Versatile Integrator for Chemical Evolution

Version 1.4.0.dev0

CONTENTS

Welcome! This is the documentation for **VICE version 1.4.0.dev0**. First time users should familiarize themselves with VICE's API by going through our tutorial, which can be launched automatically by running `python -m vice --tutorial` from a Unix terminal after *installing VICE*. Usage instructions for all of the functions and objects that VICE provides can be found in our *comprehensive API reference*. Details on VICE's implementation and justification thereof can be found in our *science documentation*.

VICE's developers are happy to consult with scientists looking to incorporate it into their research. Email one of our *contributors* or join us on Slack and start collaborating now!

CHAPTER

ONE

INSTALLING VICE

On PyPI we provide pre-compiled binary installers and source distributions for released versions of VICE. To install the latest version, we recommend simply running

```
$ python -m pip install vice [--user]
```

from the command line. Users should add the `--user` flag if they do not have administrative privileges on their current machine; this will install VICE to their `~/.local` directory. To compile and install VICE from source using PyPI, simply specify that you do not want to use the binary using the `--no-binary` flag:

```
$ python -m pip install vice [--user] --no-binary :all:
```

The option `:all:` above tells `pip` to install all of the packages in the current call to `pip install` without binaries; when installing multiple packages, this value can be specified as a comma-separated list. Further details on the `--no-binary` option can be found in the associated documentation for the `pip install` command. Rather than downloading a pre-compiled binary, `pip` will download the source distribution provided on PyPI and compile and install VICE on your machine. `pip` should also conduct an installation using the source distribution automatically in the event that a pre-compiled binary is unavailable for a user's operating system and CPU architecture.

Additionally, previous versions available on PyPI can be installed by simply specifying the version number:

```
$ python -m pip install vice==<version number> [...]
```

Designed for systems with a Unix kernel, VICE does not function within a windows environment. Windows users should therefore install VICE within the Windows Subsystem for Linux (WSL). Provided that the call to `pip` is ran from within WSL, the pre-compiled binary installer from PyPI should install VICE properly. A *manual installation* on a windows machine must also be ran within WSL.

For the current version, we provide pre-compiled binaries for Python versions 3.7-3.10 on computers with an x86_64 CPU architecture running Mac OS and Linux operating systems. We do not provide pre-compiled binaries for CPU architectures other than x86_64. This includes Linux computers with Aarch64 hardware as well as the new ARM64 Apple Silicon computers (i.e. the M1 chip). At present, the developers do not have the resources necessary to pre-compile VICE for these machines, we apologize for the inconvenience. Although VICE can be installed and ran on 32-bit hardware (e.g. i686 CPUs), we strongly discourage running VICE on such machines.

Users who have or would like to modify VICE's source code must conduct a *manual installation*. This also applies to users who would like to install a version of VICE that is still under development (these can be found on various branches of its GitHub repository) as well as for versions of python still under development.

Although it is generally not advised to mix package managers, this is not a concern for VICE. At present, a pre-compiled installation of VICE is not available through conda, though users who typically install their python packages and manage their computing environments with conda can safely conduct their installation of VICE using `pip`. *As noted below*, VICE has no run-time dependencies, meaning that there is no environment that would need solved in the event the installation were conducted using conda. In this case, the package manager would simply see that the list of dependencies is empty, then download the pre-compiled binary and install it.

If you have already installed VICE and would like help getting started, we recommend checking out VICE's tutorial. Further usage guidelines can be found *here*.

Contents

1.1 Dependencies

VICE has no *primary* run-time dependencies; that is, it does not require any external software to run properly. All that is required to run VICE is python itself. There are however a handful of features which are enabled when certain dependencies are satisfied, and we recommend users install them to make use of VICE to its full extent. These *secondary* dependencies are as follows:

1. **dill >= 0.2.0** dill allows VICE to save python functions with its output. This makes it possible to reconstruct simulations from their output.
2. **matplotlib >= 2.0.0** matplotlib is necessary for the `show` function of the `output` object. This is intended to allow users to visually inspect the results of their simulations in `ipython`, a `jupyter notebook`, or something similar without having to plot it themselves. This is included purely for convenience, and is not intended to produce publication-quality figures.
3. **NumPy** VICE's tutorial and example code often make use of NumPy, but the user does not need NumPy to use VICE.

1.1.1 A Note on Implementation

VICE is implemented in ANSI/ISO C and is wrapped using only standard library Python and Cython. It is thus independent of the user's version of Anaconda (or lackthereof). It is NumPy- and pandas-compatible, but neither NumPy- nor pandas-dependent. That is, it will recognize user input from both NumPy and pandas data types such as the NumPy array or the pandas dataframe, but is designed to run independently of them.

1.2 Manual Installation

Users who have modified VICE's source code must compile and install their updated version manually. Otherwise, we recommend simply making use of `pip` as described above. If you have already modified VICE's source code or plan to do so, we encourage you to reach out to one of our *developers* - we'd be happy to consult with you to help VICE meet your needs!

While VICE does not have any primary run-time dependencies, there are a few common compile-time dependencies that must be satisfied to install from source. They are as follows:

1. Python >= 3.7
2. setuptools >= 18.0
3. Make >= 3.81
4. gcc >= 4.6 or clang >= 3.6

On Mac OS X and Linux architectures, it is likely that Make and one of gcc or clang come pre-installed. Users may install with alternative C compilers if they so choose, but VICE is tested with only gcc and clang. While a sizable portion of VICE's source code is written in Cython and requires Cython >= 0.29.0 to compile, this should be handled automatically by setuptools. Nonetheless, it is always an easy option to install it manually via `python -m pip install Cython>=0.29.0`.

Once the build dependencies are satisfied, download the source code using a terminal and change directories into the source tree:

```
$ git clone https://github.com/giganano/VICE.git
$ cd VICE
```

From here, users may change to a specific branch if necessary. For example, VICE's latest development version is on a branch named `development`, and `git checkout development` will take you there. To then compile and install VICE, simply run:

```
$ make
$ python setup.py build install [--user]
```

This will compile the source code under a directory named `build`, and subsequently install to the appropriate `site-packages` directory once completed. Users who do not have administrator's privileges on the system they're conducting the installation should add the `--user` command-line argument, which will conduct a local installation.

Following the installation, running VICE's unit tests (if desired) and cleaning the source tree can be achieved with

```
$ make tests
$ make clean
```

Please note that users installing VICE to multiple versions of python will likely have to run `make clean` between runs of the setup.py file. The command `make tests` runs the unit tests in the current environment's default version of python. If a specific version of python is required, the tests can be ran from within the interpreter itself easily:

```
import vice
vice.test()
```

If you have issues installing or running VICE, please see the section on *Troubleshooting Your Build*. If your installation was successful and you would like help getting started, usage guidelines can be found *here*.

1.2.1 Additional Compile Options

VICE affords users flexibility in specifying how they'd like to compile from source.

1. **Parallelization** Users may spread out the job of compiling VICE across multiple cores via the `[-j N]` command-line argument. For example,

   ```
   $ python setup.py build -j 2 install [--user]
   ```

   will compile all extensions using 2 cores. **Warning**: See *note* below regarding parallel installations with the gcc C compiler.

2. **Suppress verbose output** Users may suppress the printing of compiler commands to the consoler with the `[-q --quiet]` command-line argument. For example, when running

   ```
   $ python setup.py build --quiet install [--user]
   ```

   the only lines printed to the console by the setup.py file will say that specific extensions are being cythonized.

3. **Individual extensions** If VICE's source code has already been compiled and is located in the `build` directory, then the entire code base does not need to be re-compiled every time a small modification is made. The name of the extension, which can be determined via the relative path to the file, is all that is required. For example, the `vice.singlezone` object is linked to VICE's C library in the file `vice/core/singlezone/_singlezone.pyx`, so the name of its extension is `vice.core.singlezone._singlezone`. To recompile this extension only and reinstall with all previously compiled extensions, simply run

   ```
   $ python setup.py build ext=vice.core.singlezone._singlezone install [--user]
   ```

1.2.2 Things to Avoid

1. **Parallelization with the gcc compiler** Users manually installing VICE with the gcc C compiler should omit the `[-j N]` command-line argument from their call to VICE's `setup.py` file (see *Additional Compile Options* above). In practice, the developer's find that gcc is not able to successfully complete compiling VICE across multiple cores. This should be a non-issue for those running Mac OS, as gcc must be installed and clang is the default compiler. For those on Linux, however, gcc is the default.

2. **Simultaneous installations** Users manually installing VICE from source for multiple versions of python should not run the setup.py file in separate terminals simultaneously; this will cause one of the builds to fail. Likewise, users should not run the tests for multiple versions of python simultaneously; this will almost certainly cause a `segmentation fault`.

1.3 Troubleshooting Your Build

The following are a number of issues that can arise when manually installing VICE. If none of these options solve your problem, or if you attempted an installation with `pip` as opposed to installing manually, please open an issue here.

1.3.1 Running the setup.py File Failed

Did you run it for multiple versions of python simultaneously? Alternatively, *did you run a parallelized installation using the gcc C compiler?* If neither is the case, please open an issue here.

1.3.2 Running the Tests Resulted in a Segmentation Fault

Did you run the tests for multiple versions of python simultaneously? If not, please open an issue here.

1.3.3 VICE Isn't Running from the Command Line

If `vice` doesn't run from the terminal after installing, first check that `python3 -m vice` runs; the two have the same functionality. If neither work, then it's likely there was an issue with the installation, and we recommend rerunning the install process, making sure that the instructions are followed as closely as possible. If this still does not work, please open an issue here.

If `python3 -m vice` works, but `vice` does not, then it's likely that that command line entry was copied to a directory not on your `PATH`. The simplest patch for this issue is to create an alias for `vice` mapping it to the longer command. This can be done by adding the following line to your `~/.bash_profile`:

```
alias vice="python3 -m vice"
```

Then either run `source ~/.bash_profile` or restart your terminal for the alias to take effect.

Alternatively, the proper file can simply be copied to any given directory in your computer. If this directory is not on your `PATH`, then your `PATH` must be modified to contain this file's new location. For example:

```
$ cp ./bin/vice ~/.local/bin
```

This will place the command line entry in the `~/.local/bin/` directory, which can be permanently added to your path by adding

```
export PATH=$HOME/.local/bin:$PATH
```

to your `~/.bash_profile`. As with the alias solution, this will require either running `source ~/.bash_profile` or restarting your terminal to take effect.

Note: If you have installed VICE with the `--user` option, it is likely that VICE has automatically made the above modification to your `PATH`, and that either running `source ~/.bash_profile` or restarting your terminal is all that is required after copying the file to `~/.local/bin`. If you have copied the file to a different directory, VICE will not have added that file to your `PATH`.

More information on modifying your PATH can be found here.

If this does not fix the issue, please open an issue here.

An alternative workaround to this issue is to create an alias for `vice` by adding the following line to

1.3.4 Compiler Failure

This is usually an indication that the build should not be ran on multiple cores, which *is usually the case with the gcc C compiler*. First run `make clean`, and subsequently `make`. Then replace your previous command to run the setup.py file with:

```
$ python setup.py build install [--user] [--quiet]
```

If you were not installing VICE on multiple cores to begin with, try installing without the `build` directive:

```
$ python setup.py install [--user] [--quiet]
```

If neither of these recommendations fix your problem, please open an issue here.

1.4 Uninstalling VICE

If you have installed VICE from PyPI, it can be uninstalled from the terminal via `pip uninstall vice`. When prompted, simply confirm that you would like the files removed. If you have downloaded VICE's supplementary data for use with the `milkyway` object, it is recommended that you remove these files first by running

```
import vice
vice.toolkit.hydrodisk.data._h277_remove()
```

before the `pip uninstall vice` command.

If you have installed from source, uninstalling requires a couple of steps. First, you must find the path to the directory that it was installed to. This can be done by launching python and running the following two lines:

```
import vice
print(vice.__path__)
```

Note that there are *four* underscores in total: two each before and after `path`. This will print a single-element list containing a string denoting the name of the directory holding VICE's compiled extensions, of the format `/path/to/install/dir/vice`. Change into this directory, and remove the VICE tree:

```
$ cd /path/to/install/dir/
$ rm -rf vice/
```

Then, check the remaining contents for an `egg`. This will likely be of the format `vice-<version number>.egg-info`. Remove this directory as well:

```
$ rm -rf vice-<version number>.egg-info
```

Finally, the command line entry must be removed. The full path to this script can be found with the `which` command in the terminal:

```
$ which vice
```

This will print the full path in the format `/path/to/cmdline/entry/vice`. Pass it to the `rm` command as well:

```
$ rm -f /path/to/cmdline/entry/vice
```

If this process completed without any errors, then VICE was successfully uninstalled. To double-check, re-running `which vice` should now print nothing, and attempting to import VICE into python should result in a `ModuleNotFoundError`.

CHAPTER

TWO

GETTING STARTED

VICE's developers are happy to consult with scientists looking to incorporate it into their research. Email one of our *contributors* or join us on Slack and start collaborating now!

Users should also be aware of our *comprehensive API reference*, where they can find instructions on how to use all of the functions and objects that VICE provides. Details on VICE's implementation and justification thereof can be found in our *science documentation*.

2.1 Tutorial

Under `examples` in VICE's GitHub repository is the tutorial, a `jupyter notebook` intended to provide first-time users with a primer on how to use all of VICE's features. *After installation*, this jupyter notebook can be downloaded and launched automatically by running `vice --tutorial` (or, equivalently, `python -m vice --tutorial`) from a Unix terminal. Whenever this command is ran, VICE will re-download this notebook from GitHub, falling back on a previously downloaded version if there is no internet connection (or if it cannot be downloaded for some other reason). Alternatively, if installing from source, it can be launched via `make tutorial` in the root directory. This jupyter notebook can also be obtained by cloning the git repository.

2.2 Example Code

We provide example scripts in VICE's GitHub repository under `examples`, alongside the tutorial.

2.3 Accessing Documentation

After installing VICE, the documentation can be launched in a browser window via the `vice --docs` command line entry. If this feature does not work after installing VICE, troubleshooting can be found here. Documentation can also be found in the docstrings embedded in the code, and in the GitHub repository.

2.4 From the Command Line

VICE allows simple one-zone models to be ran directly from the command line. For instructions on how to use this functionality, run `vice --help` in a terminal from any directory (with the exception of VICE's source directory). If the `vice` command-line entry isn't working, it's possible the variant `python3 -m vice` is required. Further troubleshooting can be found here.

Note: VICE's functionality is severely limited when ran from the command line in comparison to its full python capabilities.

CHAPTER

THREE

SCIENCE DOCUMENTATION

Disclaimer: This section of VICE's documentation is not intended to provide users with a holistic scientific background in galactic chemical evolution models. Rather, some knowledge of the science at hand is assumed. Where possible, we have linked additional references to peer reviewed journal articles with more in-depth scientific justification and disucssion.

In this documentation we adopt the notation where a lower-case m implicitly represents the mass ratio of the star to the sun, a unitless mass measurement. When relevant, we refer to the mass of a star with units with an upper-case M. In a similar fashion, l and u refer to the lower and upper mass limits of star formation, respectively.

All nucleosynthetic yields in chemical evolution models provided by VICE are defined as *fractional net yields*. That is, they quantify the mass of stellar material that is processed into a given element and subsequently ejected to the ISM *in units of the star or stellar population's initial mass*, and they do *not* quantify the mass fraction of nucleosynthetic material that a star or stellar population was born with. We denote these values with a lower-case y with subscripts and superscripts denoting the element and the enrichment channel, respectively.

A capital Z refers always to the metallicity by mass:

$$Z \equiv \frac{M_x}{M}$$

Where M_x refers to the mass of some element x and M to the mass of either the interstellar gas or a star.

The logarithmic abundance measurement $[X/H]$ is defined by:

$$[X/H] \equiv \log_{10}\left(\frac{Z_x}{Z_x^\odot}\right)$$

This approximation assumes hydrogen mass fractions are similar to the sun always. Relaxing this assumption would require subtracting the term $\log_{10}(X/X_\odot)$ where X is the hydrogen mass fraction. However, this is generally a negligible correction as hydrogen mass fractions vary only a little, especially on a logarithmic scale ($\lesssim 0.05$ dex), and neither the types of models that VICE provides nor observationally derived abundances can claim this level of precision anyway. The logarithmic abundance ratios $[X/Y]$ follow accordingly:

$$[X/Y] = [X/H] - [Y/H] = \log_{10}\left(\frac{Z_x}{Z_x^\odot}\right) - \log_{10}\left(\frac{Z_y}{Z_y^\odot}\right)$$

The symbols $\odot$ and τ refer to the sun and a timescale, respectively, and the term "zone models" refers to both one-zone and multi-zone models in the general sense.

3.1 Background

3.1.1 Galactic Chemical Evolution

Galactic chemical evolution and galactic archaeology study the connection between a galaxy's evolution and the chemical compositions of its gas and stars, with the latter having somewhat special emphasis on the Milky Way. Big Bang Nucleosynthesis produced only hydrogen, helium, and trace amounts of lithium, the three lightest elements on the periodic table. To first order, everything else was produced via nuclear fusion in supernovae and through various channels of stellar evolution, the yields of which are dictated by nuclear physics. The abundances of different nuclei within stars therefore has physical information on the number of nucleosynthetic events and thus the number of stars that came before it. For more theoretical background on galactic chemical evolution, see sections 1 and 2 and the citations therein of Johnson & Weinberg (2020).

3.1.2 The Singlezone Approximation

The singlezone approximation (also known as the onezone approximation, onezone models, box models, or variations thereof), refers to the assumption of instantaneous diffusion of newly produced metals in interstellar gas. This assumptions mandates that these nuclei be uniformly distributed at all times. By deliberately sacrificing all phase space information, the equations of these models reduce to a system of couple integro-differential equations of mass with time. While these equations only allow analytic solutions under further *mathematical* approximations, they can be easily integrated numerically.

VICE includes features for running numerical simulations of singlezone models in the `singlezone` class. In this documentation, we detail the analytic motivation and numerical approximations implemented in VICE in handling these simulations.

3.1.3 The Multizone Approximation

The multizone approximation refers to the extension of singlezone models to take into account phase space information with any degree of sophistication. By allowing singlezone models to evolve in parallel, and to mix stars and gas amongst them under some prescription, information on the spatial structure of the model galaxy can be added back to the simulation.

VICE includes features for running numerical simulations of multizone models in the `multizone` class. In this documentation, we detail the analytic motivation and numerical approximations implemented in VICE in handling these simulations.

3.2 Implementation

3.2.1 Motivation

VICE is designed in such a manner that as few assumptions as possible are made by the software itself. In this manner, the power the user has over the parameters of their simulations is maximized. With this motivation, any quantities that may vary are allowed to do so under user-constructed functions in Python. The only assumption VICE's model adopts is physical plausibility.

