l + 1) e^{-\Delta_l/(k_{\rm B}T)} V \left(\frac{mk_{\rm B}T}{2\pi\hbar^2}\right)^{3/2} e^{(\mu - mc^2 - \mu^C)/k_{\rm B}T}.\end{aligned} \tag{A6}$$

The number of particles as a function of (T, V, μ), where μ is the chemical potential, can be expressed as

$$N = -\left(\frac{\partial\Omega}{\partial\mu}\right)_{V,T} = V\left[\sum_{l}(2J_l + 1)e^{-\Delta_l/k_{\rm B}T}\right]\left(\frac{mk_{\rm B}T}{2\pi\hbar^2}\right)^{3/2} e^{(\mu - mc^2 - \mu^{\rm C})/k_{\rm B}T}. \tag{A7}$$

After identifying the number density $n = N/V$ and introducing the normalized partition function definition (Fowler et al. 1967; Rauscher 2003),

$$(2J_0 + 1)G(T) = \sum_{l}(2J_l + 1)e^{-\Delta_l/k_{\rm B}T}, \tag{A8}$$

where J_0 is the nucleus ground-state spin, we can rewrite Equation (A7) in terms of μ as

$$\mu = k_{\rm B}T\log\left[\frac{n}{(2J_0 + 1)G(T)}\left(\frac{2\pi\hbar^2}{mk_{\rm B}T}\right)^{3/2}\right] + mc^2 + \mu^{\rm C}. \tag{A9}$$

Finally, we can define the kinematic term of the chemical potential by

$$\mu^{\rm id} = k_{\rm B}T\log\left[\frac{n}{(2J_0 + 1)G(T)}\left(\frac{2\pi\hbar^2}{mk_{\rm B}T}\right)^{3/2}\right], \tag{A10}$$

suggesting $\mu = \mu^{\rm id} + mc^2 + \mu^{\rm C}$. The case $\mu^{\rm C} = 0$ is the noninteracting or no-screening case. By computing the derivatives of Equation (A6), it is possible to compute the entropy per ion and pressure in terms of n and T (Lippuner & Roberts 2017).

Appendix B
Reverse Rates and Detailed Balance Calculations

The relationship between the inverse and the forward reaction may be obtained by detailed balance calculations (see, e.g., Clayton 1968; Arnett 1996; Rauscher et al. 1997; Angulo et al. 1999; Cyburt et al. 2010; Lippuner & Roberts 2017). Here we describe how the detailed balance calculations are performed. A nuclear reaction between a set of reactants $\{C_i\}$ and a set of products $\{B_j\}$ in equilibrium is represented by

$$C_1 + C_2 + \cdots + C_r \rightleftharpoons B_1 + B_2 + \cdots + B_p, \tag{B1}$$

where r and p represent the number of reactants and products, respectively. The right-pointing arrow represents a forward reaction, while the left-pointing arrow represents a reaction in the opposite direction known as the inverse reaction. After applying Equation (1) to the forward reaction with r reactants,

$$\frac{1}{c_{C_1}}\frac{dn_{C_1}}{dt} = \frac{1}{c_{C_2}}\frac{dn_{C_2}}{dt} = \cdots = \frac{1}{c_{C_r}}\frac{dn_{C_r}}{dt} = -\frac{n_{C_1}n_{C_2}\cdots n_{C_r}}{\prod_{C_i} c_{C_i}!}\langle\sigma v\rangle_{C_1C_2\cdots C_r}, \tag{B2}$$

and to the inverse reaction with the p products as reactants,

$$\frac{1}{c_{B_1}}\frac{dn_{B_1}}{dt} = \frac{1}{c_{B_2}}\frac{dn_{B_2}}{dt} = \cdots = \frac{1}{c_{B_p}}\frac{dn_{B_p}}{dt} = -\frac{n_{B_1}n_{B_2}\cdots n_{B_p}}{\prod_{B_i} c_{B_i}!}\langle\sigma v\rangle_{B_1B_2\cdots B_p} \tag{B3}$$

where n_{C_i} is the nuclei C_i number density, c_{C_i} is the count of the reactant nuclei C_i, and similarly c_{B_i} is the count of the product nuclei B_i. From Equations (B2) and (B3) and imposing the conservation of nucleons,

$$\frac{\langle\sigma v\rangle_{B_1B_2\cdots B_r}}{\langle\sigma v\rangle_{C_1C_2\cdots C_r}} = \frac{n_{C_1}n_{C_2}\cdots n_{C_r}}{n_{B_1}n_{B_2}\cdots n_{B_p}} \times \frac{\prod_{B_i} c_{B_i}!}{\prod_{C_i} c_{C_i}!}. \tag{B4}$$