3.2.2 Numerical Approach

Because VICE is built to handle singlezone and multizone simulations, numerics are not the dominant source of error, but rather in the model itself. The assumption of instantaneous diffusion of newly produced metals introduces an error that which is larger than even modest numerical errors to the equations presented in this documentation.

For this reason, VICE is implemented with a Forward Euler timestep solution, and its errors are not dominated by numerics. Furthermore, quantization of the timesteps allows the quantization of the episodes of star formation with no further assumptions. At several instances in this documentation, this will simplify the equations considerably. Adopting a user-specified timestep size, this also makes it the computationally cheapest solution by not introducing intermediate timesteps. In this manner, VICE is able to achieve a high degree of generality while retaining powerful computing speeds.

3.2.3 Minimization of Dependencies

VICE is implememented entirely in ANSI/ISO C and standard library Python and Cython. For this reason, it is highly portable across platforms and is independent of the user's version of Anaconda (or lackthereof). However, VICE is not wrapped for installation in a Windows environment, and the developers do not provide support for non-POSIX operating systems. We therefore recommend that Windows users install and run VICE entirely within the Windows Subsystem for Linux.

3.2.4 Timed Runs

Due to the Forward Euler implementation and the requirement to calculate enrichment from previous episodes of star formation, we expect the integration time to scale with the square of the number of timesteps (i.e. $T \propto (T_{\text{end}}/\Delta t)^2$). VICE also treats each element independently and equally; the equations of enrichment are evaluated for an arbitrary element x. The integration time should thus scale linearly with the number of elements N.

Because VICE was implemented with the scientific motivation of studying the enrichment of oxygen, iron, and strontium under starburst evolutionary scenarios (Johnson & Weinberg 2020[1]), the first integrations were ran with these three elements. With timesteps of Δt = 1 Myr, each simulation finished in 20.4 seconds on a system with a processing speed of 2.7 GHz. With these proportionalities and this calibration, we expect the following scaling relation to describe the time per integration of the `singlezone` object as a function of the number of elements N, the end time T_{end}, and the timestep size Δt:

$$T = \left(\frac{\text{Processor Speed}}{2.7\text{ GHz}}\right)^{-1} \left(\frac{T_{\text{end}}/\Delta t}{10^4}\right)^2 N(6.8\text{ seconds})$$

Because 1 Myr is a relatively fine timestep, most integrations will typically not take this long. The default timestep size of 10 Myr is expected to finish in 68 milliseconds per element.

Here we plot the integration time for 5, 10, 15, 20, and 25 elements with timesteps ranging from 500 kyr to 10 Myr in comparison to the expected scaling relation in one-zone models with the `singlezone` object. For small Δt, the scaling relation describes the integration time with sufficient accuracy, although slightly underpredicts the integration time when the number of elements is large. This also underpredicts the integration time for coarse timestepping; this is because this scaling relation does not take into account write-out time. For large Δt, the `singlezone` object is not algorithm limited but write-out limited. Write out time may also be a potential reason that the integration time is mildly underpredicted for small Δt and high N.

[1] Johnson & Weinberg (2020), MNRAS, 498, 1364

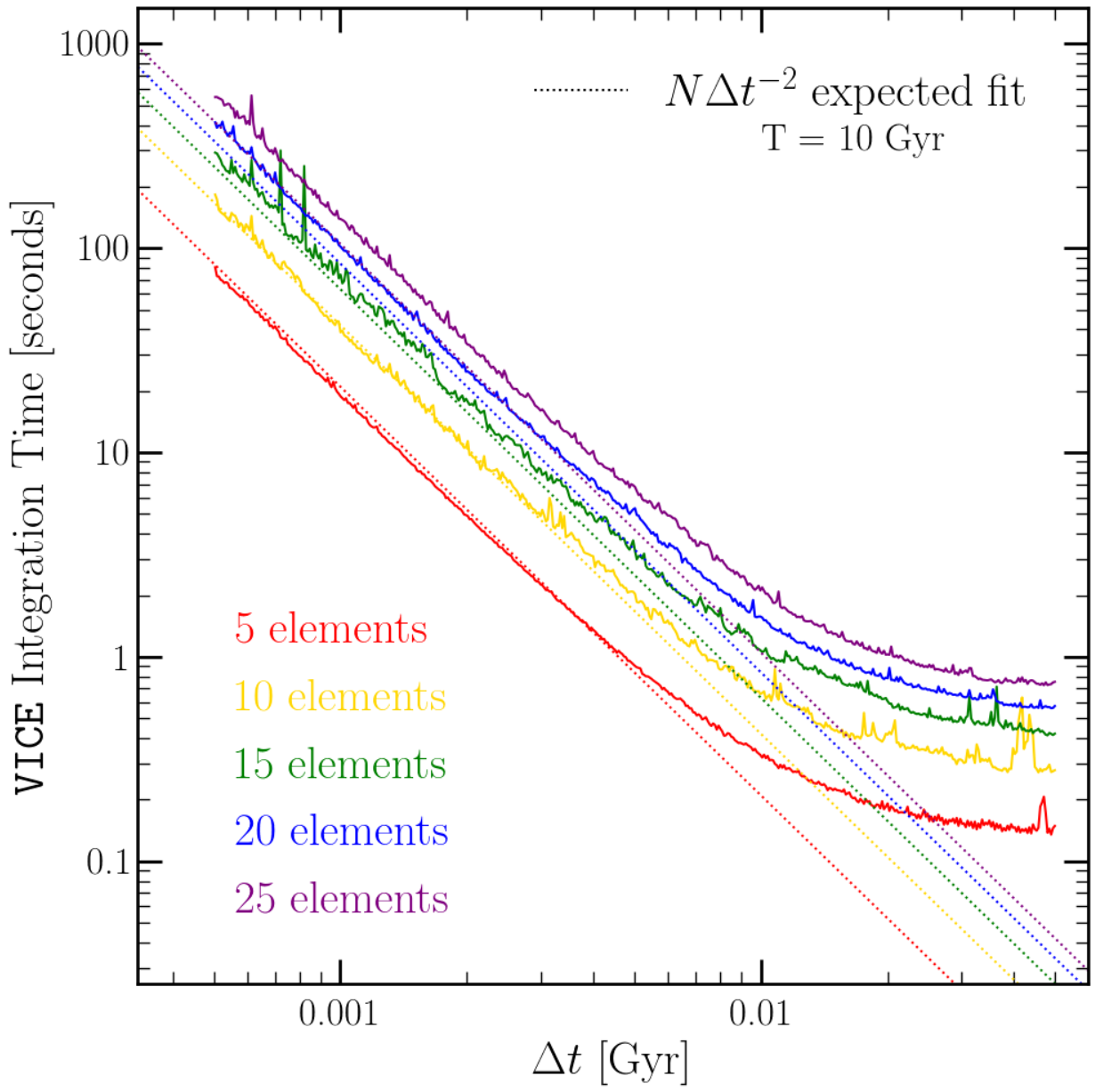

Fig. 1: Timed runs with N = 5, 10, 15, 20, and 25 elements with timesteps ranging from 500 kyr to 10 Myr (solid lines) with an ending time of $T_{\rm end}$ = 10 Gyr. The color-coded dotted lines show the $N\Delta t^{-2}$ expected scaling relation. The fit does well for small Δt, but underpredicts the integration time for coarse timestepping; this is due to the transition from an algorithm limited simulation to a write out limited simulation. The scaling relation also slightly underpredicts the integration time for high N simulations.

3.3 Single Stellar Populations

As discussed in our section on *implementation*, VICE's simulations are implemented with a Forward Euler timestep solution, an approximation made possible by numerics not being the dominant source of error. The quantization of timesteps necessitates the quantization of the episodes of star formation. This allows VICE to model enrichment in both singlezone and multizone models by using summations over a sample of discretized stellar populations.

For this reason, we implement a treatment of two quantities particularly useful in the mass evolution of single stellar populations: the *cumulative return fraction* (CRF) and the *main sequence mass fraction* (MSMF). The *CRF* represents the fraction of a single stellar population's mass that is returned to the interstellar medium as gas. The *MSMF* is the fraction of its mass that is still in the form of main sequence stars. These quantities are of particular use in calculating the rate of mass recycling and the rate of enrichment from asymptotic giant branch stars.

3.3.1 Stellar Lifetimes

The simplest description of the relationship between a star's mass and its lifetime (i.e. the *mass-lifetime relationship*, hereafter MLR) is a single power law:

$$\tau_{\text{MS}} = \tau_{\odot} m^{-\alpha}$$

where $\tau_{\odot}$ is the sun's main sequence lifetime and α is the power law index of the MLR. The constants `SOLAR_LIFETIME` and `MASS_LIFETIME_PLAW_INDEX`, both declared in `vice/src/ssp.h`, assign the values of $\tau_{\odot}$ = 10 Gyr and α = 3.5, respectively. Motivated by a popular exercise in undergradaute astronomy courses, this form follows from an assumed single-power law relating the mass and luminosity of stars: $L \sim M^{1+\alpha}$. Because the lifetime of a star scales with its luminosity per unit mass, the power law then follows trivially: $\tau \sim M/L \sim M/M^{1+\alpha} \sim M^{-\alpha}$.

VICE generalizes this form to describe the *total lifetime* of a star of mass m by simply amplifying the main sequence lifetime by a factor of $1 + p_{\text{MS}}$:

$$\tau_{\text{total}} = (1 + p_{\text{MS}})\tau_{\odot} m^{-\alpha}$$

where p_{MS} is the ratio of a star post main sequence lifetime to its main sequence lifetime (e.g. if the post main sequence lifetime is 10% of the main sequence lifetime, then p_{MS} = 0.1). As a consequence, this quantity describes the time interval between a star's formation and when it produces a remnant.

By interpreting τ_{total} as lookback time, VICE solves for the mass of "remnant-producing" stars from a stellar population of known age:

$$m_{\text{postMS}} = \left(\frac{\tau_{\text{lookback}}}{(1 + p_{\text{MS}})\tau_{\odot}}\right)^{-1/\alpha}$$

Since these are the stars that are at the ends of the lifetimes, VICE adopts m_{postMS} as written here as the mass of AGB stars enriching the interstellar medium at a given timestep. This equation also allows the solution of the *main sequence turnoff mass* by simply taking p_{MS} = 0.

The scaling of $\tau_{\text{MS}} \sim m^{-3.5}$ fails for higher mass stars ($\gtrsim 4M_{\odot}$). However, these stars generally have lifetimes that are short compared to the relevant timescales of galactic chemical evolution ($\lesssim$ 100 Myr compared to $\sim$ few Gyr). Regardless of its accuracy for low mass stars ($\lesssim 0.5M_{\odot}$), their lifetimes are considerably longer than the age of the universe anyway, and VICE does not support simulations on timescales longer than 15 Gyr. Consequently, this approximation generally suffices for galactic chemical evolution models.

The important exception to this rule is that an accurate MLR for $4 \lesssim M/M_{\odot} \lesssim 8$ stars is necessary when considering elements with significant nucleosynthetic yields from these stars (e.g. nitrogen, Johnson et al. 2021[1]). Prior to version 1.3.0, VICE implemented this single power law relationship only. In subsequent versions, a handful of additional forms are available to fill this need:

[1] Johnson et al. (2021), in prep

- Larson (1974)[2] **(default in versions >= 1.3.0)**

  This form is a metallicity-independent parabola in $\log\tau - \log m$ space describing the main sequence lifetimes only:

  $$\log_{10}\tau = \alpha + \beta\log_{10} m + \gamma(\log_{10} m)^2$$

  where the original values of α = 1.02, β = -3.57, and γ = 0.90 were derived from a fit to the compilation of evolutionary lifetimes presented in Tinsley (1972)[3]. VICE however adopts the updated values of α = 1.0, β = -3.42, and γ = 0.88 from Kobayashi (2004)[4] and David, Forman & Jones (1990)[5]. In detail, the value of α directly quantifies the log of the main sequence lifetime of the sun $\log_{10}\tau_\odot$ in whatever units τ is in (α = 1.0 for $\tau_\odot$ = 10 Gyr; α = 10.0 for $\tau_\odot = 10^{10}$ yr).

  Solutions to the inverse function (i.e. mass as a function of lifetime) follow directly from the quadratic formula: $\log_{10} m = (-\beta \pm \sqrt{\beta^2 - 4\gamma(\alpha - \log_{10}\tau)})/(2\gamma)$. The correct solution comes when choosing subtraction in the numerator as this corresponds to increasing lifetimes with decreasing stellar mass.

  This is the default MLR for VICE version 1.3.0 and later.

- Maeder & Meynet (1989)[6]

  This form is a metallicity-independent broken power law quantifying only the main sequence lifetimes:

  $$\tau = \begin{cases} 10^{\alpha\log_{10} m + \beta} \ (m < 60 M_\odot) \\ 1.2m^{-1.85} + 3 \text{ Myr} \ (m \geq 60 M_\odot) \end{cases}$$

  where the values of α and β are given by:

Mass Range	α	β
$m \leq 1.3$	-0.6545	1
$1.3 < m \leq 3$	-3.7	1.35
$3 < m \leq 7$	-2.51	0.77
$7 < m \leq 15$	-1.78	0.17
$15 < m \leq 60$	-0.86	-0.94

  In detail, the values of β shift linearly depending on the choice of units for τ; those listed here are appropriate for τ in Gyr. For a shift to τ in yr, all values should increase by 9 (e.g. β = 10.35 for masses between 1.3 and 3 $M_\odot$).

  Though this form was originally published in Maeder & Meynet (1989), the exact form as written here was taken from Romano et al. (2005)[7]. While analytic solutions to the inverse function (i.e. mass as a function of lifetime) are possible, VICE takes a numerical approach, implementing a recursive version of the bisection algorithm described in chapter 9 of Press, Teukolsky, Vetterling & Flannery (2007)[8].

- Padovani & Matteucci (1993)[9]

  This form is a metallicity-independent curve describing the main-sequence lifetime only:

  $$\log_{10}\tau = \frac{\alpha - \sqrt{\beta - \gamma\,(\eta - \log_{10} m)}}{\mu}$$

[2] Larson (1974), MNRAS, 166, 585
[3] Tinsley (1972), A&A, 20, 383
[4] Kobayashi (2004), MNRAS, 347, 740
[5] David, Forman & Jones (1990), ApJ, 359, 29
[6] Maeder & Meynet (1989), A&A, 210, 155
[7] Romano et al. (2005), A&A, 430, 491
[8] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press
[9] Padovani & Matteucci (1993), ApJ, 416, 26

The values of the coefficients are given by:

Coefficient	Value
α	0.334
β	1.790
γ	0.2232
η	7.764
μ	0.1116

These values are appropriate for τ in units of Gyr. Solutions to the inverse function (i.e. mass as a function of lifetime) follow analytically. Though this form was originally published in Padovani & Matteucci (1993), the form as written here was taken from Romano et al. (2005).

- Kodama & Arimoto (1997)[10]

  Using the stellar evolution code presented in Iwamoto & Saio (1999)[11], Kodama & Arimoto (1997) tabulate the *total* lifetimes (i.e. including post main sequence evolution) of stars as a function of both initial mass and metallicity. VICE stores internal data at 41 initial masses and 9 metallicities, using 2-dimensional linear interpolation to approximate a smooth function based on these discrete points.

  Because of the necessary interpolation, solutions to the inverse function (i.e. mass as a function of lifetime and metallicity) follow numerically, for which VICE implements a recursive version of the bisection algorithm described in chapter 9 of Press, Teukolsky, Vetterling & Flannery (2007).

- Hurley, Pols & Tout (2000)[12]

  This is a metallicity-dependent characterization of the main sequence lifetimes of stars given by:

  $$\tau = \max(\mu, x) t_{\text{BGB}}$$

  where t_{BGB} is the time required for a star to reach the base of the giant branch on the Hertzsprung-Russell diagram:

  $$t_{\text{BGB}} = \frac{a_1 + a_2 m^4 + a_3 m^{5.5} + m^7}{a_4 m^2 + a_5 m^7}$$

  The coefficients a_n vary with metallicity according to:

  $$a_n = \alpha_n + \beta_n \zeta + \gamma_n \zeta^2 + \eta_n \zeta^3$$

  VICE stores the values of α, β, γ, and η for the coefficients a_n as internal data, and the quantity ζ is related to the metallicity by mass Z by $\zeta = \log_{10}(Z/0.02)$. The value of 0.02 corresponds to the metallicity of the sun; although there has been some evolution in the accepted value of $Z_\odot$, VICE takes this value of 0.02 *always* when calculating lifetimes according to the Hurley, Pols & Tout (2000) parameterization regardless of the user's setting in a chemical evolution model.

  The coefficients μ and x are given by:

  $$\mu = \max\left(0.5, 1.0 - 0.01\max\left(\frac{a_6}{m^{a_7}}, a_8 + \frac{a_9}{m^{a_{10}}}\right)\right)$$

  $$x = \max\left(0.95, \min\left[0.95 - 0.03\left(\zeta + 0.30103\right)\right]\right)$$

  Solutions to the inverse function (i.e. mass as a function of lifetime and metallicity) are numerical, for which VICE implements a recursive version of the bisection algorithm described in chapter 9 of Press, Teukolsky, Vetterling & Flannery (2007).

[10] Kodama & Arimoto (1997), A&A, 320, 41
[11] Iwamoto & Saio (1999), ApJ, 521, 297
[12] Hurley, Pols & Tout (2000), MNRAS, 315, 543

- Vincenzo et al. (2016)[13]

  This form characterizes the total lifetimes of stars (i.e. including the post main sequence evolution) as a function of stellar mass and metallicity according to:

  $$\tau = A\exp(Bm^{-C})$$

  where the coefficients A, B, and C depend on metallicity. VICE stores their values sampled at 299 values of the metallicity Z as internal data, interpolating linearly between them to approximate smooth functions out of the discrete points. With their values known at a given metallicity, the inverse function (i.e. mass as a function of lifetime) follows analytically from the above equation.

  Vincenzo et al. (2016) determined the values of these coefficients by using isochrones computed using the PARSEC stellar evolution code (Bressan et al. 2012[14]; Tang et al. 2014[15]; Chen et al. 2015[16]) in combination with a one-zone chemical evolution model parameterized to reproduce the color-magnitude diagram of the Sculptor dwarf galaxy.

Here we plot stellar lifetime as a function of progenitor mass according to each of these forms along with the single power law described above; its failure at high masses compared to the other, more sophisticated parameterizations is quite clear. VICE affords users the ability to evaluate these functions using the `vice.mlr` module (e.g. `vice.mlr.hpt2000` correspond to the Hurley, Pols & Tout (2000) form, and `vice.mlr.ka1997` to the Kodama & Arimoto (1997) form). The form to be adopted in all chemical evolution models and single stellar population calculations is assigned via a global setting stored at `vice.mlr.setting`.

Of these parameterizations of the MLR, the following take into account the metallicity dependence:

- Vincenzo et al. (2016)
- Hurley, Pols & Tout (2000)
- Kodama & Arimoto (1997)

The following require numerical solutions for the inverse function (i.e. stellar mass as a function of lifetime):

- Hurley, Pols & Tout (2000)
- Kodama & Arimoto (1997)
- Maeder & Meynet (1989)

The following quantify the total lifetimes *a priori*, making calculations of purely main sequence lifetimes unavailable:

- Vincenzo et al. (2016)
- Kodama & Arimoto (1997)

Except where measurements of the total lifetime is available, VICE always implements the simplest assumption of allowing the user to specify the parameter p_{MS} (see above), and the total lifetime then follows trivially via:

$$\tau_{\text{total}} = (1 + p_{\text{MS}})\tau_{\text{MS}}$$

Relevant source code:

- `vice/core/mlr.py`
- `vice/src/ssp.h`
- `vice/src/ssp/mlr.c`
- `vice/src/ssp/mlr/powerlaw.c`

[13] Vincenzo et al. (2016), MNRAS, 460, 2238
[14] Bressan et al. (2012), MNRAS, 427, 127
[15] Tang et al. (2014), MNRAS, 445, 4287
[16] Chen et al. (2015), MNRAS, 452, 1068

- `vice/src/ssp/mlr/vincenzo2016.c`
- `vice/src/ssp/mlr/hpt2000.c`
- `vice/src/ssp/mlr/ka1997.c`
- `vice/src/ssp/mlr/pm1993.c`
- `vice/src/ssp/mlr/mm1989.c`
- `vice/src/ssp/mlr/larson1974.c`
- `vice/src/ssp/mlr/root.c`

3.3.2 The Cumulative Return Fraction

The cumulative return fraction is defined as the mass fraction of a single stellar population that is returned back to the interstellar medium (ISM) as gas. When dying stars produce their remnants, whatever material that does not end up in the remnant is returned to the ISM. This quantity can be calculated from an initial-final mass relation and an adopted stellar initial mass function (IMF). In short, the cumulative return fraction can be stated mathematically as "ejected material from dead stars in units of total initial amount of material." Its analytic form is therefore given by:

$$r(t) = \int_{m_{\text{to}}(t)}^{u} (m - m_{\text{rem}}) \frac{dN}{dm} dm \left[\int_{l}^{u} M \frac{dN}{dm} dm\right]^{-1}$$

The current version of VICE employs the initial-final remnant mass relation of Kalirai et al. (2008)[17]:

$$m_{\text{rem}}(m) = \begin{cases} 1.44 \ (m \geq 8) \\ 0.394 + 0.109m \ (m < 8) \end{cases}$$

For a power-law IMF $dN/dm \sim m^{-\alpha}$, the numerator of $r(t)$ is thus given by:

$$\int_{m_{\text{to}}(t)}^{u} (m - m_{\text{rem}}(m)) \frac{dN}{dm} dm = \frac{1}{2-\alpha} m^{2-\alpha} \Bigg|_{m_{\text{to}}(t)}^{u} - \frac{1.44}{1-\alpha} m^{1-\alpha} \Bigg|_{m_{\text{to}}(t)}^{u}$$

for $m_{\text{to}}(t) \geq 8$, and

$$\int_{m_{\text{to}}(t)}^{u} (m - m_{\text{rem}}(m)) \frac{dN}{dm} dm = \frac{1.44}{1-\alpha} m^{1-\alpha} \Bigg|_{8}^{u} + \left[\frac{0.394}{1-\alpha} m^{1-\alpha} + \frac{0.109}{2-\alpha} m^{2-\alpha}\right]_{m_{\text{to}}(t)}^{8}$$

for $m_{\text{to}}(t) < 8$.

This solution is analytic. For piecewise IMFs, this becomes a summation over the relevant mass ranges of the IMF, and each term has the exact same form. The normalization of the IMF is irrelvant here, because the same normalization will appear in the denominator.

The denominator has a simpler analytic form:

$$\int_{l}^{u} m \frac{dN}{dm} dm = \frac{1}{2-\alpha} m^{2-\alpha} \Bigg|_{l}^{u}$$

Here we plot r as a function of the stellar population's age assuming the mass-lifetime relation of Hurley, Pols & Tout (2000)[18] (see discussion *here*). Weinberg, Andrews, and Freudenburg (2017)[19] adopted instantaneous recycling, whereby a fraction of the stellar population's mass r_{inst} is returned *instantaneously* in the interest of an analytic approach

[17] Kalirai et al. (2008), ApJ, 676, 594
[18] Hurley, Pols & Tout (2000), MNRAS, 315, 543
[19] Weinberg, Andrews & Freudenburg (2017), ApJ, 837, 183

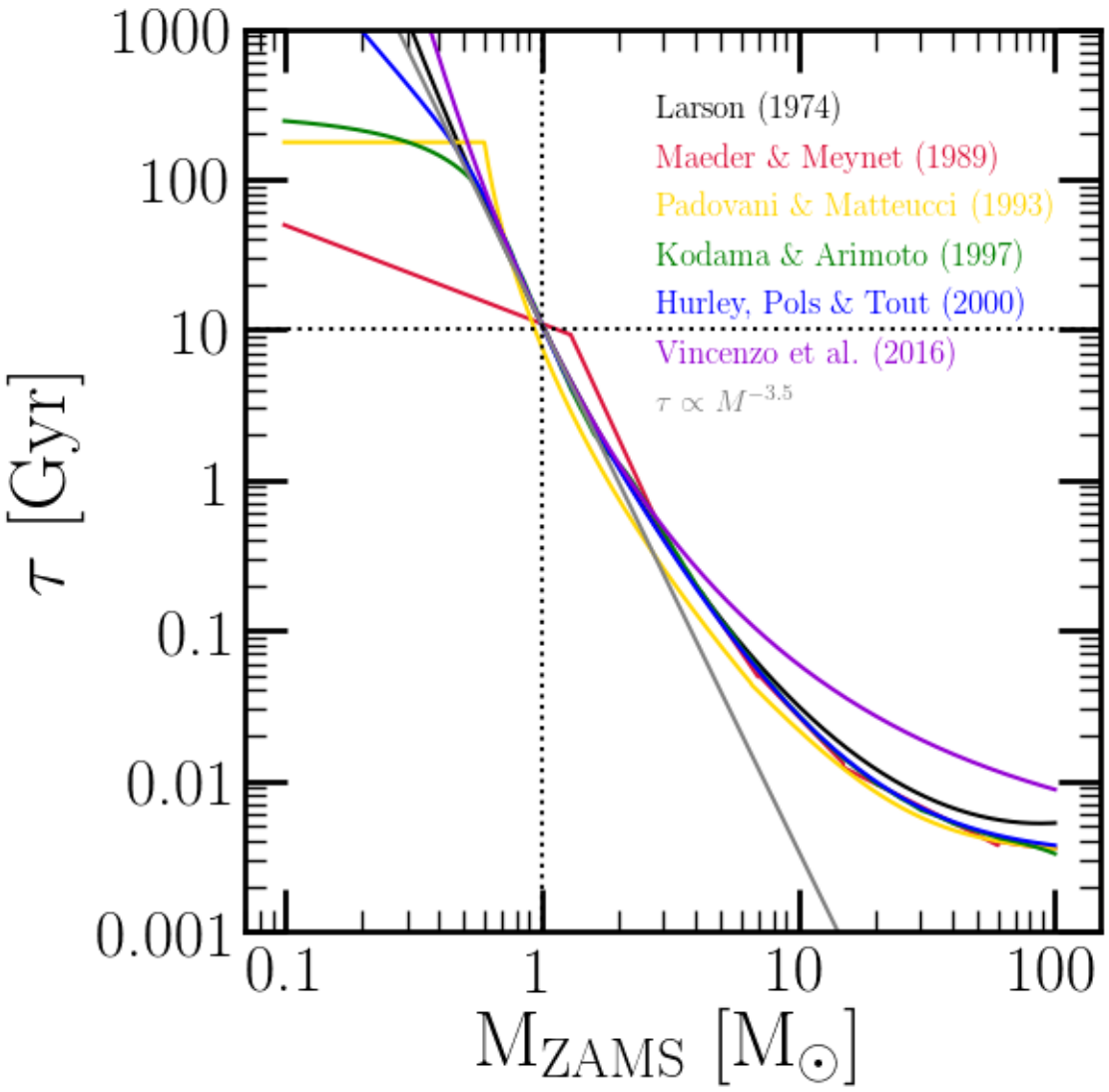

Fig. 2: The total lifetimes of stars as a function of initial mass for the various forms recognized by VICE: Larson (1974; black), Maeder & Meynet (1989; red), Padovani & Matteucci (1993; yellow), Kodama & Arimoto (1997; green), Hurley, Pols & Tout (2000; blue), Vincenzo et al. (2016; purple), and the single power-law relationship (grey; see above). For the forms which take into account the metallicity-dependence of stellar lifetimes (Kodama & Aritmoto 1997; Hurley, Pols & Tout 2000; Vincenzo et al. 2016), the relationship is shown at solar metallicity. Those that compute main sequence lifetimes only as opposed to total lifetimes (all except Kodama & Arimoto 1997 and Vincenzo et al. 2016) are calculated with $p_{\rm MS}$ = 0.1 (i.e. the inferred *total* lifetimes are plotted for all forms).

to singlezone models. They find that $r_{\text{inst}} = 0.4$ and $r_{\text{inst}} = 0.2$ is an adequate approximation for Kroupa[20] and Salpeter[21] IMFs. This reduces the more sophisticated formulation implemented here to:

$$r(t) \approx \begin{cases} r_{\text{inst}} \ (t = 0) \\ 0 \ (t > 0) \end{cases}$$

In reality, the rate of mass return from a stellar population of mass M_* is given by $\dot{r}M_*$, but in implementation, the quantization of timesteps allows each timestep to represent a single stellar population which will eject mass $M_* dr$ in a time interval dt. For that reason, VICE is implemented with a calculation of $r(t)$ rather than $\dot{r}$.

In simulations, VICE allows users the choice between the time-dependent formulation of $r(t)$ derived here and the instantaneous approximation of Weinberg, Andrews, and Freudenburg (2017) by specifying a preferred value of r_{inst}, which allows any fraction between 0 and 1.

In calculations of $r(t)$ with the built-in Kroupa and Salpeter IMFs, the analytic solution is calculated. In the case of a user-customized IMF, VICE solves the equation numerically using quadrature with the methods described in chapter 4 of Press, Teukolsky, Vetterling & Flannery (2007)[22].

Note: The approximation of $h(t) \approx 1 - r(t)$ where h is the *main sequence mass fraction* fails at the $\sim 5 - 10\%$ level. See our discussion of this point here.

Relevant source code:

- `vice/src/ssp/crf.c`
- `vice/src/yields/integral.c`

3.3.3 The Main Sequence Mass Fraction

The main sequence mass fraction, as the name suggests, is the fraction of a single stellar population's initial mass that is still in the form of main sequence stars. Because this calculation does not concern evolved stars, neither a model for the post main sequence lifetime nor an initial-final remnant mass relation is needed; it is thus considerably simpler than the *cumulative return fraction*. This quantity is instead specified entirely by the IMF and the mass-lifetime relation.

It's analytic form is given by:

$$h(t) = \int_l^{m_{\text{to}}(t)} m \frac{dN}{dm} dm \left[\int_l^u m \frac{dN}{dm} dm\right]^{-1}$$

which for a power-law IMF $dN/dm \sim m^{-\alpha}$ becomes

$$h(t) = \left[\frac{1}{2-\alpha} m^{2-\alpha} \Big|_l^{m_{\text{to}}(t)}\right] \left[\frac{1}{2-\alpha} m^{2-\alpha} \Big|_l^u\right]^{-1}$$

It may be tempting to cancel the factor of $1/(2-\alpha)$, but more careful consideration must be taken for piece-wise IMFs like Kroupa[26]:

$$h(t) = \left[\sum_i \frac{1}{2-\alpha_i} m^{2-\alpha_i}\right]_l^{m_{\text{to}}(t)} \left(\left[\sum_i \frac{1}{2-\alpha_i} m^{2-\alpha_i}\right]_l^u\right)^{-1}$$

[20] Kroupa (2001), MNRAS, 322, 231
[21] Salpeter (1955), ApJ, 121, 161
[22] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press
[23] Kroupa (2001), MNRAS, 322, 231
[24] Salpeter (1955), ApJ, 121, 161
[25] Hurley, Pols & Tout (2000), MNRAS, 315, 543
[26] Kroupa (2001), MNRAS, 322, 231

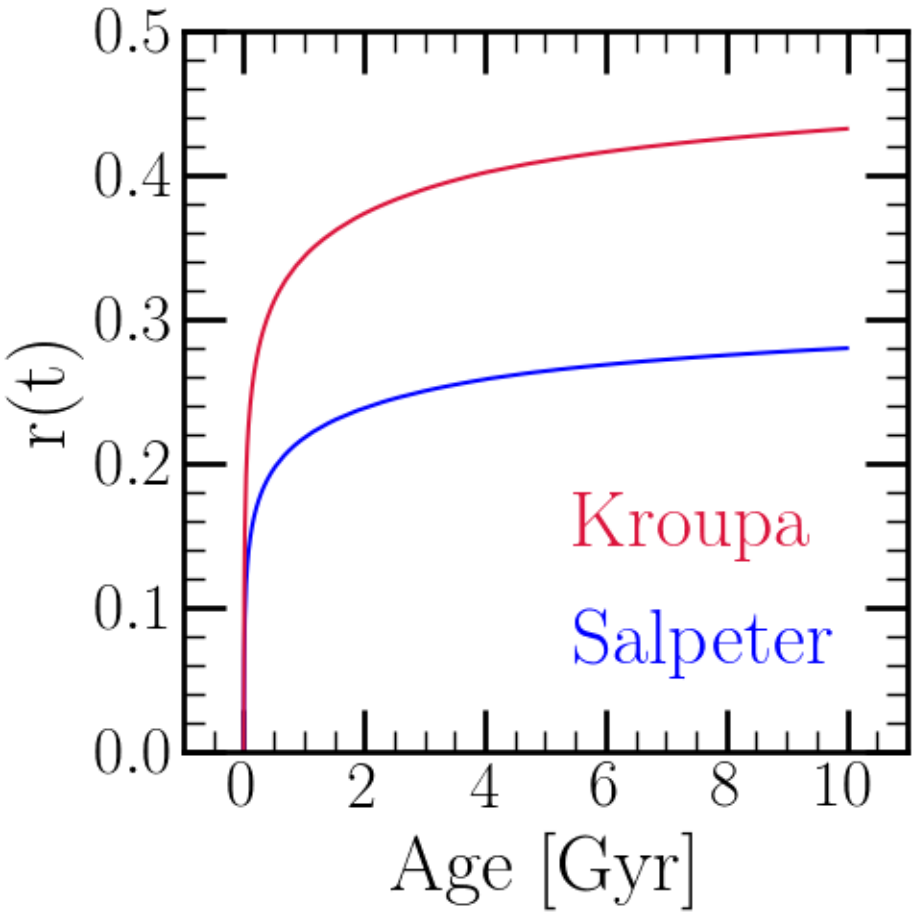

Fig. 3: The cumulative return fraction as a function of age for Kroupa[23] (red) and Salpeter[24] (blue) IMFs assuming the mass-lifetime relation of Hurley, Pols & Tout (2000)[25] (see discussion *here*). The Kroupa IMF is higher at all nonzero ages because it has fewer low mass stars than Salpeter. In both cases the post main sequence lifetime is assumed to be 10% of the main sequence lifetime (i.e. $p_{\rm MS} = 0.1$).

where the summation is over the relevant mass ranges with different power-law indeces α_i. In the case of kroupa α = 2.3, 1.3, and 0.3 for m > 0.5, 0.08 $\leq m \leq$ 0.5, and m < 0.08, respectively.

Here we plot h as a function of the stellar population's age assuming the mass-lifetime relation of Hurley, Pols & Tout (2000)[27] (see discussion *here*). By 10 Gyr, $h(t)$ is as low as ~ 0.45 for the Kroupa IMF and ~ 0.65 for the Salpeter[28] IMF. In comparison, the *cumulative return fraction* $r(t) \approx 0.45$ for the Kroupa IMF and ~ 0.28 for the Salpeter IMF. This suggests that the approximation $h(t) \approx 1 - r(t)$ fails at the $\sim 5 - 10\%$ level, depending on the choice of IMF. This suggests that for old stellar populations, a non-negligible portion of the mass is contained in evolved stars and stellar remnants. VICE therefore differentiates between these two quantities in its implementation.

In reality, the rate of the stellar mass evolving off of the main sequence is given by $\dot{h}M_*$ where M_* is the initial mass of the stellar population. However, the quantization of timesteps in VICE allows each timestep to represent a single stellar population which will eject mass $M_* dh$ in a time interval dt. For that reason, VICE is implemented with a calculation of $h(t)$ rather than $\dot{h}$.

In calculations of $h(t)$ with the built-in Kroupa and Salpeter IMFs, the analytic solution is calculated. In the case of a user-customized IMF, VICE solves the equation numerically using quadrature with the methods described in chapter 4 of Press, Teukolsky, Vetterling & Flannery (2007)[29].

Relevant source code:

- `vice/src/ssp/msmf.c`
- `vice/src/yields/integral.c`

3.3.4 Enrichment from Single Stellar Populations

While galaxies form stars continuously, it is often an interesting scientific problem to quantify the nucleosynthetic production of only one population of conatal stars. This is inherently cheaper computationally, since this is only one stellar population while galaxy simulations require many stellar populations.

VICE includes functionality for simulating the mass production of a given element from a single stellar population (i.e. an individual star cluster) of given mass and metallicity under user-specified yields. This by construction does not take into account depletion from infall low metallicity gas and star formation, ejection in outflows, recycling, etc. It only calculates the mass production of the element as a function of the stellar population's age.

The star cluster is assumed to form at time $t = 0$, and thus at this time there is no net production. Because VICE operates under the assumption that all core-collapse supernovae (CCSNe) occur instantaneously following the star cluster's formation[33], the entire CCSN net yield is injected within the first timestep at $t = \Delta t$:

$$\Delta M_x = y_x^{\text{CC}}(Z) M_\star$$

where $y_x^{\text{CC}}(Z)$ is the user's current setting for CCSN yields at a stellar metallicity Z. At subsequent timesteps, enrichment from asymptotic giant branch (AGB) stars is injected according to[34]:

$$\dot{M}_x^{\text{AGB}} \Delta t \approx y_x^{\text{AGB}}(m_{\text{postMS}}(t), Z) M_\star \left[h(t) - h(t + \Delta t)\right]$$

and from type Ia supernovae (SN Ia) according to[35]:

$$\dot{M}_x^{\text{Ia}} \Delta t \approx y_x^{\text{Ia}}(Z) M_\star \frac{R_{\text{Ia}}(t)}{\int_0^\infty R_{\text{Ia}}(t') dt'}$$

[27] Hurley, Pols & Tout (2000), MNRAS, 315, 543
[28] Salpeter (1955), ApJ, 121, 161
[29] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press
[30] Kroupa (2001), MNRAS, 322, 231
[31] Salpeter (1955), ApJ, 121, 161
[32] Hurley, Pols & Tout (2000), MNRAS, 315, 543
[33] See the discussion of *enrichment from CCSNe* for justification of this assumption.
[34] Justification of this can be found *here*.
[35] Justification of this can be found *here*.