Furthermore, after computing n_{N_i} in terms of μ_{N_i}, from Equation (A9), for each N_ith nucleus, and using the approximation $m_{N_i} = A_{N_i}m_u$,

$$n_{N_i} = (2J_0 + 1)A_{N_i}^{3/2}\left(\frac{m_u k_{\rm B}T}{2\pi\hbar^2}\right)^{3/2} e^{(\mu_{N_i} - m_{N_i}c^2 - \mu_{N_i}^{\rm C})/k_{\rm B}T} G_{N_i}(T). \tag{B5}$$

From Equations (B4) and (B5) we obtain, depending on the value of μ^{C}, a relationship between the forward and reverse rate. The case $\mu^{\mathrm{C}} = 0$ is the no-screening case, where $\mu^{\mathrm{C}} \neq 0$ implies the presence of Coulomb screening.

Case $\mu^{\mathrm{C}} = 0$: In this case, the previous substitution leads to

$$\frac{N_a^{p-1}\langle\sigma v\rangle_{B_1B_2\cdots B_r}}{N_a^{r-1}\langle\sigma v\rangle_{C_1C_2\cdots C_r}} = \left(\frac{1}{N_a}\right)^{r-p}\frac{(2J_{C_1}+1)\cdots(2J_{C_r}+1)}{(2J_{B_1}+1)\cdots(2J_{B_p}+1)}\left(\frac{A_{C_1}\cdots A_{C_r}}{A_{B_1}\cdots A_{B_p}}\right)^{3/2}\left(\frac{m_u k_{\mathrm{B}}}{2\pi\hbar^2}\right)^{\frac{3}{2}(r-p)} T^{\frac{3}{2}(r-p)}$$
$$\times \frac{G_{C_1}\cdots G_{C_r}(T)}{G_{B_1}\cdots G_{B_p}(T)}\frac{\prod_{B_i} c_{B_i}!}{\prod_{C_i} c_{C_i}!}\exp\left[-\frac{1}{k_B T}\left(\sum_{i=1}^{r} m_{C_i} - \sum_{i=1}^{p} m_{B_i}\right)c^2\right]$$
$$\times \exp\left[\frac{1}{k_{\mathrm{B}}T}\left(\sum_{i=1}^{r}\mu_{C_i} - \sum_{i=1}^{p}\mu_{B_i}\right)\right], \tag{B6}$$

where N_a is Avogadro's number, J_{N_i} is the spin value of the N_ith nucleus ground state, c_{N_i} is the count of nuclei N_i in the reaction, and $G_{N_i}(T)$ is the ground spin-state normalized nuclear partition function (A8), with $\Delta_{N_i,l}$ and $J_{N_i,l}$ as the energy spectrum and the l-energy level spin of the nuclei N_i, respectively. The quantities G_{N_i} are constructed and tabulated between 10^7 and 10^{11} K in Rauscher et al. (1997) and Rauscher (2003). In pynucastro we import these tables and convert them into a suitable format to be merged; consequently, no calculations of Equation (A8) are performed. Each A_{n_i} represents the atomic weight of the nuclei n_i, and we define Q as the capture energy of the reaction, by

$$Q = \left(\sum_{i=1}^{r} m_i - \sum_{i=1}^{p} m_p\right)c^2. \tag{B7}$$

Since the reaction is in equilibrium, we can assert that

$$\sum_{i=1}^{r}\mu_{C_i} - \sum_{i=1}^{p}\mu_{B_i} = 0. \tag{B8}$$

Using Equations (B7) and (B8), we can write the detailed balance inverse rate of the forward rate $N_a^{r-1}\langle\sigma v\rangle_{C_1C_2\cdots C_r}$ by

$$N_a^{p-1} <\sigma v>_{B_1,B_2,\cdots,B_p} = \left(\frac{1}{N_a}\right)^{r-p}\frac{(2J_{C_1}+1)\cdots(2J_{C_r}+1)}{(2J_{B_1}+1)\cdots(2J_{B_p}+1)}\left(\frac{A_{C_1}\cdots A_{C_r}}{A_{B_1}\cdots A_{B_p}}\right)^{3/2}\left(\frac{m_u k_{\mathrm{B}}}{2\pi\hbar^2}\right)^{\frac{3}{2}(r-p)} T^{\frac{3}{2}(r-p)}$$
$$\times \frac{G_{C_1}\cdots G_{C_r}(T)}{G_{B_1}\cdots G_{B_p}(T)}\frac{\prod_{B_i} c_{B_i}!}{\prod_{C_i} c_{C_i}!}\mathrm{e}^{-Q/k_{\mathrm{B}}T} N_a^{r-1} <\sigma v>_{C_1,C_2,\cdots,C_r}. \tag{B9}$$

From Equation (B9), the determination of the quantities J_{n_i} and Q becomes essential in the goal of computing the reverse rates. In REACLIB, each rate $N_a^{n-1}\langle\sigma v\rangle_{n_1,n_2\cdots,n_n}$ is represented by seven parameters described in Equation (4). After setting each partition function by $G_{n_i}(T) = 1$ in Equation (B9), replacing Equation (4), and taking the logarithm on both sides, we may rewrite Equation (B9) as Equation (13), where $N_a^{p-1}\langle\sigma v\rangle_{B_1,B_2\cdots,B_p}{}'$ should satisfy fit (4) with the set $a_{0,\mathrm{rev}}, \cdots, a_{6,\mathrm{rev}}$ of reverse coefficients described in Rauscher et al. (1997), but generalized for r reactants and p products as

$$a_{0,\mathrm{rev}} = \log\left\{\frac{(2J_{C_1}+1)\cdots(2J_{C_r}+1)}{(2J_{B_1}+1)\cdots(2J_{B_p}+1)}\left(\frac{A_{C_1}\cdots A_{C_r}}{A_{B_1}\cdots A_{B_p}}\right)^{3/2}\frac{\prod_{B_i} c_{B_i}!}{\prod_{C_i} c_{C_i}!}F\right\} + a_0 \tag{B10a}$$

$$a_{1,\mathrm{rev}} = a_1 - Q/(10^9\ \mathrm{K}\cdot k_{\mathrm{B}}) \tag{B10b}$$

$$a_{2,\mathrm{rev}} = a_2 \tag{B10c}$$

$$a_{3,\mathrm{rev}} = a_3 \tag{B10d}$$

$$a_{4,\mathrm{rev}} = a_4 \tag{B10e}$$

$$a_{5,\mathrm{rev}} = a_5 \tag{B10f}$$

$$a_{6,\mathrm{rev}} = a_6 + \frac{3}{2}(r-p), \tag{B10g}$$

with F defined by

$$T_9^{\frac{3}{2}(r-p)}F = \left(\frac{1}{N_a}\right)^{r-p}\left(\frac{m_u k_{\rm B}}{2\pi\hbar^2}\right)^{\frac{3}{2}(r-p)} T^{\frac{3}{2}(r-p)}. \tag{B12}$$

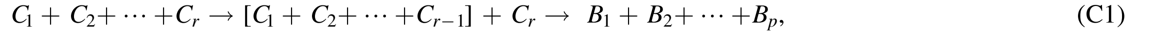
Case $\mu^{\rm C} \neq 0$: In this case, the rates are screened, and due to $\mu^{\rm C} \neq 0$, we have to incorporate the screening interactions in the detailed balance calculations. We may find an example of these calculations in Seitenzahl et al. (2009) and Calder et al. (2007). From Equations (B9) and (B5),