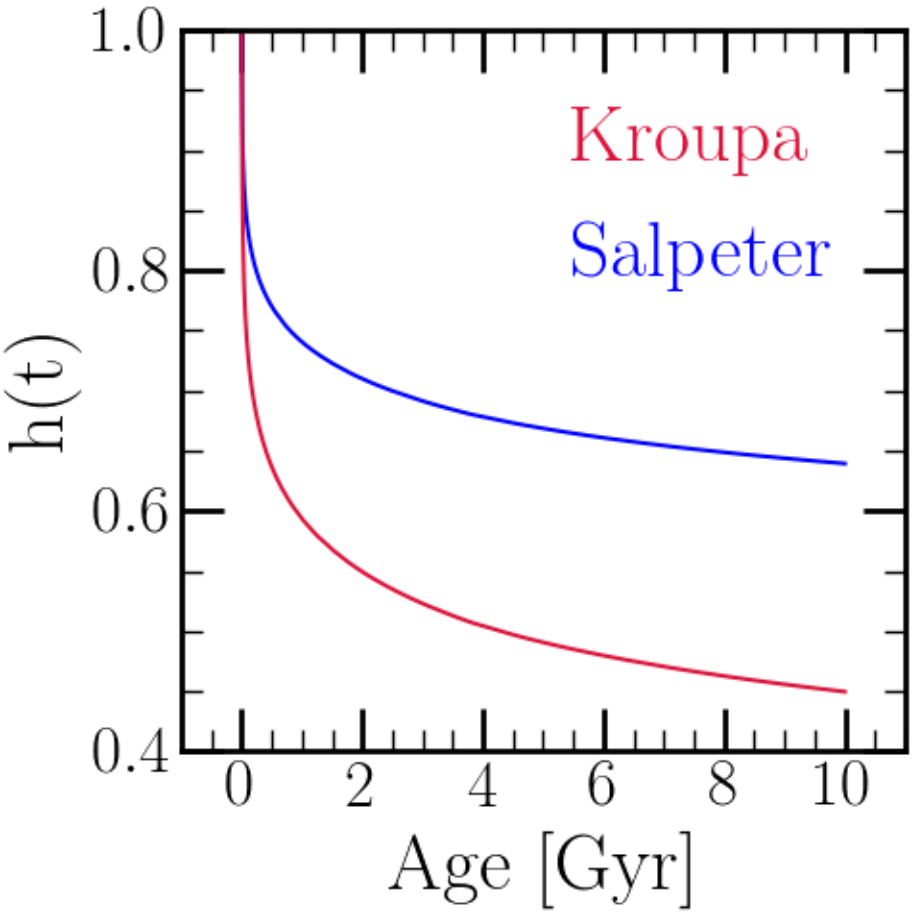

Fig. 4: The main sequence mass fraction as a function of age for Kroupa[30] and Salpeter[31] IMFs assuming the mass-lifetime relation of Hurley, Pols & Tout (2000)[32] (see discussion *here*). The Kroupa IMF is lower at all nonzero ages because it has fewer low mass stars than Salpeter.

These are the same equations that are implemented in simulating enrichment under the single-zone approximation, but applied to only one episode of star formation.

Users can run these simulations by calling `vice.single_stellar_population`.

Relevant Source Code:

- `vice/src/ssp/ssp.c`
- `vice/core/ssp/_ssp.pyx`

3.3.5 In Multizone Models

Like the timesteps in singlezone simulations, VICE treats star particles in multizone simulations as stand-ins for entire stellar populations. They eject newly produced heavy nuclei to the interstellar medium and recycle mass at their birth metallicities at the rates detailed in this section, where $M_\star$ denotes the mass of the star particle.

3.4 The Gas Supply

3.4.1 Inflows, Star Formation, and Efficiency

Like the *enrichment equation*, the time derivative of the mass of the gas in the interstellar medium (ISM) $M_{\rm g}$ is a simple sum of source and sink terms. For an infall rate (IFR) $\dot{M}_{\rm in}$, star formation rate (SFR) $\dot{M}_\star$, and outflow rate (OFR) $\dot{M}_{\rm out}$:

$$\dot{M}_g = \dot{M}_{\rm in} - \dot{M}_\star - \dot{M}_{\rm out} + \dot{M}_{\rm r}$$

where $\dot{M}_{\rm r}$ is the rate of recycling from stars producing remnants and return gas to the ISM at their birth metallicity. Because VICE is implemented with a Forward Euler solution, this equation is evaluated via:

$$\Delta M_g \approx \dot{M}_g \Delta t = \dot{M}_{\rm in}\Delta t - \dot{M}_\star \Delta t - \dot{M}_{\rm out}\Delta t + \dot{M}_{\rm r}\Delta t$$

By construction, VICE operates such that the user specifies either an infall history ($\dot{M}_{\rm in}$ in $M_\odot yr^{-1}$ as a function of time), a star formation history ($\dot{M}_\star$ in $M_\odot yr^{-1}$ as a function of time), or the gas history ($M_{\rm g}$ in $M_\odot$ as a function of time). The user also specifies a star formation efficiency timescale[1]:

$$\tau_\star \equiv \frac{M_g}{\dot{M}_\star}$$

Users may specify an arbitrary function of time in Gyr to describe $\tau_\star$, whose units are assumed to be Gyr. With one of either $\dot{M}_{\rm in}$, $\dot{M}_\star$, or $M_{\rm g}$ specified by the user, $\tau_\star$, and the implementation of $\dot{M}_{\rm out}$ and $\dot{M}_{\rm r}$ discussed in this section, the solutions to $\dot{M}_{\rm in}$, $\dot{M}_\star$, and M_g as functions of time are unique.

VICE also allows users to adopt a formulation of $\tau_\star$ that depends on the gas supply; this is an application of the Kennicutt-Schmidt relation to the single-zone approximation. This is implemented as a power-law:

$$\tau_\star^{-1} = \tau_{\star,\rm spec}^{-1}\left(\frac{M_g}{M_{g,\rm Schmidt}}\right)^\alpha$$

where $M_{g,\rm Schmidt}$ is a normalizing gas supply and $\tau_{\star,\rm spec}$ is the user-specified $\tau_\star$. The `singlezone` object will employ this scaling when the attribute `schmidt = True`.

Users may also allow $\tau_\star$ to vary with the gas supply in a customized way by specifying a function which accepts a second parameter in addition to time in Gyr. In infall and gas modes, VICE will interpret the second parameter as the

[1] In the interstellar medium literature, this quantity is often referred to as the "depletion time" due to star formation. In the chemical evolution literature, it quantifies the fractional rate at which gas forms stars, and is thus often refered to in terms of star formation efficiency. We retain this nomenclature here.

gas mass in $M_\odot$; in star formation mode, VICE will interpret it as the star formation rate in $M_\odot/yr$. While such an approach also allows the single power-law solution, such a model has a well-defined solution implemented in VICE's C library, allowing them to not suffer from a decrease in computational speed.

Relevant Source Code:

- `vice/src/singlezone/ism.c`

3.4.2 Outflows

In the astronomical literature, the strength/efficiency of outflows are typically quantified according to a dimensionless parameter referred to as the *mass loading factor*, defined as the ratio of the mass outflow rate to the star formation rate: $\eta \equiv \dot{M}_{\rm out}/\dot{M}_\star$. Johnson & Weinberg (2020) introduced a new parameter to generalize this, dubbed the "outflow smoothing time." This is the timescale on which the star-formation rate is averaged (i.e. "smoothed") to determine the outflow rate:

$$\dot{M}_{\rm out} = \eta(t)\langle\dot{M}_\star\rangle_{\tau_{\rm s}} = \frac{\eta(t)}{\tau_{\rm s}}\int_{t-\tau_{\rm s}}^{t}\dot{M}_\star(t')dt'$$

At early times when $0 \leq t \leq \tau_{\rm s}$, this average is taken over only the time interval from 0 to t. This equation is approximated numerically according to:

$$\dot{M}_{\rm out} \approx \eta(t)\frac{\Delta t}{\tau_{\rm s}}\sum_{i=0}^{\tau_{\rm s}/\Delta t}\dot{M}_\star(t - i\Delta t)$$

Put simply, at each timestep VICE looks backs at the number of timesteps corresponding to the smoothing time, and determines the arithmetic mean of the star formation rate at those timesteps, then multiplies this number by $\eta(t)$, which may be a user-specified function of time in Gyr. An advantage of this formulation is that when $\tau_{\rm s} < \Delta t$, VICE automatically recovers the traditional relation of $\dot{M}_{\rm out} = \eta(t)\dot{M}_\star(t)$.

Note: It is only the star formation rate which is time averaged. The mass loading factor is not time-averaged in any way.

Relevant Source Code:

- `vice/src/singlezone/ism.c`

3.4.3 Recycling

As stars produce remnants, the mass that does not end up in the remnant is returned to the interstellar medium (ISM). The net effect of this from all previous episodes of star formation quantifies the rate of recycling:

$$\dot{M}_{\rm r} = \int_0^t \dot{M}_\star(t - t')\dot{r}(t')dt'$$

where $r(\tau)$ is the *cumulative return fraction* from a single stellar population of age τ. This is approximated numerically according to

$$\dot{M}_{\rm r} \approx \sum_i \dot{M}_\star(t - i\Delta t)\left[r((i+1)\Delta t) - r(i\Delta t)\right]$$

This is an instance where the quantization of star forming episodes due to the Forward Euler solution simplifies the implementation; the stars that form in previous timesteps contribute Δr of their mass back to the ISM.

In the case of instantaneous recycling, this simplifies further to

$$\dot{M}_{\rm r} \approx r_{\rm inst}\dot{M}_{\star}$$

Weinberg, Andrews & Freudenburg (2017)[2] demonstrate that $r_{\rm inst}$ = 0.4 (0.2) for a Kroupa[3] (Salpeter[4]) IMF are good approximations.

Note: Instantaneous recycling refers only previously produced nucleosynthetic products. While this term has been used to refer to instantaneous production of new heavy nuclei in the astronomical literature in the past, VICE retains this approximation only for enrichment from core collapse supernovae.

Relevant Source Code:

- `vice/src/singlezone/recycling.c`

Extension to Multizone Models

In a multizone simulation, the rate of recycling may change due to stars forming in a given zone and moving out of it, which decreases the rate of recycling, as well as stars moving into it, which increases the rate of recycling. In these simulations, however, VICE knows the zone number of each star particle; the rate of recycling can then be determined from the initial mass and age of each star particle in a given zone. This is given by:

$$\dot{M}_{\rm r}\Delta t = \sum_i M_i \left[r\left(\tau_i + \Delta t\right) - r\left(\tau_i\right)\right]$$

where M_i and τ_i are the initial mass and age, respectively, of the i'th star particle in a given zone.

Relevant Source Code:

- `vice/src/multizone/recycling.c`

3.5 Enrichment

VICE takes a general approach in modeling nucleosynthesis. All elements are treated equally; there are no special considerations for any element. In this documentation we derive the analytic form of *the enrichment equation* for an arbitrary element x with arbitrary nucleosynthetic yields for arbitrary evolutionary histories. This is an integro-differential equation of the element's mass as a function of time, which VICE solves as an initial-value problem by imposing the boundary condition that its abundance at time zero is given by the primordial abundance from big bang nucleosynthesis. In this version of VICE, helium is the only element for which this value is nonzero.

3.5.1 The Enrichment Equation

The enrichment equation quantifies the rate of change of an element's total mass present in the interstellar medium (ISM). At its core, it is a simple sum of source and sink terms.

$$\dot{M}_x = \dot{M}_x^{\rm CC} + \dot{M}_x^{\rm Ia} + \dot{M}_x^{\rm AGB} - \frac{M_x}{M_g}\left[\dot{M}_{\star} + \xi_{\rm enh}\dot{M}_{\rm out}\right] + \dot{M}_x^{\rm r} + Z_{x,\rm in}\dot{M}_{\rm in}$$

[2] Weinberg, Andrews & Freudenburg (2017), ApJ, 837, 183
[3] Kroupa (2001), MNRAS, 322, 231
[4] Salpeter (1955), ApJ, 121, 161

where M_x is the mass of the element x in the interstellar medium, $\dot{M}_x$ its time-derivative, and M_g the mass of the ISM gas. $\dot{M}_x^{\text{CC}}$, $\dot{M}_x^{\text{Ia}}$, and $\dot{M}_x^{\text{AGB}}$ quantify the rate of production from core-collapse supernovae (CCSNe), type Ia supernovae (SNe Ia), and asymptotic giant branch (AGB) stars, respectively.

We detail each term individually here.

3.5.2 Core Collapse Supernovae

Core collapse supernovae (CCSNe) are the explosions of massive stars ($\gtrsim 8M_\odot$) at the end of their post main sequence lifetimes. Due to the steep nature of the lifetime-stellar mass relationship, these stars have lifetimes that are extremely short compared to the relevant timescales of galactic chemical evolution ($\sim$ few Myr compared to $\sim$ few Gyr). To a good approximation, the lifetimes of these stars can be treated as instantaneous in zone models.

Note: Another motivation for this approximation is that the lifetimes are often significantly shorter than the typical mixing timescales in even modestly sized galaxies. The longest lifetimes of these stars is of order tens of megayears; in comparison, the mixing timescale in the solar annulus of the Milky Way is likely comparable to the dynamical timescale at this distance ($\sim$ 250 Myr, a factor of ten larger). Zone models at their core already assume that these mixing timescales are negligibly short due to the assumption of instantaneous mixing; if CCSN timescales are even shorter, then they can certainly also be modeled as instantaneous.

VICE therefore approximates CCSNe as being simultaneous with the formation of their progenitor stars. This implies a linear relationship between the rate of production of some element x from CCSNe and the star formation rate:

$$\dot{M}_x^{\text{CC}} = \epsilon_x^{\text{CC}} y_x^{\text{CC}}(Z)\dot{M}_\star$$

where y_x^{CC} is the *IMF-averaged fractional net yield* of the element x from CCSNe at a metallicity Z: the fraction of the entire stellar population's initial mass that is processed into the element x *and* ejected to the interstellar medium *minus* the amount that the star was born with. ϵ_x^{CC} is the *entrainment fraction* of the element x from CCSNe; this is the mass fraction of the net yield which is retained by the interstellar medium, the remainder of which is added directly to the outflow.

Note: VICE implements recycling of previously produced elements separate from nucleosynthesis, running from the standpoint of *net* rather than *absolute* yields.

In practice, y_x^{CC} is highly uncertain[1]. VICE therefore makes no assumptions about the user's desired form of the yield; this parameter can be assigned either a number to represent a metallicity-independent yield, or a function of the metallicity by mass $Z = M_x/M_g$. VICE includes features which will calculate the value of y_x^{CC} for a given element and metallicity based on the results of supernova nucleosynthesis studies upon request, but requires the user to specify an exact number or function.

Relevant Source Code:

- `vice/src/singlezone/ccsne.c`
- `vice/core/dataframe/_yield_settings.pyx`
- `vice/yields/ccsne/__init__.py`

[1] See Andrews, Weinberg, Schoenrich & Johnson (2017), ApJ, 835, 224 and the citations therein for a detailed analysis of multiple elements.

Extension to Multizone Models

Because VICE approximates core collapse supernovae as occuring instantaneously following the formation of their progenitor stars, this implies that CCSN progenitors also should not migrate between zones in a multizone model. Therefore, the formalism implemented for singlezone models is retained in multizone simulations.

Relevant Source Code:

- `vice/src/multizone/ccsne.c`

3.5.3 Type Ia Supernovae

Type Ia supernovae are the thermonuclear detonations of white dwarf stars. Being the remnants of lower-mass stars, white dwarfs are born and explode on timescales longer than the mixing timescales of galaxies. Therefore, the intrinsic time delay is non-negigible.

This requires a model for the SN Ia delay-time distribution (DTD), defined as the rate of SN Ia explosions associated with a single stellar population. Given a DTD $R_{\rm Ia}$ and an age τ, the rate of production of some element x from a single stellar population is given by

$$\dot{M}_x^{\rm Ia} = \epsilon_x^{\rm Ia} y_x^{\rm Ia}(Z) M_* \frac{R_{\rm Ia}(\tau)}{\int_0^\infty R_{\rm Ia}(t)dt}$$

Note: The integral of this equation from $t = 0$ to ∞ must equal the yield times the mass of the stellar population. This necessitates the normalization of the DTD.

---

where $y_x^{\rm Ia}$ is the *IMF-averaged fractional net yield* of the element x from SNe Ia at metallicity Z: the fraction of the stellar population's initial mass that is processed into the element x *and* ejected to the interstellar medium *minus* the amount that the star was born with. $\epsilon_x^{\rm Ia}$ is the *entrainment fraction* of the element x from SNe Ia; this is the mass fraction of the net yield which is retained by the interstellar medium, the remainder of which is added directly to the outflow.

Note: VICE implements recycling of previously produced elements separate from nucleosynthesis, running from the standpoint of *net* rather than *absolute* yields.

In practice, $y_x^{\rm Ia}$ is highly uncertain[2]. VICE therefore makes no assumptions about the user's desired form of the yield; this parameter can be assigned either a number to represent a metallicity-independent yield or a function of metallicity by mass $Z = M_x/M_g$. VICE includes features which will calculate the value of $y_x^{\rm Ia}$ for a given element and metallicity based on the results of supernova nucleosynthesis studies upon request, but requires the user to specify an exact number or function.

The rate of enrichment from all previous episodes of star formation can be derived by integrating this equation over all ages:

$$\dot{M}_x^{\rm Ia} = \epsilon_x^{\rm Ia} y_x^{\rm Ia}(Z) \frac{\int_0^t \dot{M}_*(t') R_{\rm Ia}(t - t')dt'}{\int_0^\infty R_{\rm Ia}(t')dt'}$$

This can also be expressed as the star formation history up to a time t weighted by the SN Ia rate. VICE approximates this equation as:

$$\dot{M}_x^{\rm Ia} \approx \frac{\sum_i \epsilon_x^{\rm Ia} y_x^{\rm Ia}(Z_{\rm ISM}(i\Delta t))\dot{M}_*(i\Delta t) R_{\rm Ia}(t - i\Delta t)\Delta t}{\sum_i^{T_{\rm Ia}} R_{\rm Ia}(i\Delta t)\Delta t}$$

[2] See Andrews, Weinberg, Schoenrich & Johnson (2017), ApJ, 835, 224 and the citations therein for a detailed analysis of multiple elements.

where the sum in the numerator is over all timesteps and in the denominator up to a time $T_{\rm Ia}$ denoting an adopted full length of the SN Ia duty cycle. The constant `RIA_MAX_EVAL_TIME` declares $T_{\rm Ia}$ = 15 Gyr in `vice/src/sneia.h`.