$$\begin{aligned} N_a^{p-1} <\sigma v>^{\rm sc}_{B_1,B_2,\cdots,B_p} &= \left(\frac{1}{N_a}\right)^{r-p}\frac{(2J_{C_1}+1)\cdots(2J_{C_r}+1)}{(2J_{B_1}+1)\cdots(2J_{B_p}+1)}\left(\frac{A_{C_1}\cdots A_{C_r}}{A_{B_1}\cdots A_{B_p}}\right)^{3/2}\left(\frac{m_u k_{\rm B}}{2\pi\hbar^2}\right)^{\frac{3}{2}(r-p)} T^{\frac{3}{2}(r-p)} \\ &\times \frac{G_{C_1}\cdots G_{C_r}(T)}{G_{B_1}\cdots G_{B_p}(T)}\frac{\prod_{B_i} c_{B_i}!}{\prod_{C_i} c_{C_i}!} {\rm e}^{-Q/k_{\rm B}T} \\ &\times \exp\left[-\frac{1}{k_{\rm B}T}\left(\sum_{i=1}^{r}\mu^C_{C_i} - \sum_{i=1}^{p}\mu^C_{B_i}\right)\right] N_a^{r-1} <\sigma v>^{\rm sc}_{C_1,C_2,\cdots,C_r}. \end{aligned} \tag{B13}$$

This equation will become essential for the discussion in the next appendix.

Appendix C
Electron Screening: Theoretical Considerations

The Coulomb screening enhancement factors (16) and (19) of a reaction can be decomposed as the product of the enhancement factors of the intermediate reactions. Similarly, from Equation (B13) we can assert that the enhancement factor of an inverse detailed balance reaction replaces the reactants by the products in Equation (19). Let us explore these interesting features of the factor (19). From the chain of rates



$$C_1 + C_2 + \cdots + C_r \rightarrow [C_1 + C_2 + \cdots + C_{r-1}] + C_r \rightarrow B_1 + B_2 + \cdots + B_p, \tag{C1}$$

where the brackets denote a composite nucleus obtained as a result of the $r-1$ original C_1, C_2, $\cdots$ C_{r-1} nucleus collisions, we have

$$\begin{aligned} & f_{C_1\cdots C_{r-1}} f_{[C_1+\cdots+C_{r-1}],C_r} = e^{[\mu^{\rm C}(Z_{C_1})+\cdots+\mu^{\rm C}(Z_{C_{r-1}})-\mu^{\rm C}(Z_{C_1}+\cdots Z_{C_{r-1}})]/k_{\rm B}T} \\ & \times e^{[\mu^{\rm C}(Z_{C_1}+\cdots+Z_{C_{r-1}})+\mu^{\rm C}(Z_{C_r})-\mu^{\rm C}(Z_{C_1}+\cdots Z_{C_r})]/k_{\rm B}T} \\ & = e^{[\mu^{\rm C}(Z_{C_1})+\cdots+\mu^{\rm C}(Z_{C_r})-\mu^{\rm C}(Z_{C_1}+\cdots Z_{C_r})]/k_{\rm B}T} \\ & = f_{C_1\cdots C_r}. \end{aligned} \tag{C2}$$

Therefore, independent of the number of reactants, each rate can be screened in sequences of pairs until all the remaining reactants are exhausted. Using this property, screening factors of reactions like the triple-α chain,

$$\alpha + \alpha + \alpha \rightarrow {}^8{\rm Be} + \alpha \rightarrow {}^{12}{\rm C}, \tag{C3}$$

may be computed in two steps: first we compute $f_{\alpha\alpha}$ and $f_{^8{\rm Be},\alpha}$, and then we multiply them to obtain the original rate $f_{\alpha\alpha\alpha}$ screening factor.

The quantities $\mu_i^{\rm C}$ required to compute the screening factors are also required to perform the NSE state calculations of the network, as shown in Equation (22). Therefore, the electron screening not only enhances the reaction network convergence but also modifies its NSE state. Similarly, according to the extra term $\mu^{\rm C}$ from Equation (A9), the reverse reactions are also modified by the screening effects into Equation (B13), altering the performance of the network convergence (Kushnir et al. 2019). Let us examine how the detailed balance calculations (B9) are modified by the inclusion of $\mu^{\rm C}$ in Equation (B13), where

$$N_a^{r-1} <\sigma v>^{\rm sc}_{C_1,C_2,\cdots,C_r} = N_a^{r-1} <\sigma v>^{\rm no-sc}_{C_1,C_2,\cdots,C_r} \times {\rm e}^{\left(\sum_{C_i}\mu^{\rm C}_{C_i} - \mu^{\rm C}_{C_1+C_2+\cdots+C_r}\right)/k_{\rm B}T} \tag{C4}$$

is the forward screened rate expressed in terms of the forward unscreened rate by Equation (19). Substituting this into Equation (B13),