In implementation, VICE normalizes the DTD at the beginning of the simulation. For an age $\tau = t - t'$:

$$R_{\rm Ia}(\tau) \to \frac{R_{\rm Ia}(\tau)}{\int_0^{T_{\rm Ia}} R_{\rm Ia}(\tau)d\tau} \approx \frac{R_{\rm Ia}(t - i\Delta t)}{\sum_i^{T_{\rm Ia}} R_{\rm Ia}(i\Delta t)\Delta t} \implies R_{\rm Ia}(t - t')\Delta t \to \frac{R_{\rm Ia}(t - i\Delta t)\Delta t}{\sum_i^{T_{\rm Ia}} R_{\rm Ia}(i\Delta t)\Delta t}$$

Inserting the normalized rate into the equation for $\dot{M}_x^{\rm Ia}$:

$$\dot{M}_x^{\rm Ia} \approx \sum_i \epsilon_x^{\rm Ia} y_x^{\rm Ia}(Z_{\rm ISM}(i\Delta t))\dot{M}_*(i\Delta t)R_{\rm Ia}(t - i\Delta t)$$

VICE implements this normalization of $R_{\rm Ia}$ at the beginning of simulations due to the simplification of this expression introduced in doing so. This reduces the computational expense in calculating this quantity for each element at each timestep.

VICE includes two built-in DTDs, denoting by strings as `plaw` and `exp`. As their names suggest, they are a power-law and an exponential DTD:

- "plaw": $R_{\rm Ia} \sim t^{-1.1}$
- "exp": $R_{\rm Ia} \sim e^{-t/\tau_{\rm Ia}}$

Users may also construct their own functional forms of $R_{\rm Ia}$, which must accept time in Gyr as the only parameter. These functions need not be normalized in any way; VICE normalizes the DTD automatically.

Relevant Source Code:

- `vice/src/sneia.h`
- `vice/src/singlezone/sneia.c`
- `vice/yields/sneia/__init__.py`

Extension to Multizone Models

The migration of star particles into and out of zones can affect the SN Ia rate in a given zone. In a singlezone simulation it is exactly as expected for the star formation history, but in a multizone model, it is coupled to the star formation histories in other zones. Because VICE knows the zone each star particle occupies at all times in simulation, the rate of SN Ia production of some element x should be expressed not as an integral over the star formation history, but as a summation over the stellar populations present in the zone:

$$\dot{M}_x^{\rm Ia} \approx \sum_i \epsilon_x^{\rm Ia} y_x^{\rm Ia}(Z_i) M_i R_{\rm Ia}(\tau_i)$$

where (Z_i), M_i, and τ_i are the metallicity, initial mass, and age, respectively, of the i'th star particle in a given zone at a given time.

It is important to note that with this implementation, VICE is not sampling the SN Ia delay time distribution and stochastically ejecting iron according thereto. Instead, star particles gradually eject iron to the interstellar medium with a time-dependence given by the delay-time distribution.

Relevant Source Code:

- `vice/src/multizone/sneia.c`

3.5.4 Asymptotic Giant Branch Stars

Asymptotic giant branch (AGB) stars are evolved stars that have carbon-oxygen cores surrounded by helium and hydrogen shells. These stars undergo thermal pulsations due to explosive ignition of helium fusion in the shell, typically referred to as helium shell flashes. During these pulses, material from the core is often mixed into the outer layers via convection, a process known as *dredge-up*. This brings heavy nuclei produced in the deeper regions of the star to the envelope, which is then ejected to the interstellar medium (ISM). This is one of the primary sites of s-process nucleosynthesis in the universe.

It may be tempting to model AGB star enrichment as a delay-time distribution (DTD) similar to that adopted for SNe Ia. However, this approach would implicitly adopt the assumption that every element is enriched via AGB stars with the same DTD, or that for a given element, the effective DTD is independent of metallicity. These may be fine assumptions, but it is not adopted in VICE due to the desire for as few assumptions as possible.

Instead, AGB star enrichment in VICE is implemented using the *mass-lifetime relationship for stars* and the *main sequence mass fraction* (MSMF). However, the form of the *MSMF* required here differs in detail from the true *MSMF*. Being evolved stars, the *MSMF* does not consider AGB stars. It is thus not the *MSMF* and the main sequence lifetimes of stars that are of interest, but the mass fraction of both main sequence and evolved stars and the *total* lifetime of stars. The form of $h(t)$ necessary for modeling AGB star enrichment then changes to:

$$h(t) \to \frac{\int_l^{m_{\text{postMS}}(t)} m \frac{dN}{dm} dm}{\int_l^u m \frac{dN}{dm} dm}$$

The numerator is evaluated from l to the mass of stars ending their post main sequence lifetime m_{postMS} rather than the main sequence turnoff mass m_{to}. As detailed here for a stellar population of age τ:

$$m_{\text{postMS}} = \left(\frac{\tau}{(1 + p_{\text{MS}})\tau_\odot}\right)^{-1/\alpha}$$

where α is the power-law index on the *mass-lifetime relationship*, $\tau_\odot$ is the main sequence lifetime of the sun, and p_{MS} is the ratio of a star's post main sequence lifetime to its main sequence lifetime.

From a single stellar population, the rate of ejection of an element x from AGB stars to the ISM is given by:

$$\dot{M}_x^{\text{AGB}} = -\epsilon_x^{\text{AGB}} y_x^{\text{AGB}}(m_{\text{postMS}}, Z) M_\star \dot{h}$$

where $\dot{h}$ is evaluated at the lookback time to the stellar population's formation[3], $M_\star$ is the initial mass of the stellar population, and y_x^{AGB} is the *fractional net yield* of x from an AGB star of initial mass m_{postMS} and metallicity Z: the fraction of a single star's initial mass that is processed into element x *and* ejected to the interstellar medium *minus* the amount that the star was born with. ϵ_x^{AGB} is the *entrainment fraction* of the element x from AGB stars; this is the mass fraction of the net yield which is retained by the interstellar medium, the remainder of which is added directly to the outflow.

Note: VICE implements recycling of previously produced elements separate from nucleosynthetic yields, running from the standpoint of *net* rather than *absolute* yields.

For continuous star formation, the enrichment rate can be expressed as this quantity integrated over the star formation history:

$$\dot{M}_x^{\text{AGB}} = -\int_0^t \epsilon_x^{\text{AGB}} y_x^{\text{AGB}}(m_{\text{postMS}}(t - t'), Z_{\text{ISM}}(t')) \dot{M}_\star(t') \dot{h}(t - t') dt$$

This expression is approximated numerically as:

$$\dot{M}_x^{\text{AGB}} \approx \sum_i \epsilon_x^{\text{AGB}} y_x^{\text{AGB}}(m_{\text{postMS}}(t - i\Delta t), Z_{\text{ISM}}(i\Delta t)) \dot{M}_\star(i\Delta t) \left[h((i+1)\Delta t) - h(i\Delta t)\right]$$

[3] There is a minus sign here because $h(t)$ is a monotonically decreasing function, and thus $\dot{h} < 0$.

where the summation is taken over all previous timesteps. The need to differentiate h with time is eliminated in the numerical approximation by allowing each stellar population to be weighted by Δh between the current timestep and the next, made possible by the quantization of timesteps.

In practice, y_x^{AGB} is highly uncertain[4]. VICE therefore makes no assumptions about the user's desired form of the yield; this parameter can be assigned either a built-in table published in an AGB star nucleosynthesis study or a function of stellar mass and metallicity constructed by the user.

Relevant source code:

- `vice/src/singlezone/agb.c`
- `vice/core/dataframe/_agb_yield_settings.pyx`
- `vice/yields/agb/__init__.py`

Extension to Multizone Models

The migration of star particles into and out of zones can affect the AGB star enrichment rate in a given zone. In a singlezone simulation it is exactly as expected for the star formation history, but in a multizone model, it is coupled to the star formation histories in other zones. Because VICE knows the zone each star particle occupies at all times in simulation, the rate of AGB star enrichment rate of some element x should not be expressed as an integral over the star formation history, but as a summation over the stellar populations present in the zone:

$$\dot{M}_x^{\text{AGB}} \approx \sum_i \epsilon_x^{\text{AGB}} y_x^{\text{AGB}}(m_{\text{postMS}}(\tau_i), Z_i) M_i [h(\tau_i) - h(\tau_i + \Delta t)]$$

where Z_i, M_i, and τ_i are the metallicity, initial mass, and age, respectively, of the i'th star particle in a given zone at a given time.

Relevant Source Code:

- `vice/src/multizone/agb.c`

3.5.5 Subsequent Terms

The remaining terms in the enrichment equation make simple statements about remaining source and sink terms.

VICE retains the assumption that stars are born at the same metallicity as the ISM from which they form. This motivates the sink term

$$-\left(\frac{M_x}{M_g}\right)\dot{M}_\star$$

where the mass of the element x is depleted at the metallicity of the ISM $Z_x = M_x/M_g$ in proportion with the star formation rate $\dot{M}_\star$.

Many galactic chemical evolution models to date have assumed that outflows from galaxies occur at the same metallicity of the ISM. This would suggest that $\dot{M}_x^{\text{out}} \approx (M_x/M_g)\dot{M}_{\text{out}}$. However, recent work in the astronomical literature from both simulations (e.g. Christensen et al. (2018)[5]) and observations (e.g. Chisholm, Trimonti & Leitherer (2018)[6]) suggest that this may not be the case. Therefore, VICE allows outflows to occur at some multiplicative factor ξ_{enh} above or below the ISM metallicity, which may vary with time. This motivates the sink term

$$-\left(\frac{M_x}{M_g}\right)\xi_{\text{enh}}\dot{M}_{\text{out}}$$

[4] See Andrews, Weinberg, Schoenrich & Johnson (2017), ApJ, 835, 224 and the citations therein for a detailed analysis of multiple elements.
[5] Christensen et al. (2018), ApJ, 867, 142
[6] Chisholm, Trimonti & Leitherer (2018), MNRAS, 481, 1690

Because *VICE works with net rather than absolute yields*, simulations must quantify the rate at which stars return mass to the ISM at their birth metallicity. This is mathematically similar to the rate of total gas recycling, but weighted by the metallicities of the stars recycling. Since stars are assumed to form at the metallicity of the ISM,

$$\dot{M}_x^{\text{r}} = \int_0^t \dot{M}_\star(t') Z_{x,\text{ISM}}(t') \dot{r}(t - t') dt$$

where $r(\tau)$ is the *cumulative return fraction* from a single stellar population of age τ. This is approximated numerically as

$$\dot{M}_x^{\text{r}} \approx \sum_i \dot{M}_\star(i\Delta t) Z_{x,\text{ISM}}(i\Delta t) \left[r((i+1)\Delta t) - r(i\Delta t)\right]$$

where the summation is taken over all previous timesteps. The need to differentiate r with time is eliminated in the numerical approximation by allowing each stellar population to be weighted by Δr between the current timestep and the next, made possible by the quantization of timesteps. In the event that the user has specified instantaneous recycling:

$$\dot{M}_x^{\text{r}} = r_{\text{inst}} \dot{M}_\star Z_{x,\text{ISM}}$$

At any given timestep, there is gas infall onto the simulated galaxy of a given metallicity Z. In most cases this term is negligibly small, but in some interesting cases it may not be (e.g. a major merger event). This necessitates the final term $Z_{x,\text{in}}\dot{M}_{\text{in}}$.

Relevant Source Code:

- `vice/src/singlezone/recycling.c`
- `vice/src/singlezone/element.c`
- `vice/src/singlezone/ism.c`

Extension to Multizone Models

The only subsequent term of the enrichment equation modified in multizone simulations is that quantifying the rate of recycling of an element x. The migration of star particles into and out of zones can affect the recycling rate in a given zone. In a singlezone simulation it is exactly as expected for the star formation history, but in a multizone model, it is coupled to the star formation histories in other zones. Because VICE knows the zone each star particle occupies at all times in simulation, the rate of recycling of some element x should be expressed not as an integral over the star formation history, but as a summation over the stellar populations in the zone:

$$\dot{M}_x^{\text{r}} \approx \sum_i M_i Z_{x,i} [r(\tau_i + \Delta t) - r(\tau_i)]$$

where M_i, Z_i, and τ_i are the mass, metallicity, and age, respectively, of the i'th star particle in a given zone at a given time.

Relevant Source Code:

- `vice/src/multizone/recycling.c`
- `vice/src/multizone/element.c`
- `vice/src/multizone/ism.c`

3.5.6 Sanity Checks

At all timesteps VICE forces the mass of every element to be non-negative. If the mass is found to be below zero at any given time, it is assumed to not be present in the interstellar medium and is assigned a mass of exactly zero. Absent this, the mass of each element reported by VICE is merely the numerically estimated solution to the enrichment equation.

Relevant source code:

- `vice/src/singlezone/element.c`

3.6 Nucleosynthetic Yields

Due to the associated uncertainties[1], VICE takes an agnostic approach to the user's desired nucleosynthetic yields. Rather than adopting the results of a nucleosynthesis study, the user declares their yields outright. VICE includes features which will calculate yields upon request, but requires the user to explicitly tell it what the yield of each element from each enrichment channel should be (although there is a set of defaults).

All yields in VICE are defined as *fractional net yields*. This is the amount of an element that is *produced and ejected* to the interstellar medium *minus* that which was already present, in units of the star or stellar population's initial mass. Previously produced nuclei should not be taken into account, because this is handled via *recycling*. For example, if a stellar population is born with $1M_\odot$ of oxygen total and ejects $1M_\odot$ of oxygen back to the interstellar medium, the yield is zero since there is no net gain.

Yields are also defined for the average star or stellar population. Stochasticity in yields introduced by, e.g., sampling of the stellar initial mass function, should not be taken into account in yield calculations intended for use in VICE.

3.6.1 Core Collapse Supernovae

Because core collapse supernovae (CCSNe) are assumed to occur simultaneously with the formation of their progenitor stars[2], y_x^{CC} represents the total yield from all CCSNe associated with a single stellar population. Letting m_x denote the net mass of some element x present in the CCSN ejecta, the yield at a given metallicity is defined by:

$$y_x^{\text{CC}} \equiv \frac{\int_{l_{\text{CC}}}^{u} (E(m)m_x + w_x - Z_{x,\text{prog}}m)\frac{dN}{dm}dm}{\int_l^u m\frac{dN}{dm}dm}$$

where the numerator is taken from the minimum mass for a CCSN explosion l_{CC} to the upper mass limit of star formation u, but the denominator is over the entire mass range of star formation, and dN/dm is the stellar initial mass function (IMF). This equation is nothing more or less than the mathematical statement of "production divided by total initial mass." $E(m)$ denotes the *explodability*: the fraction of star of mass m which explode as a CCSN. w_x denotes the mass yield of the element x due to winds at a mass m, and the corrective term $Z_{x,\text{prog}}m$ accounts for the birth abundances of the star to compute a *net* rather than a *gross* yield. The constant `CC_MIN_STELLAR_MASS` declares $l_{\text{CC}} = 8M_\odot$ in `vice/src/ccsne.h`.

In practice, supernova nucleosynthesis studies determine the value of m_x for of order 10 values of m at a given metallicity and rotational velocity. To compute the numerator of this equation, VICE adopts a grid of m_x and w_x values from a user-specified nucleosynthesis study, interpolating linearly between values of m on the grid. We clarify that the interpolation is linear in m, and not $\log m$.

In this version of VICE, users can choose between the following nucleosynthesis studies:

- Limongi & Chieffi (2018), ApJS, 237, 13
- Sukhbold et al. (2016), ApJ, 821, 38 (W18 and N20 explosion engines)

[1] See Andrews, Weinberg, Schoenrich & Johnson (2017), ApJ, 835, 224 and the citations therein for a detailed analysis of multiple elements.
[2] See the discusion on *CCSN enrichment* for justification of this assumption.

- Chieffi & Limongi (2013), ApJ, 764, 21
- Nomoto, Kobayashi & Tominaga (2013), ARA&A, 51, 547
- Chieffi & Limongi (2004), ApJ, 608, 405
- Woosley & Weaver (1995), ApJS, 101, 181

VICE affords users the ability to specify whether or not winds should be included in their yield calculations; it stores values of w_x for all recognized elements at each metallicity and rotational velocity it has built-in tables for. It also allows users the option to calculate net or gross yields based on whether or not it sets $Z_{x,\text{prog}}$ equal to zero. We caution however that not every study separates their wind yields from their explosive yields, in which case $w_x = 0$ and the wind contribution is included in m_x. Furthermore, not every study reports the detailed initial composition of their model stars, in which case assigning $Z_{x,\text{prog}}$ is ambiguous. For these studies VICE is incapable of computing net yields, so it sets $Z_{x,\text{prog}}$ to zero always for these studies, only reporting gross yields. For a breakdown on which of these cautionary tales apply to which studies, we refer users to the `vice.yields.ccsne.fractional` documentation.

By default, VICE will assume that all stars above $8M_\odot$ explode as a CCSN. Because stellar explodability is an open question in astronomy[3], $E(m)$ can be specified as an arbitrary mathematical function, which must accept stellar mass in $M_\odot$ as the only parameter. Lastly, this can be done with either the built-in Kroupa[4] or Salpeter[5] IMFs, or a function of mass interpreted as a user-constructed IMF.

Note: VICE also forces m_x = 0 at $8M_\odot$, the default value of l_{CC}, in order to minimize numerical artifacts introduced when extrapolating off of the grid in m to lower stellar masses.

Although $E(m)$ can take on any mathematical form in VICE as long as its value is always between 0 and 1, a number of popular forms both simple and complex can be found in the `vice.yields.ccsne.engines` module.

Users can evaluate the solution to this equation by calling the function `vice.yields.ccsne.fractional`, implemented in `vice/yields/ccsne/_yield_integrator.pyx`. This function makes use of numerical quadrature routines written in ANSI/ISO C built into VICE, and is thus not dependent on any publicly available quadrature functions such as those found in `scipy`.

In addition to evaluating the solution to this equation, users may also read in the table of m_x values by calling `vice.yields.ccsne.table`, and may request the full isotopic breakdown. A `dataframe` is returned from this function.

Note: These functions have no impact whatsoever on the chemical enrichment simulations built into VICE. Users declare their own yields for that purpose, while this function merely calculates them.

Relevant Source Code:

- `vice/src/yields/integral.c`
- `vice/yields/ccsne/_yield_integrator.pyx`
- `vice/yields/ccsne/table.py`
- `vice/core/dataframe/_ccsn_yield_table.pyx`

[3] See the discussion in Sukhbold et al. (2016), ApJ, 821, 38 and the citations therein for details.
[4] Kroupa (2001), MNRAS, 231, 322
[5] Salpeter (1955), ApJ, 121, 161

3.6.2 Type Ia Supernovae

The net yield of some element x from a single stellar population due to type Ia supernovae (SNe Ia) can be expressed as the total production from the duty cycle of the delay-time distribution (DTD) R_{Ia}:

$$y_x^{\text{Ia}} \equiv M_x \int_0^\infty R_{\text{Ia}}(t)dt$$

where M_x is the average mass yield of the element x from a single type Ia supernovae.