$$N_a^{p-1} <\sigma v>^{\rm sc}_{B_1,B_2,\cdots,B_p} = \left(\frac{1}{N_a}\right)^{r-p} \frac{(2J_{C_1}+1)\cdots(2J_{C_r}+1)}{(2J_{B_1}+1)\cdots(2J_{B_p}+1)} \left(\frac{A_{C_1}\cdots A_{C_r}}{A_{B_1}\cdots A_{B_p}}\right)^{3/2} \left(\frac{m_u k_{\rm B}}{2\pi\hbar^2}\right)^{\frac{3}{2}(r-p)} T^{\frac{3}{2}(r-p)}$$
$$\times \frac{G_{C_1}\cdots G_{C_r}(T)}{G_{B_1}\cdots G_{B_p}(T)} \frac{\prod_{B_i} c_{B_i}!}{\prod_{C_i} c_{C_i}!} {\rm e}^{-Q/k_{\rm B}T} N_a^{r-1} <\sigma v>^{\rm no-sc}_{C_1,C_2,\cdots,C_r}$$
$$\times {\rm e}^{\left(\sum_{C_i}\mu^C_{C_i} - \mu^C_{C_1+C_2+\cdots+C_r}\right)\Big/k_{\rm B}T} {\rm e}^{\left(-\sum_{C_i}\mu^{\rm C}_{C_i} + \sum_{B_i}\mu^{\rm C}_{B_i}\right)\Big/k_{\rm B}T}, \tag{C5}$$

and from the conservation of charge $Z_{C_1} + \cdots + Z_{C_r} = Z_{B_1} + \cdots + Z_{B_p}$,

$$\mu^{\rm C}_{C_1+\cdots+C_r} = \mu^{\rm C}_{B_1+\cdots+B_p}, \tag{C6}$$

we can write, from Equations (C5) and (C6) and by identifying $N_a^{p-1}\langle\sigma v\rangle^{\rm no-sc}_{B_1,B_2,\cdots,B_p}$ from Equation (B9),

$$N_a^{p-1} <\sigma v>^{\rm sc}_{B_1,B_2,\cdots,B_p} = N_a^{p-1} <\sigma v>^{\rm no-sc}_{B_1,B_2,\cdots,B_p} \times {\rm e}^{\left(\sum_{C_i}\mu^{\rm C}_{C_i} - \mu^{\rm C}_{C_1+C_2+\cdots+C_r}\right)\Big/k_{\rm B}T} {\rm e}^{\left(-\sum_{C_i}\mu^{\rm C}_{C_i} + \sum_{B_i}\mu^{\rm C}_{B_i}\right)\Big/k_{\rm B}T}$$
$$= N_a^{p-1} <\sigma v>^{\rm no-sc}_{B_1,B_2,\cdots,B_p} \times {\rm e}^{\left(\sum_{B_i}\mu^{\rm C}_{B_i} - \mu^{\rm C}_{B_1+B_2+\cdots+B_r}\right)\Big/k_{\rm B}T}$$
$$= N_a^{p-1} <\sigma v>^{\rm no-sc}_{B_1,B_2,\cdots,B_p} \times f_{B_1\cdots B_p}, \tag{C7}$$

where $f_{B_1\cdots B_p}$ is the screening factor of the reverse reaction rate. This is a very powerful result because each rate, independent of its forward/inverse nature, may be computed by using its reactants/products, respectively. Therefore, in order to implement screening in detailed balanced reverse rates, we need to just consider the reversed rate in the definition of the enhancement factor (19), leaving the unscreened detailed balance calculations unchanged.

Appendix D
Computing the NSE State

The NSE state of a network describes a unique composition in which each nucleus's set of protons and neutrons are in equilibrium if the protons and neutrons that assemble the nuclei were free, given a set of thermodynamic state variables and electron fraction (see, e.g., Kushnir & Katz 2020). In this state, the energy required to assemble the ith nuclei is just the energy required to set up Z_i free protons and N_i free neutrons (Clifford & Tayler 1965). This condition is given by

$$\mu_i = Z_i\mu_p + N_i\mu_n. \tag{D1}$$

Using Equation (A9),

$$\mu_i^{\rm id} + m_i c^2 + \mu_i^{\rm C} = Z_i(\mu_p^{\rm id} + m_p c^2 + \mu_p^{\rm C}) + N_i(\mu_n^{\rm id} + m_n c^2 + \mu_n^{\rm C}), \tag{D2}$$

or equivalently, after setting $\mu_n^{\rm C} = 0$, because neutrons are not charged, and introducing the binding energy of the ith nuclei,

$$Q_i = (Z_i m_p + N_i m_n - m_i)c^2, \tag{D3}$$

we can write

$$\mu_i^{\rm id} = Z_i\mu_p^{\rm id} + N_i\mu_n^{\rm id} + Q_i - \mu_i^{\rm C} + Z_i\mu_p^{\rm C}. \tag{D4}$$

From Equations (D4) and (A10), and using the nucleon fraction X_i definition $n_i = \rho X_i/m_i$, we may finally compute Equation (22).

Appendix E
The `subch_approx` Network

The `subch_approx` network is designed to model explosive helium and carbon burning, containing the nuclei in the standard `aprox13` network, as well as other nuclei and pathways identified in Shen & Bildsten (2009) for bypassing the ^{12}C(α, γ)^{16}O rate. The original version of this network appeared in Zingale et al. (2022). This approximate version approximates some of the (α, p)(p,

γ) rates, as described in Section 4. It also modifies rates, as described in that same section, to change the endpoints, assuming fast neutron captures. It is generated as follows:

```
rl = pyna.ReacLibLibrary()

nucs = ["p", "he4",
        "c12", "o16", "ne20",
        "mg24", "si28", "s32",
        "ar36", "ca40", "ti44",
        "cr48", "fe52", "ni56",
        "al27", "p31", "cl35",
        "k39", "sc43", "v47",
        "mn51", "co55",
        "n13", "n14", "f18",
        "ne21", "na22", "na23"]

subch = rl.linking_nuclei(nucs)

other = [(("c12", "c12"), ("mg23", "n"), ("mg24")),
         (("o16", "o16"), ("s31", "n"), ("s32")),
         (("c12", "o16"), ("si27", "n"), ("si28"))]

for r, p, mp in other:
    rfilter = pyna.RateFilter(reactants=r,
                              products=p)
    _library = rl.filter(rfilter)
    r = _library.get_rates()[0]
    r.modify_products(mp)
    subch += _library

net = pyna.PythonNetwork(libraries=[subch],
                         symmetric_screening=True)
net.make_ap_pg_approx(intermediate_nuclei=["cl35", "k39", "sc43",
                                           "v47", "mn51", "co55"])
net.remove_nuclei(["cl35", "k39", "sc43",
                   "v47", "mn51", "co55"])
```

This network is visualized in Figure 16.

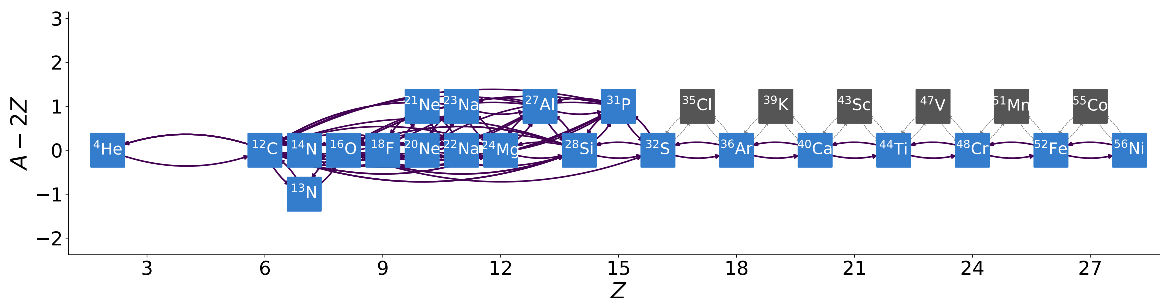


Figure 16. The `subch_approx` network.