Note: In the astronomical literature, the delay-time distribution is usually defined as the rate of SN Ia explosions per unit stellar mass formed $M_\star$. R_{Ia} thus has units of $M_\odot^{-1} yr^{-1}$, making y_x^{Ia} unitless as it should be. We retain this definition here for consistency.

The integral over the DTD is simply the number of SN Ia events that occur per unit stellar mass formed:

$$y_x^{\text{Ia}} = M_x \frac{N_{\text{Ia}}}{M_\star}$$

Intuitively, the SN Ia yield is thus specified by the mass yield of a single SN Ia explosion and the number of SN Ia events that occur per unit solar mass formed.

Maoz & Mannucci (2012)[6] found that $N_{\text{Ia}}/M_\star = (2 \pm 1) \times 10^{-3} M_\odot^{-1}$. That is, on average, approximately 500 $M_\odot$ of stars must form for a given stellar population to produce a single SN Ia.

The value of M_x can be determined from the results of simulation of SNe Ia. The yield is then evaluated with a user-specified value of $N_{\text{Ia}}/M_\star$; the default value is $N_{\text{Ia}}/M_\star = 2.2 \times 10^{-3}$, the best-fit value from Maoz & Mannucci (2012).

In this version of VICE, users can choose between the following nucleosynthesis studies:

- Iwamoto et al. (1999), ApJ, 124, 439
- Seitenzahl et al. (2013), MNRAS, 429, 1156

Note: These functions have no impact whatsoever on the chemical enrichment simulations built into VICE. Users declare their own yields for that purpose, while this function merely calculates them.

Relevant Source Code:

- `vice/yields/sneia/_yield_lookup.pyx`

3.6.3 Asymptotic Giant Branch Stars

The net yield of some element x from an asymptotic giant branch (AGB) star is defined as the net fraction of a star's mass that is converted to an element x. For many elements, this also varies considerably with the initial metallicity of the star. This is therefore inherently a function of two parameters:

$$y_x^{\text{AGB}}(M_\star, Z) = \frac{M_{x,\text{ejected}}}{M_\star(|Z)}$$

where $M_\star(|Z)$ is the mass of a single star of known metallicity Z.

Contrary to yields from supernovae, no remaining calculations are necessary, because $M_{x,\text{ejected}}$ is quantified in supernova nucleosynthesis studies, and VICE's internal data tables have already divided these values by $M_\star(|Z)$. These

[6] Maoz & Mannucci (2012), PASA, 29, 447

tables are sampled on of order ~ 10 solar masses and metallicities; users may adopt these tables in their simulations and VICE will determine the yield for all other masses and metallicities via bilinear interpolation between masses and metallicities on the grid. For masses and metallicities above or below the grid, it extrapolates from the two highest or lowest elements on the grid, respectively. Users may also construct their own mathematical forms of y_x^{AGB}.

When using a built-in table of yields, VICE enforces non-negative yields for progenitors with masses below 1.5 $M_\odot$. This is to prevent numerical artifacts associated with extrapolation to progenitor masses below the lowest value on the yield table. In the *figure below*, we plot the AGB star yields of barium as a function of progenitor mass for a handful of metallicities on the Cristallo et al. (2011, 2015) table of yields. This is a prototypical example of the motivation behind this decision - many elements have a non-monotonic dependence of their AGB star yields on progenitor mass, and this grid does not sample below 1.3 $M_\odot$. As a consequence, VICE's interpolation routines without modification would extrapolate $y_{\text{Ba}}^{\text{AGB}}$ to be negative for lower masses; this would be a purely numerical artifact. Rather than allowing linear extrapolation to infer potentially large, negative yields, it is safer to assume that the yields approach zero with decreasing mass.

If this behavior isn't desired and it would instead be preferred to allow linear extrapolation at low masses, the object `vice.yields.agb.interpolator` does not force yields to zero, and it can be used as an element's AGB star yield setting (i.e. its entry in the `vice.yields.agb.settings` object). Users looking to modify the AGB star yields of a given element with this object should also be aware that they may need to force yields to zero below some characteristic mass in order to suppress these numerical artifacts.

In this version of VICE, users can choose between the following nucleosynthesis studies:

- Cristallo et al. (2011, 2015)[7][8]
- Karakas (2010)[9]
- Ventura et al. (2013)[10]
- Karakas & Lugaro (2016)[11]; Karakas et al. (2018)[12]

Users can also read these tables in with the `vice.yields.agb.grid` function.

Relevant Source Code:

- `vice/src/singlezone/agb.c`
- `vice/yields/agb/_grid_reader.pyx`

3.7 Migration

The fundamental contrast between singlezone and multizone models of chemical evolution is the variation of parameters between zones. However, these models often involve some sort of prescription for how stars mix between zones in order to mimic the changes in a stars' locations over time. In nature, the details of this component of evolution likely varies on a galaxy-by-galaxy basis. In keeping with VICE's philosophy of making as few assumptions as possible to maximize the user's power over their simulations, VICE is implemented with an agnostic approach to the migration prescription in a multizone model.

In multizone models, VICE knows nothing of the spatial configuration of the zones the user is operating under. The only piece of information identifying a zone is an integer index (a *zero-based* integer index, specifically). These zones can be coupled via migration by moving gas and stars from one zone to any other zone under a user-constructed prescription. In principle, this allows the construction of 1-, 2-, and 3-dimensional zone configurations with an arbitrary migration prescription.

[7] Cristallo et al. (2011), ApJS, 197, 17
[8] Cristallo et al. (2015), ApJS, 219, 40
[9] Karakas (2010), MNRAS, 403, 1413
[10] Ventura et al. (2013), MNRAS, 431, 3642
[11] Karakas & Lugaro (2016), ApJ, 825, 26
[12] Karakas et al. (2018), MNRAS, 477, 421

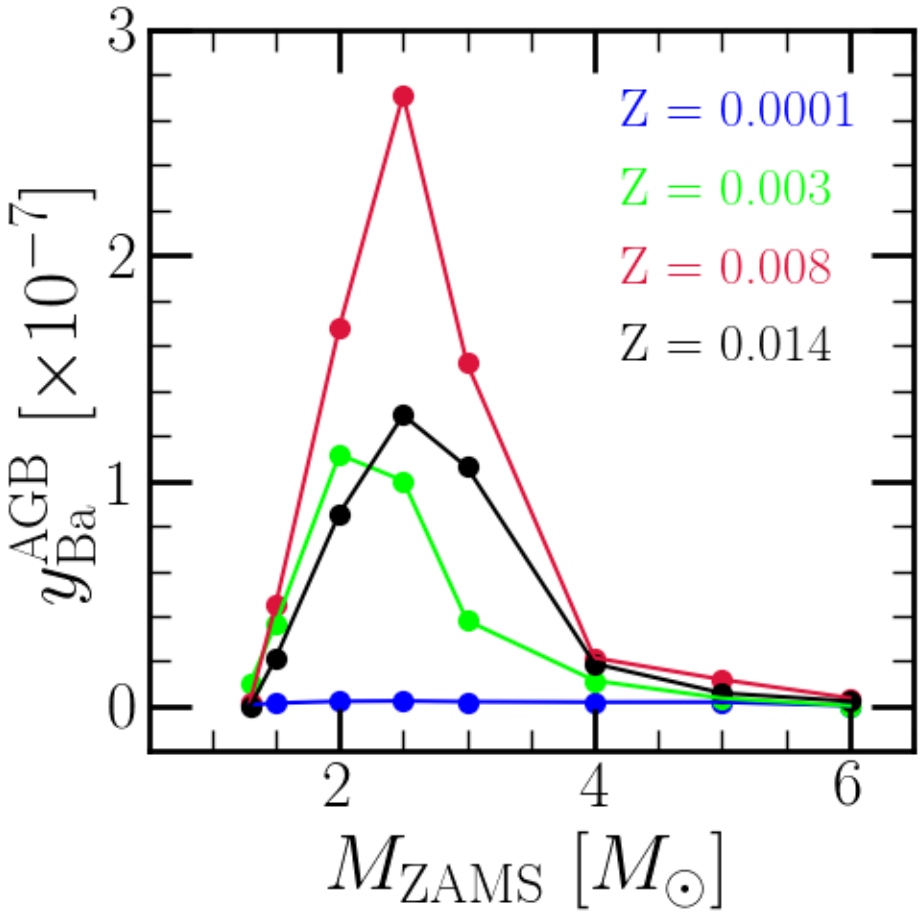

Fig. 5: The AGB star yields of barium (Ba) built into VICE as reported by Cristallo et al. (2011, 2015) for $Z = 0.0001$ (blue), 0.003 (bright green), 0.008 (dark red), and 0.014 (black) progenitors.

3.7.1 Stars

VICE adopts a star particle approach to the treatment of stars in multizone models. Users specify how many star particles should form per zone per timestep $n_\star$ via the attribute `n_stars` of the `multizone` class. The mass of a star particle is then given by:

$$M_\star \equiv \frac{\dot{M}_\star \Delta t}{n_\star}$$

where $\dot{M}_\star$ and Δt are the star formation rate and timestep size in the star particles zone of formation, respectively. In words, VICE divides the total mass of newly formed stars evenly amongst the star particles it should form. Rather than forming more/fewer star particles when the star formation rate is higher/lower, VICE forms star particles of varying mass. Star particles are also still formed when $\dot{M}_\star$ = 0; they simply have zero mass.

For a given star particle formed in a given zone and at a given time, the zone it occupies at subsequent times can be expressed as an arbitrary function of time S[1]. Allowing for the fact that star particles form in all zones and at all timesteps, the full stellar migration prescription can be expressed as a function of three variables $S(i, t_{\text{form}}, t)$ where i is the zone number the star particle forms in, t_{form} the time at which it forms, and t the time in the simulation (obviously, only times at which $t > t_{\text{form}}$ are relevant).

S is specified by the user. If the user does not specify a value of S, VICE adopts a default value which returns i at all times; that is, star particles remain in their zone of birth and do not migrate. If star particles forming in the same zone at the same time should have different zone histories, the user-defined function S can be specified to take a fourth parameter: a keyword argument `n`. VICE will then call the function with `n = 1`, `n = 2`, `n = 3`, and so on, up to `n = n_stars`, and users can assign zone histories to star particles based on the value of `n`.

Relevant Source Code:

- `vice/core/multizone/_migration.pyx`
- `vice/src/multizone/migration.c`

3.7.2 Gas

As discussed above, VICE knows nothing of the spatial configuration of the zones in a multizone model: the only piece of information identifying a zone is an integer index. Gas must then be able to move between zones in an arbitrary manner.

We thus define the gas migration matrix G_{ij} to denote the mass fraction of gas that moves *from* the i'th zone *to* the j'th zone in a 10 Myr time interval. We normalize to a specific time interval so that the rate of migration does not depend on the timestep size. The mass fraction of gas that migrates from zone i to zone j at a time t is then given by:

$$f_{ij}(t) = G_{ij}(t) \frac{\Delta t}{10 \text{ Myr}}$$

The mass that migrates is then given by $M_{g,i} f_{ij}(t)$, where $M_{g,i}$ is the total mass of the interstellar medium in zone i.

For a multizone simulation with N zones, G is an $N \times N$ matrix, the elements of which the user can fill with either numerical values denoting a constant rate of gas migration between zones or functions of time denoting a varying rate of gas migration. The diagonal elements G_{ii} are however irrelevant, because this corresponds to migration within the same zone; VICE forces these values to zero always.

Relevant Source Code:

- `vice/core/multizone/_migration.pyx`
- `vice/src/multizone/migration.c`

[1] Because Z for zone would cause confusion with the metallicity by mass Z, we choose S for star to denote this function instead.

3.8 Milky Way-Like Galaxies

VICE's `milkyway` object is designed handle multi-zone models of Milky Way-like galaxies in a flexible manner. Rather than requiring the user to construct them from scratch from the base `multizone` object, `milkyway` eases the burden by adopting a spatial configuration in which each zone represents an annulus of the Milky Way disk; these annuli are concentric from a radius $R = 0$ to 20 kpc. The width of each annulus ΔR is a value the user may set upon construction of a `milkyway` object. As defaults, it adopts an observationally-motivated star formation law and a stellar migration prescription based on N-body simulations. For an in-depth example of an application of the `milkyway` object, we refer users to the models of Johnson et al. (2021)[1], for which these features were designed.

The default stellar migration model of the `milkyway` object is implemented in the `vice.toolkit.hydrodisk.hydrodiskstars` object. This object is built around data from the `h277` simulation (Christensen et al. 2021)[2], a zoom-in hydrodynamical simulation ran from cosmological initial conditions which has made a number of appearances in the literature to date (e.g. Zolotov et al. 2012[3]; Loebman et al. 2012[4], 2014[5]; Brooks & Zolotov[6]; Bird et al. 2021[7]). Although `h277` is currently the only simulation whose data is available to the `hydrodiskstars` object, it's implementation could be extended to include others.

Note: The `h277` star particle data is not included in VICE's default distribution, but is available in its GitHub repository at vice/toolkit/hydrodisk/data. VICE will download these files automatically when a `milkyway` or `hydrodiskstars` object is created for the first time. With a decent internet connection, this process takes about one minute to complete, and does not need repeated. If this process fails, it may be due to not having administrator's privileges; users in this situation should speak with their administrator, who would then be able to download these data with the following few lines in `python`:

```
>>> import vice
>>> vice.toolkit.hydrodisk.data.download()
```

3.8.1 The Sample of Star Particles

The `hydrodiskstars` object, the default stellar migration prescription for the `milkyway` object, makes use of the birth radii, final radii, and birth times of star particles from hydrodynamical simulations, for which only the `h277` simulation is currently available (see above). `h277` did not record the birth radius of each star particle; however, each star particle does have an accurate age at each snapshot. The orbital radii of stars that are sufficiently young in their first snapshot should be good approximations of their birth radii since dynamical heating will have little effect in a short time interval. We have therefore restricted the sample of `h277` star particles in the `hydrodiskstars` object to those with an age at first snapshot less than 150 Myr, adopting their galactocentric radius at first snapshot as their birth radius. The choice of 150 Myr makes no significant impact on the predictions of the `milkyway` object (see discussion in section 2.1 of Johnson et al. 2021).

Of the star particles that remain after imposing this cut, the oldest one has an age of 13.23 Gyr. Since `h277` ran for ~13.7 Gyr, we have therefore subtracted 500 Myr from the birth times of all star particles, letting $T = 0$ in the `hydrodiskstars` and `milkyway` objects correspond to $T = 500$ Myr in `h277`, and placing the onset of star formation in these models at that time. As a consequence, these models support calculations of chemical evolution up to lookback times of 13.2 Gyr. Although this limit is not enforced in VICE, simulations on longer timescales using the `milkyway` object are highly likely to produce a `segmentation fault`.

[1] Johnson et al. (2021), MNRAS, 508, 4484
[2] Christensen et al. (2012), MNRAS, 425, 3058
[3] Zolotov et al. (2012), ApJ, 761, 71
[4] Loebman et al. (2012), ApJ, 758, L23
[5] Loebman et al. (2014), ApJ, 794, 151
[6] Brooks & Zolotov (2014), ApJ, 786, 87
[7] Bird et al. (2020), arxiv:2005.12948

We further restrict the sample of `h277` star particles to only those with both formation and final radii of $R \leq 20$ kpc, and to have formed within $|z| \leq 3$ kpc of the disk midplane. These criteria ensure that our sample reflects only the star particles that formed *in-situ*, and can therefore be described by a disc GCE model. Although it's possible some number of these star particles formed in a dwarf galaxy as it was being accreted by `h277`, these stars are few in number, and are only relevant at large radii and early times, where few stars form in nature anyway.

Based on a kinematic decomposition of these star particles, we exclude halo stars from the sample, but include those with bulge, pseudobulge, and disc-like kinematics. This ensures that all stars which can be attributed to the spatially confined regions reasonably defining a spiral galaxy disk can be modeled using the `hydrodiskstars` object. Altogether, these cuts yield a sample of 3,102,519 star particles from `h277`, accessible via the `analog_data` attribute of the `hydrodiskstars` object. For an analysis of the results of these cuts, we refer users to section 2.1 of Johnson et al. (2021).

3.8.2 Migration Models

As in many numerical models of galaxy evolution, stars in VICE are stand-ins for entire stellar populations. In the `milkyway` object (assuming the `hydrodiskstars` object is driving migration), they are said to be in a given zone if their radius is between the inner and outer edges of the annulus. At all times, VICE places their nucleosynthetic products and returned envelopes in the ISM of the annulus that they are in *at that time*.

The `hydrodiskstars` object assumes that star particles are born at the centers of their birth annuli. For a stellar population born at a time T and galactocentric radius R, it first searches for star particles in the `h277` sample (see above) which formed at $T\pm$ 250 Myr and $R\pm$ 250 pc. It then randomly selects a star particle from this subsample to act as an *analog*. The stellar population in the VICE model then adopts the change in orbital radius ΔR, and moves their with an assumed time-dependence (see below). If no candidate analogs are found, the `hydrodiskstars` object widens the search to $T\pm$ 500 Myr and $R\pm$ 500 pc. If still no analog is found, it maintains the $T \pm 500$ Myr criterion, but finds the one with the smallest difference in birth radius, assigning that star particle as the analog. While this prescription allows stellar populations to be assigned analogs with significantly birth radii, this is only an issue for small T and large R where there are few star particles from `h277`, and where few stars form in nature anyway. When an `h277` star particle is assigned as an analog, it is *not* thrown out of the sample of candidate analogs, in theory allowing a star particle to act as an analog for multiple stellar populations.

The `hydrodiskstars` object provides four models for the time-dependence of a star's radius between its birth and the present day. The first case is one in which stars remain at their birth radius until the present-day, at which time they instantly migrate; mixing is a post-processing prescription in this scenario. The second case is a generalization of this in which the sudden migration to the present-day radius occurs at some time randomly drawn between the birth time and the end of the simulation. The third is one in which the radius change with a $\sqrt{\text{age}}$ dependence, and the final is one with a linear dependence on time. These are the "post-processing", "sudden", "diffusion", and "linear" migration models from Johnson et al. (2021, see section 2.2 therein for further details). The `hydrodiskstars` object adopts "diffusion" as the default.

Here we illustrate these four migration models in the $R - T$ plane. While VICE's internal `h277` data supply a stellar population in VICE's models with ΔR, given one of these four assumptions and its birth radius (assumed to be in the center of its zone of birth), its radius at all remaining times is known. We emphasize that there is no N-body integration that goes into VICE's `milkyway` models. Although these are four built-in presets that users may choose from, they are not restricted to these options. A custom migration scheme based on the `h277` data can be implemented by subclassing the `hydrodiskstars` object, overriding its `__call__` function, and setting the attribute `mode` to `None`.

3.8.3 The Default Star Formation Law

As a default, the `milkyway` object adopts the `vice.toolkit.J21_sf_law` to describe the relation between the surface density of star formation $\dot{\Sigma}_\star$ and the surface density of the interstellar medium $\Sigma_{\rm g}$. This is also the star formation law adopted in Johnson et al. (2021). This star formation law is a broken power-law with two breaks; below $\Sigma_{\rm g} = 5 \times 10^6 M_\odot\ kpc^{-2}$, the relation scales as $\dot{\Sigma}_\star \propto \Sigma_{\rm g}^{1.7}$. Between $5 \times 10^6 M_\odot\ kpc^{-2}$ and $2 \times 10^7 M_\odot\ kpc^{-2}$, it scales as $\dot{\Sigma}_\star \propto \Sigma_{\rm g}^{3.6}$. Above $2 \times 10^7 M_\odot\ kpc^{-2}$, the relation becomes linear.

The `J21_sf_law` calculates the star formation efficiency timescale $\tau_\star$ (usually referred to as a "depletion time" in the star formation, feedback, and interstellar medium literatures) for use with the `singlezone` and `multizone` objects. This timescales is defined as the gas density per unit star formation: $\tau_\star \equiv \dot{\Sigma}_\star / \Sigma_{\rm g}$. To set the normalization of the star formation law, the `J21_sf_law` object assumes that in the linear regime, $\tau_\star = \tau_{\rm mol}$, the value of $\tau_\star$ for a star forming reservoir where hydrogen is entirely in the molecular phase. Below surface densities of $2 \times 10^7 M_\odot kpc^{-2}$, the timescale increases in a piece-wise continuous manner. The `J21_sf_law` object affords users the ability to modify the surface densities at which there are breaks in the power-law, as well as the power-law indeces themselves. For additional discussion, we refer users to section 2.5 of Johnson et al. (2021).

3.8.4 Additional Parameters

The `milkyway` object adopts a scaling of the mass loading factor $\eta \equiv \dot{M}_{\rm out} / \dot{M}_\star$ with galactocentric radius. The scaling is tuned such that the equilibrium abundances as a function of radius reflect a reasonable metallicity gradient in agreement with observational results from APOGEE (see section 2.3 of Johnson et al. 2021). The default scaling is based on alpha-elements (e.g., O, Ne, Mg) under a constant star formation history. The slope of the gradient is assumed to be mode([α/H]) $\propto$ -0.08 kpc^{-2}, with the normalization set by mode([α/H]) = +0.3 at R = 4 kpc. This default scaling is implemented via the function `vice.milkyway.default_mass_loading`, and like the star formation law and stellar migration prescription, is only a default and can be overridden by the user if they so choose.

The `milkyway` object does not have any treatment for vertical structure of the star forming disk; that is, the boundaries between zones are purely radial. There are no zones off the disk midplane. This implicitly assumes that the star forming reservoir is well-mixed in the azimuthal and vertical directions, and that significant abundance differences occur only in the radial direction. By default it also neglects gas migration, because the Johnson et al. (2021) models for which it was designed focused instead on the impact of varying assumptions about stellar migration.

For further discussion of the `milkyway` object, we refer users to section 2 of Johnson et al. (2021).

Relevant Source Code

- `vice/milkyway/milkyway.py`
- `vice/toolkit/J21_sf_law.py`
- `vice/toolkit/hydrodisk/hydrodiskstars.py`
- `vice/toolkit/hydrodisk/_hydrodiskstars.pyx`
- `vice/src/toolkit/hydrodiskstars.c`

[8] Johnson et al. (2021), MNRAS, 508, 4484

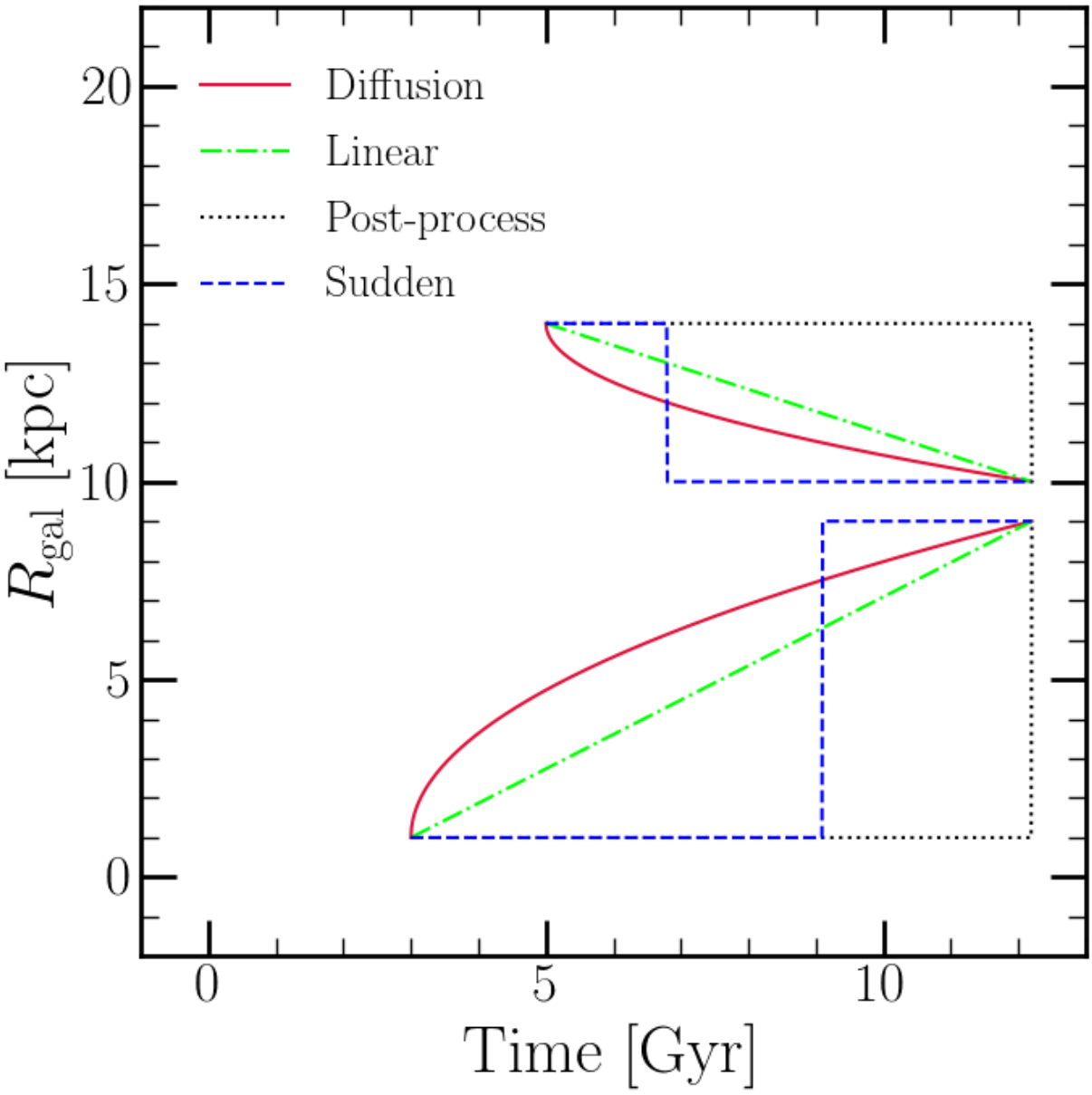

Fig. 6: The four models for the time-dependence of radial migration adopted by VICE's `hydrodiskstars` object. This is Fig. 2 of Johnson et al. (2021)[8], which investigated the impact of these models on the model-predicted abundances. With the birth radius of a stellar population assumed to be at the center of its annulus of birth, a final radius implied by the ΔR of the assigned `h277` star particle, and one of these assumptions, the radius at all times is known.

3.9 Scaling of the Total Metallicity

VICE quantifies the total metallicity by mass of both gas and stars in VICE according to:

$$Z = Z_\odot \frac{\sum_i Z_i}{\sum_i Z_{i,\odot}}$$

where the summation is taken over all elements tracked by the simulation. This is motivated by numerical artifacts that would be introduced into metallicity dependent quantities when only a small number of elements are being simulated. For example, if there are only three elements in a simulation and they are all near the solar abundance, this scaling ensures that metallicity dependent yields will behave as if the metallicity is near solar, as opposed to the much lower total metallicity of only three elements.

This is where the user's adopted solar metallicity $Z_\odot$ enters in their simulations. Because the element-by-element breakdown of the solar composition $Z_{i,\odot}$ is taken from Asplund et al. (2009)[1], we recommend adopting $Z_\odot = 0.014$ from their findings for a self-consistent scaling.

The total logarithmic metallicity $[M/H]$ relative to the sun is then evaluated according to:

$$[M/H] = \log_{10}\left(\frac{Z}{Z_\odot}\right) = \log_{10}\left(\sum_i Z_i\right) - \log_{10}\left(\sum_i Z_{i,\odot}\right)$$

Note: These quantities are not recorded with outputs in order to minimize write-out time when the number of elements is high. Instead, `history` and `tracer` objects evaluate these equations automatically for gas and stars, respectively.

3.10 Stellar Metallicity Distribution Functions

VICE's `singlezone` and `multizone` objects automatically determine normalized stellar metallicity distribution functions (MDFs) for each simulation. The MDF, in its most general form, is given by:

$$\frac{dN}{dZ} = \frac{\dot{N}}{\dot{Z}} \propto \frac{\dot{M}_\star}{\dot{Z}}$$

This is fairly intuitive; the number of stars that form at a metallicity $\approx$ Z is proportional to the star formation rate at that time and inversely related to the rate at which the metallicity is evolving away from that value. VICE converts MDFs to probability distribution functions by ensuring that the integral over the bins is equal to one:

$$\frac{dN}{d[X/Y]} \to \frac{dN/d[X/Y]}{\int dN} = \frac{dN/d[X/Y]}{\int_{-\infty}^{\infty} \frac{dN}{d[X/Y]} d[X/Y]}$$

Note: In its current version, VICE only reports MDFs at the final timestep of the simulation.

In practice, the user specifies an array of bin-edges that they would like the MDF sorted into, and VICE creates arrays of zeroes whose lengths are the number of bins in the user's array. In a singlezone simulation, the appropriate bins for each combination of [X/H] and [X/Y] are incremented by the star formation rate. At the final timestep, the normalization of the i'th bin is then approximated numerically by:

$$\frac{\Delta N_i}{\Delta[X/Y]_i} \to \frac{\Delta N_i / \Delta[X/Y]_i}{\sum_j \frac{\Delta N_j}{\Delta[X/Y]_j} \Delta[X/Y]_j} = \frac{\Delta N_i / \Delta[X/Y]_i}{\sum_j \Delta N_j}$$

[1] Asplund et al. (2009), ARA&A, 47, 481

The fraction of stars in a given range $\Delta[X/Y]$ is then given by the value of the reported MDF times $\Delta[X/Y]$.

In a multizone simulation, the metallicity distribution function is calculated directly from the star particles that are in a given zone at a given time. For each star particle in a given zone, the appropriate bins in [X/H] and [X/Y] are incremented by the mass of the star rather than by the star formation rate at previous timesteps. The same normalization process is then applied.

CHAPTER

FOUR

COMPREHENSIVE API REFERENCE

4.1 From the Command Line

Included with VICE is a command line entry which runs simple one-zone models from a terminal. This feature allows the parameters of the model to be specified as command-line arguments; for usage guidelines, run `python3 -m vice --help` from a terminal after installing VICE (from any directory except the source tree, if installed from source). While these command-line capabilities are useful for their ease, VICE is severaly limited in capability when ran from the command-line in comparison to when ran from the python interpreter.

This same command-line entry can be used for automatic access to the tutorial and the documentation. The commands are `python3 -m vice --docs` and `python3 -m vice --tutorial`.

These features can also be accessed via the simpler command `vice` (e.g. `python3 -m vice --docs` should do the same thing as `vice --docs`).

4.2 Package Contents

VICE: Versatile Integrator for Chemical Evolution

- 77 elements on the periodic table
- Fast integration of one-zone models
- Enrichment from single stellar populations
- Highly flexible nucleosynthetic yield calculations
- **User-defined mathematical forms describing:**
    - Nucleosynthetic yields in simulations
    - Mixing processes in multi-zone models
    - Infall and star formation histories
    - The stellar initial mass function
    - The star formation law
    - Element-by-element infall metallicities
    - Type Ia supernova delay-time distributions

4.2.1 How to Access the Documentation:

Documentation is available in several forms:

1. Online: http://vice-astro.readthedocs.io
2. In PDF format, available for download at the same address
3. In the docstrings embedded within the software

Running `vice --docs` from the terminal will open the online documentation in the default web browser.

First time users should go through VICE's QuickStartTutorial jupyter notebook, available under examples/ in the git repository. This can be launched from the command line by running `vice --tutorial`. Other example scripts can be found there as well.

4.2.2 Contents

singlezone [`object`] Simulate a single-zone galactic chemical evolution model

multizone [`object`] Simulate a multi-zone galactic chemical evolution model

milkyway [`object`] A `multizone` object optimized for modeling the Milky Way.

output [`object`] Read and store output from `singlezone` simulations.

multioutput [`object`] Read and store output from `multizone` simulations.

migration [<module>] Utilities for mixing prescriptions in multizone simulations.

single_stellar_population [<function>] Simulate enrichment from a single conatal star cluster

cumulative_return_fraction [<function>] Calculate the cumulative return fraction of a star cluster of known age

main_sequence_mass_fraction [<function>] Calculate the main sequence mass fraction of a star cluster of known age

imf [<module>] Built-in funcitonal forms of popular stellar initial mass functions.

mlr [`object`] Built-in popular function forms of the stellar mass-lifetime relationship. Also stores which form to adopt in chemical evolution models.

yields [<module>] Calculate, access, and declare nucleosynthetic yield settings for use in simulations.

elements [<module>] Access, and declare nucleosynthetic yield settings for use in simulations. Access other relevant information for each element such as the solar abundance or atomic number.

dataframe [`object`] A dictionary-like object with case-insensitive lookup and data storage.

history [<function>] Reads in time-evolution of interstellar medium from singlezone simulation.

mdf [<function>] Reads in stellar metallicity distribution from singlezone simulation.

stars [<function>] Read in stellar population abundances from a multizone simulation output.

toolkit [<module>] Generally useful utilities.

_dev [<module>] Developer's tools.

4.2.3 Built-In Dataframes

- atomic_number : The atomic number of each element
- primordial : The abundance of each element following big bang nucleosynthesis.
- solar_z : The abundance of each element in the sun.
- sources : The primary astrophysical production channels of each element.
- stable_isotopes : Lists of each elements' stable isotopes.

4.2.4 Utilities

- VisibleDeprecationWarning : A DeprecationWarning that is visible by default.
- VisibleRuntimeWarning : A RuntimeWarning that is visible by default.
- ScienceWarning : A Warning concerning scientific accuracy and precision.
- test : Runs VICE's unit tests.
- version : VICE's version breakdown.
- __version__ : The version string.

vice.version

VICE Version Information

In keeping with convention, VICE's version string can be accessed via `vice.__version__`. Alternatively, this object can simply be type-casted to a string via `str(vice.version)`.

VICE records its version number according to the semantic versioning method described in PEP 440[1].

Attributes

major [`int`] The major version number of this release.

minor [`int`] The minor version number of this release.

micro [`int`] The micro version number of this release (also known as patch number).

dev [`int`] The development version number of this release. `None` if this is not a development release.

alpha [`int`] The alpha version number of this release. `None` if this is not an alpha release.

beta [`int`] The beta version number of this release. `None` if this is not a beta release.

rc [`int`] The release candidate number of this release. `None` if this is not a release candidate.

post [`int`] The post number of this release. `None` if this is not a post release.

isreleased [`bool`] Whether or not this version has been released. If False, users are advised to contact a contributor to VICE if they are not a contributor themselves.

Note: At most one of the attributes `dev`, `alpha`, `beta`, `rc`, and `post` will not be `None`.

---

[1] https://www.python.org/dev/peps/pep-0440/

Notes

This object can be type-cast to a tuple of the form:

```
(major, minor, micro, dev, alpha, beta, rc, post)
```

Alternatively, the information can be obtained in dictionary format via `vice.version.todict()`.

vice.atomic_number

The VICE dataframe: atomic numbers

Stores the proton numbers of each recognized element. Stored values are of type `int`.

Indexing

- **`str` [case-insensitive]** The symbol of a chemical element as it appears on the periodic table.

Item Assignment

This instance of the VICE dataframe does not support item assignment.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> vice.atomic_number['o']
8
>>> vice.atomic_number['fe']
26
>>> vice.atomic_number['sr']
38
```

vice.primordial

The VICE dataframe: primordial abundances

Stores the abundance by mass of each element following big bang nucleosynthesis. Stored values are of type `float`, and are zero for all elements with the exception of helium, which is assigned a value of $Y_{\rm p} = 0.24721 \pm 0.00014$ (Pitrou et al. 2021[1]).

New in version 1.1.0: Previous versions of VICE did not implement helium, and therefore did not require any information on primordial abundances.

[1] Pitrou et al. (2021), MNRAS, 502, 2474

Indexing

- **str [case-insensitive]** The symbol of a chemical element as it appears on the periodic table.

Item Assignment

This instance of the VICE dataframe does not support item assignment.

Functions

- keys
- todict

Notes

In versions >= 1.3.0, the primordial abundance of helium is taken to be $Y_\mathrm{p} = 0.24721 \pm 0.00014$ (Pitrou et al. 2021), which updates the value of $Y_\mathrm{p} = 0.24672 \pm 0.00017$ (Planck Collaboration et al. 2016[2]; Pitrou et al. 2018[3]; Pattie et al. 2018[4]) from previous versions of VICE based on updates to the neutron lifetime and the D$(p, \gamma)^3$He reaction.

Example Code

```
>>> import vice
>>> vice.primordial['o']
0.0
>>> vice.primordial['he']
0.24721
>>> vice.primordial['c']
0.0
```

vice.solar_z

The VICE dataframe: solar composition

Stores the abundance by mass of all recognized elements in the sun. Stored values are of type `float`; defaults are assigned according to the Asplund et al. (2009)[1] photospheric measurements. For elements where the photospheric measurements were not feasible, this object adopts the meteoritic measurements.

New in version 1.2.0: In versions >= 1.2.0, users may modify the values stored for each individual element. The only restriction imposed is that the values be between 0 and 1. In prior versions, the values stored by this dataframe were not modifiable.

[2] Planck Collaboration et al. (2016), A&A, 594, A13
[3] Pitrou et al. (2018), Phys. Rep., 754, 1
[4] Pattie et al. (2018), Science, 360, 627
[1] Asplund et al. (2009), ARA&A, 47, 481

Indexing

- **str [case-insensitive]** The symbol of a chemical element as it appears on the periodic table.

Item Assignment

- **float** The new abundance by mass (i.e. the mass fraction) of a given element within the sun. Must be between 0 and 1.

Notes

Changes to the values stored by this object will be reflected in any chemical evolution simulations ran by VICE. However, these values will reset every time the python interpreter is restarted.

Default values are calculated with `vice.solar_z.epsilon_to_z_conversion`. For details, see the associated documentation.

For helium ('he'), the default value is modified from the photospheric abundance of He as measured by Asplund et al. (2009) (Y = 0.2485) to their recommended bulk abundance (Y = 0.2703) (see their section 3.12).

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> vice.solar_z['o']
0.00572
>>> vice.solar_z['o'] = 0.005
>>> vice.solar_z['o']
0.005
>>> vice.solar_z['c']
0.00236
>>> vice.solar_z['c'] *= 1.1 # increase by 10 percent
0.0025960000000000002
```

vice.solar_z.epsilon_to_z_conversion

Convert an element's abundance in the sun according to the definition

$$\epsilon_x = \log_{10}(N_x/N_H) + 12$$

to a metallicity by mass $Z_x = M_x/M_\odot$.