ORCID iDs

Alexander I. Smith https://orcid.org/0000-0001-5961-1680
Eric T. Johnson https://orcid.org/0000-0003-3603-6868
Zhi Chen https://orcid.org/0000-0002-2839-107X
Kiran Eiden https://orcid.org/0000-0001-6191-4285
Donald E. Willcox https://orcid.org/0000-0003-2300-5165
Brendan Boyd https://orcid.org/0000-0002-5419-9751
Lyra Cao https://orcid.org/0000-0002-8849-9816
Christopher J. DeGrendele https://orcid.org/0000-0002-7815-1496
Michael Zingale https://orcid.org/0000-0001-8401-030X

References

Aikawa, M., Arnould, M., Goriely, S., Jorissen, A., & Takahashi, K. 2005, A&A, 441, 1195

Almgren, A., Sazo, M. B., Bell, J., et al. 2020, JOSS, 5, 2513

AMReX-Astro Microphysics Development Team, Bishop, A., Fields, C. E., et al. 2022, AMReX-Astro/Microphysics: Release 22.10, Zenodo, doi:10.5281/zenodo.7133136

Angulo, C., Arnould, M., Rayet, M., et al. 1999, NuPhA, 656, 3

Arnett, D. 1996, Supernovae and Nucleosynthesis: An Investigation of the History of Matter from the Big Bang to the Present (Princeton, NJ: Princeton Univ. Press)

Brown, P. N., Byrne, G. D., & Hindmarsh, A. C. 1989, SJSC, 10, 1038

Calder, A. C., Townsley, D. M., Seitenzahl, I. R., et al. 2007, ApJ, 656, 313

Calder, A. C., Willcox, D. E., DeGrendele, C. J., et al. 2019, JPhCS, 1225, 012002
Caughlan, G. R., & Fowler, W. A. 1988, ADNDT, 40, 283
Chabrier, G., & Potekhin, A. Y. 1998, PhRvE, 58, 4941
Chamulak, D. A., Brown, E. F., Timmes, F. X., & Dupczak, K. 2008, ApJ, 677, 160
Chugunov, A. I. 2021, JPhCS, 1787, 012047
Chugunov, A. I., & DeWitt, H. E. 2009, PhRvC, 80, 014611
Chugunov, A. I., DeWitt, H. E., & Yakovlev, D. G. 2007, PhRvD, 76, 025028
Clayton, D. D. 1968, Principles of Stellar Evolution and Nucleosynthesis (New York: McGraw-Hill)
Clifford, F. E., & Tayler, R. J. 1965, MNRAS, 129, 104
Cyburt, R. H., Amthor, A. M., Ferguson, R., et al. 2010, ApJS, 189, 240
Dewitt, H. E., Graboske, H. C., & Cooper, M. S. 1973, ApJ, 181, 439
Fan, D., Nonaka, A., Almgren, A. S., Harpole, A., & Zingale, M. 2019, ApJ, 887, 212
Fan, D., Willcox, D. E., DeGrendele, C., Zingale, M., & Nonaka, A. 2022, ApJ, 940, 134
Fowler, W. A., Caughlan, G. R., & Zimmerman, B. A. 1967, ARA&A, 5, 525
Fuller, G. M., Fowler, W. A., & Newman, M. J. 1985, ApJ, 293, 1
Hagberg, A. A., Schult, D. A., & Swart, P. J 2008, in Proceedings of the 7th Python in Science Conf., ed. G. Varoquaux, T. Vaught, & J Millman, 11, https://conference.scipy.org/proceedings/scipy2008/SciPy2008_proceedings.pdf
Hill, T. L. 1960, An Introduction to Statistical Thermodynamics (Reading, MA: Addison-Wesley)
Hix, W. R., & Meyer, B. S. 2006, NuPhA, 777, 188
Huang, W., Wang, M., Kondev, F., Audi, G., & Naimi, S. 2021, ChPhC, 45, 030002
Hunter, J. D. 2007, CSE, 9, 90
Katz, M. P., Almgren, A., Sazo, M. B., et al. 2020, in Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis, SC '20 (Piscataway, NJ: IEEE), https://dl.acm.org/doi/abs/10.5555/3433701.3433822
Kluyver, T., Ragan-Kelley, B., Pérez, F., et al. 2016, in Positioning and Power in Academic Publishing: Players, Agents and Agendas, ed. F. Loizides & B. Schmidt (Amsterdam: IOS Press), 87
Krekel, H., Oliveira, B., Pfannschmidt, R., et al. 2004, pytest 7.0, GitHub, https://github.com/pytest-dev/pytest
Kushnir, D., & Katz, B. 2020, MNRAS, 493, 5413
Kushnir, D., Waxman, E., & Chugunov, A. I. 2019, MNRAS, 486, 449
Lam, S. K., Pitrou, A., & Seibert, S. 2015, in Proc. Second Workshop on the LLVM Compiler Infrastructure in HPC, 1
Langanke, K., & Martínez-Pinedo, G. 2000, NuPhA, 673, 481
Lippuner, J., & Roberts, L. F. 2017, ApJS, 233, 18
Longland, R., Martin, D., & José, J. 2014, A&A, 563, A67
Ma, H., Woosley, S. E., Malone, C. M., Almgren, A., & Bell, J. 2013, ApJ, 771, 58
Mestas, P. O., Clayton, P., & Niemeyer, K. E. 2019, JOSS, 4, 1543
Meurer, A., Smith, C. P., Paprocki, M., et al. 2017, PeerJ Comput. Sci., 3, e103
Meyer, B. S. 2018, in AIP Conf. Ser. 1947, 14th Int. Symp. Origin of Matter and Evolution of Galaxies (OMEG 2017) (Melville, NY: AIP), 020016
More, J. J., Garbow, B. S., & Hillstrom, K. E. 1980, User guide for MINPACK-1. [In FORTRAN], ANL-80-74
Müller, E. 1986, A&A, 162, 103
Niemeyer, K. E., & Sung, C.-J. 2011, CoFl, 158, 1439
Oliphant, T. E. 2007, CSE, 9, 10
Pepiot-Desjardins, P., & Pitsch, H. 2008, CoFl, 154, 67
Powell, M. J. D. 1964, CompJ, 7, 155
Powell, M. J. D. 1970, in Numerical Methods for Nonlinear Algebraic Equations, ed. P. Rabinowitz (London: Gordon and Breach)
Rauscher, T. 2003, ApJS, 147, 403
Rauscher, T., Thielemann, F.-K., & Kratz, K.-L. 1997, PhRvC, 56, 1613
Sallaska, A. L., Iliadis, C., Champange, A. E., et al. 2013, ApJS, 207, 18
Schatz, H., Becerril Reyes, A. D., Best, A., et al. 2022, JPhG, 49, 110502
Seitenzahl, I. R., Röpke, F. K., Fink, M., & Pakmor, R. 2010, MNRAS, 407, 2297
Seitenzahl, I. R., Timmes, F. X., Marin-Laflèche, A., et al. 2008, ApJL, 685, L129
Seitenzahl, I. R., Townsley, D. M., Peng, F., & Truran, J. W. 2009, ADNDT, 95, 96
Shampine, L. F., & Reichelt, M. W. 1997, SJSC, 18, 1
Shen, K. J., & Bildsten, L. 2009, ApJ, 699, 1365
Sun, W., Chen, Z., Gou, X., & Ju, Y. 2010, CoFl, 157, 1298
Suzuki, T., Toki, H., & Nomoto, K. 2016, ApJ, 817, 163
Timmes, F. X. 1999, ApJS, 124, 241
Timmes, F. X., Hoffman, R. D., & Woosley, S. E. 2000, ApJS, 129, 377
van der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, CSE, 13, 22
Wallace, R. K., & Woosley, S. E. 1981, ApJS, 45, 389
Wallace, R. K., Woosley, S. E., & Weaver, T. A. 1982, ApJ, 258, 696
Wang, M., Huang, W., Kondev, F., Audi, G., & Naimi, S. 2021, ChPhC, 45, 030003
Weaver, T. A., Zimmerman, G. B., & Woosley, S. E. 1978, ApJ, 225, 1021
Willcox, D. E., & Zingale, M. 2018, JOSS, 3, 588
Xu, Y., Goriely, S., Jorissen, A., Chen, G. L., & Arnould, M. 2013, A&A, 549, A106
Zhang, W., Almgren, A., Beckner, V., et al. 2019, JOSS, 4, 1370
Zhu, L., Ma, Y.-G., Chen, Q., & Han, D.-D. 2016, NatSR, 6, 31882
Zingale, M., Almgren, A. S., Sazo, M. G. B., et al. 2018, JPhCS, 1031, 012024
Zingale, M., Katz, M. P., Bell, J. B., et al. 2019, ApJ, 886, 105
Zingale, M., Katz, M. P., Nonaka, A., & Rasmussen, M. 2022, ApJ, 936, 6