Signature: vice.solar_z.epsilon_to_z_conversion(epsilon, mu, Xsun = 0.73)

New in version 1.2.0.

Parameters

epsilon [float] The log-scaled number density relative to hydrogen defined above for a particular element.

mu [float] The mean molecular weight of the element.

Xsun [float [default][0.73]] The hydrogren mass fraction of the solar photosphere.

Returns

Zsun [float] The abundance by mass of the element x in the sun, $M_x/M_\odot$.

Notes

The values returned by this function are calculated according to the following conversion, which can be derived from the definition of ϵ_x and Z_x above:

$$Z_x = X_\odot \mu_x 10^{\epsilon_x - 12}$$

VICE uses this function to compute the default solar composition based on internal data storing the Asplund et al. (2009)[1] photospheric measurements.

vice.sources

The VICE dataframe: nucleosynthetic sources

Stores the dominant astrophysical enrichment channels of each element. These values are adopted from Johnson (2019)[1]. Stored values are of type `list`, elements of which are of type `str`.

- **"BBN": Big Bang Nucleosynthesis** A statistically significant portion of this element's present-day abundances was present prior to the onset of star formation in the universe.
- **"CCSNE": Core Collapse Supernovae** A statistically significant portion of this element's present-day abundances is due to massive star explosions.
- **"SNEIA": Type Ia Supernovae** A statistically significant portion of this element's present-day abundances is due to white dwarf explosions.
- **"AGB": Asymptotic Giant Branch Stars** A statistically significant portion of this element's present-day abundances is due to synthesis in AGB stars.
- **"NSNS": Neutron Star Mergers, R-process Nucleosynthesis** A statistically significant portion of this element's present-day abundances is due to neutron star mergers, or other astrophysical sites of r-process nucleosynthesis.

[1] Asplund et al. (2009), ARA&A, 47, 481
[1] Johnson (2019), Science, 363, 474

Indexing

- **str [case-insensitive]** The symbol of a chemical element as it appears on the periodic table.

Item Assignment

This instance of the VICE dataframe does not support item assignment.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> vice.sources['o']
["CCSNE"]
>>> vice.sources['he']
["BBN", "CCSNE", "AGB"]
>>> vice.sources['fe']
["CCSNE", "SNEIA"]
```

vice.stable_isotopes

The VICE dataframe: stable isotopes

The mass number (protons and neutrons) of the stable isotopes of each element. Stored values are of type `list`, elements of which are of type `int`.

New in version 1.1.0.

Indexing

- **str [case-insensitive]** The symbol of a chemical element as it appears on the periodic table.

Item Assignment

This instance of the VICE dataframe does not support item assignment.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> vice.stable_isotopes['he']
[3, 4]
>>> vice.stable_isotopes['o']
[16, 17, 18]
>>> vice.stable_isotopes['fe']
[54, 56, 57, 58]
```

vice.cumulative_return_fraction

Calculate the cumulative return fraction for a single stellar population at a given age. This quantity represents the fraction of the stellar population's mass that is returned to the interstellar medium as gas at the birst metallicity of the stars.

Signature: vice.cumulative_return_fraction(age, IMF = "kroupa", m_upper = 100, m_lower = 0.08, postMS = 0.01)

Parameters

age [real number] The age of the stellar population in Gyr.

IMF [`str` [case-insensitive] or <`function`> [default]["kroupa"]] The assumed stellar initial mass function (IMF). Strings denote built-in IMFs. Functions must accept only one numerical parameter and will be interpreted as a custom, arbitrary stellar IMF.

Recognized built-in IMFs:

- Kroupa[1]
- Salpeter[2]

Note: Functions do not need to be normalized. VICE will take care of this automatically.

---

m_upper [real number [default][100]] The upper mass limit on star formation in solar masses.

m_lower [real number [default][0.08]] The lower mass limit on star formation in solar masses.

postMS [real number [default][0.1]] The ratio of a star's post main sequence lifetime to its main sequence lifetime.

New in version 1.1.0: Prior to version 1.1.0, VICE approximated postMS = 0.

[1] Kroupa (2001), MNRAS, 231, 322
[2] Salpeter (1955), ApJ, 121, 161

Returns

crf [real number] The value of the cumulative return fraction for a stellar population at the specified age under the specified parameters.

Notes

Note: VICE operates under the approximation that stars have a mass-luminosity relationship given by:

$$L \sim M^{4.5}$$

leading to a mass-lifetime relation that is also a power law, given by:

$$\tau \sim M/L \sim M^{-3.5}$$

Note: VICE implements the remnant mass model of Kalirai et al. (2008)[3], assuming that stars above 8 $M_\odot$ leave behind remnants of 1.44 $M_\odot$, while stars below 8 $M_\odot$ leave behind remnants of $0.394M_\odot + 0.109M$.

Raises

- **TypeError**
  - age is not a real number
  - IMF is neither a string nor a function
  - m_upper is not a real number
  - m_lower is not a real number
  - postMS is not a real number
- **ValueError**
  - age < 0
  - built-in IMF is not recognized
  - m_upper <= 0
  - m_lower <= 0
  - m_lower >= m_upper
  - postMS < 0 or > 1

[3] Kalirai et al. (2008), ApJ, 676, 594

Example Code

```
>>> vice.cumulative_return_fraction(1)
        0.3560160079575864
>>> vice.cumulative_return_fraction(2)
        0.38056657042902253
>>> vice.cumulative_return_fraction(3)
        0.394760119115021
```

vice.main_sequence_mass_fraction

Calculate the main sequence mass fraction for a single stellar population at a given age. This quantity represents the fraction of the stellar population's mass that is still in the form of stars on the main sequence.

Signature: vice.main_sequence_mass_fraction(age, IMF = "kroupa", m_upper = 100, m_lower = 0.08)

Parameters

age [real number] The age of the stellar population in Gyr.

IMF [`str` [case-insensitive] or <`function`> [default]["kroupa"]] The assumed stellar initial mass function (IMF). Strings denote built-in IMFs. Functions must accept only one numerical parameter and will be interpreted as a custom, arbitrary stellar IMF.

Recognized built-in IMFs:

- Kroupa[1]
- Salpeter[2]

Note: Functions do not need to be normalized. VICE will take care of this automatically.

m_upper [real number [default][100]] The upper mass limit on star formation in solar masses.

m_lower [real number [default][0.08]] The lower mass limit on star formation in solar masses.

Returns

msmf [real number] The value of the main sequence mass fraction for a stellar population at the specified age under the specified parameters.

[1] Kroupa (2001), MNRAS, 231, 322
[2] Salpeter (1955), ApJ, 121, 161

Notes

Note: VICE operates under the approximation that stars have a mass-luminosity relationship given by:

$$L \sim M^{4.5}$$

leading to a mass-lifetime relation that is also a power-law, given by:

$$\tau \sim M/L \sim M^{-3.5}$$

Raises

- **TypeError**
  - age is not a real number
  - IMF is neither a string nor a function
  - m_upper is not a real number
  - m_lower is not a real number
  - postMS is not a real number
- **ValueError**
  - age < 0
  - built-in IMF is not recognized
  - m_upper <= 0
  - m_lower <= 0
  - m_lower >= m_upper

Example Code

```
>>> vice.main_sequence_mass_fraction(1)
        0.5815004968281556
>>> vice.main_sequence_mass_fraction(2)
        0.5445877675278488
>>> vice.main_sequence_mass_fraction(3)
        0.5219564300200146
```

vice.single_stellar_population

Simulate the nucleosynthesis of a given element from a single star cluster of given mass and metallicity. This does not take into account galactic evolution - whether or not it is depleted from inflows or ejected in winds is not considered. Only the net mass of the given element produced by the star cluster is calculated.

Signature: vice.single_stellar_population(element, mstar = 1.0e+06, Z = 0.014, time = 10, dt = 0.01, m_upper = 100, m_lower = 0.08, postMS = 0.1, IMF = "kroupa", RIa = "plaw", delay = 0.15)

Parameters

element [`str` [case-insensitive]] The symbol of the element to simulate the enrichment for.

mstar [real number [default][1.0e+06]] The birth mass of the star cluster in solar masses.

Z [real number [default][0.014]] The metallicity by mass of the stars in the cluster.

time [real number [default][10]] The amount of time in Gyr to run the simulation for.

dt [real number [default][0.01]] The size of each timestep in Gyr.

m_upper [real number [default][100]] The upper mass limit on star formation in solar masses.

m_lower [real number [default][0.08]] The lower mass limit on star formation in solar masses.

postMS [real number [default][0.1]] The ratio of a star's post main sequence lifetime to its main sequence lifetime.

New in version 1.1.0: Prior to version 1.1.0, VICE approximated postMS = 0.

IMF [`str` [case-insensitive] or <`function`> [default]["kroupa"]] The stellar initial mass function (IMF) to assume. Strings denote built-in IMFs. Functions must accept only one numerical parameter and will be interpreted as a custom, arbitrary stellar IMF.

Recognized built-in IMFs:

- Kroupa[1]
- Salpeter[2]

Note: Functions do not need to be normalized. VICE will take care of this automatically.

RIa [`str` [case-insensitive] or <`function`> [default]["plaw"]] The delay-time distribution for type Ia supernovae to adopt. Strings denote built-in distributions. Functions must accept only one numerical parameter and will be interpreted as a custom, arbitrary delay-time distribution.

Recognized built-in distributions:

- "plaw": $R_{\rm Ia} \sim t^{-1.1}$
- "exp": $R_{\rm Ia} \sim e^{-t/1.5\ {\rm Gyr}}$

Note: Functions do not need to return 0 at times smaller than the SN Ia minimum delay time. VICE will take care of this automatically.

Note: Functions do not need to be normalized. VICE will take care of this automatically.

---

[1] Kroupa (2001), MNRAS, 231, 322
[2] Salpeter (1955), ApJ, 121, 161

delay [real number [default][0.15]] The minimum delay time following the formation of a single stellar population before the onset of type Ia supernovae in Gyr.

agb_model [string [case-insensitive] or None [default][None]] **[DEPRECATED]**

A keyword denoting which table of nucleosynthetic yields from AGB stars to adopt.

Recognized Keywords:

- "cristallo11"[3]
- "karakas10"[4]

Deprecated since version 1.2.0: Users should instead modify their AGB star yield settings through `vice.yields.agb.settings`. Users may specify either a built-in study or a function of stellar mass and metallicity.

Returns

mass [list] The net mass of the element in solar mass produced by the star cluster at each timestep.

times [list] The times in Gyr corresponding to each mass yield.

Raises

- **ValueError**
  - The element is not built into VICE.
  - mstar < 0
  - Z < 0
  - time < 0 or time > 15 [VICE does not simulate enrichment on timescales significantly longer than the age of the universe]
  - dt < 0
  - m_upper < 0
  - m_lower < 0
  - m_lower > m_upper
  - postMS < 0 or > 1
  - built-in IMF is not recognized
  - delay < 0
  - agb_model is not built into VICE
- **LookupError**
  - agb_model == "karakas10" and the atomic number of the element is larger than 29. The Karakas (2010), MNRAS, 403, 1413 study did not report yields for elements heavier than nickel.
- **ArithmeticError**
  - A functional RIa evaluated to a negative value, inf, or NaN at any given timestep.
- **IOError [Only occurs if VICE's file structure has been modified]**

[3] Cristallo et al. (2011), ApJS, 197, 17
[4] Karakas (2010), MNRAS, 403, 1413

- The AGB yield file is not found.

Example Code

```
>>> mass, times = vice.single_stellar_population("sr", Z = 0.008)
>>> mass[-1]
        0.04808964406448721
>>> mass, times = vice.single_stellar_population("fe")
>>> mass[-1]
        2679.816051685778
```

vice.mlr

The Mass-Lifetime Relationship (MLR) for Stars: VICE provides several functional forms available for individual calculations as well as for use in chemical evolution models.

Signature: vice.mlr

New in version 1.3.0.

Contents

setting [str] A string denoting which of the following functional forms is to describe the MLR in all chemical evolution models.

recognized [tuple] A tuple of strings denoting the allowed values of the parameter `setting`. Each string corresponds directly to the name of the function to adopt.

- "powerlaw"
- "vincenzo2016"
- "hpt2000"
- "ka1997"
- "pm1993"
- "mm1989"
- "larson1974"

powerlaw [<function>] The MLR parameterized by a single power-law, a popular exercise in undergraduate astronomy courses.

vincenzo2016 [<function>] The MLR as characterized by Vincenzo et al. (2016)[1].

hpt2000 [<function>] The MLR as described in Hurley, Pols & Tout (2000)[2].

ka1997 [<function>] The MLR as tabulated in Kodama & Arimoto (1997)[3].

pm1993 [<function>] The MLR as formulated by Padovani & Matteucci (1993)[4].

mm1989 [<function>] The MLR as characterized by Maeder & Meynet (1989)[5].

[1] Vincenzo et al. (2016), MNRAS, 460, 2238
[2] Hurley, Pols & Tout (2000), MNRAS, 315, 543
[3] Kodama & Arimoto (1997), A&A, 320, 41
[4] Padovani & Matteucci (1993), ApJ, 416, 26
[5] Maeder & Meynet (1989), A&A, 210, 155

larson1974 [<function>] The MLR as parameterized by Larson (1974)[6].

test [<function>] Run unit-tests on VICE's MLR capabilities.

The following forms of the mass-lifetime relation take into account the metallicity dependence:

- "vincenzo2016" : Vincenzo et al. (2016)
- "hpt2000" : Hurley, Pols & Tout (2000)
- "ka1997" : Kodama & Arimoto (1997)

The following require numerical solutions to the inverse function (i.e. stellar mass as a function of lifetime), and consequently can increase the required integration times in chemical evolution models, particularly for fine timestepping:

- "hpt2000" : Hurley, Pols & Tout (2000)
- "ka1997" : Koama & Arimoto (1997)
- "mm1989" : Maeder & Meynet (1989)

The following quantify the total lifetimes *a priori*, and any prescription for the post main sequence lifetimes will consequently be neglected in chemical evolution models:

- "vincenzo2016": Vincenzo et al. (2016)
- "ka1997" : Kodama & Arimoto (1997)

Except where measurements of the total lifetimes are available, VICE always implements the simplest assumption of allowing the user to specify the parameter `postMS` describing the ratio of post main sequence to main sequence lifetimes, and the total lifetime then follows trivially via:

$$\tau_{\text{total}} = (1 + p_{\text{MS}})\tau_{\text{MS}}$$

where p_{MS} denotes this ratio.

Note: For reasons relating to the implementation, this set of functions is not a module but an object. Consequently, importing them with `from vice import mlr` will work fine, but for example `from vice.mlr import vincenzo2016` will produce a `ModuleNotFoundError`. If necessary, new variables can always be assigned to map to these functions (e.g. `vincenzo2016 = vice.mlr.vincenzo2016`).

---

vice.mlr.setting

Type : str [assignment is case-insensitive]

Default : "larson1974"

New in version 1.3.0.

A keyword denoting which functional form of the mass-lifetime relation to adopt in chemical evolution models. Allowed keywords and the journal references for them:

- "powerlaw" : N/A
- "vincenzo2016": Vincenzo et al. (2016)[1]
- "hpt2000": Hurley, Pols & Tout (2000)[2]
- "ka1997": Kodama & Arimoto (1997)[3]

[6] Larson (1974), MNRAS, 166, 585
[1] Vincenzo et al. (2016), MNRAS, 460, 2238
[2] Hurley, Pols & Tout (2000), MNRAS, 315, 543
[3] Kodama & Arimoto (1997), A&A, 320, 41

- "pm1993": Padovani & Matteucci (1993)[4]
- "mm1989": Maeder & Meynet (1989)[5]
- "larson1974": Larson (1974)[6]

See also:

- vice.mlr.powerlaw
- vice.mlr.vincenzo2016
- vice.mlr.hpt2000
- vice.mlr.ka1997
- vice.mlr.pm1993
- vice.mlr.mm1989
- vice.mlr.larson1974

Note: Though assignment of this object is case-insensitive, the value stored will always be lower-cased. See example below assigning the Kodama & Arimoto (1997) MLR as the setting.

Example Code

```
>>> import vice
>>> vice.mlr.setting # the default setting
"larson1974"
>>> vice.mlr.setting = "KA1997"
>>> vice.mlr.setting
"ka1997"
>>> vice.mlr.setting = "hpt2000"
>>> vice.mlr.setting
"hpt2000"
```

vice.mlr.recognized

Type : `tuple` [elements of type `str`]

Value : ("powerlaw", "vincenzo2016", "hpt2000", "ka1997", "pm1993", "mm1989", "larson1974")

The allowed values of the parameter `vice.mlr.setting`.

[4] Padovani & Matteucci (1993), ApJ, 416, 26
[5] Maeder & Meynet (1989), A&A, 210, 155
[6] Larson (1974), MNRAS, 166, 585

Example Code

```
>>> import vice
>>> vice.mlr.recognized
("powerlaw",
 "vincenzo2016",
 "hpt2000",
 "ka1997",
 "pm1993",
 "mm1989",
 "larson1974")
>>> "hpt2000" in vice.mlr.recognized
True
>>> "name2003" in vice.mlr.recognized
False
```

vice.mlr.powerlaw

Compute either the lifetime or the mass of a dying star according to a single power-law relationship between the two.

Signature: vice.mlr.powerlaw(qty, postMS = 0.1, which = "mass")

New in version 1.3.0.

Parameters

qty [float] Either the mass of a star in $M_\odot$ or the age of a stellar population in Gyr. Interpretation set by the keyword argument `which`.

postMS [float [default][0.1]] The ratio of a star's post main sequence lifetime to its main sequence lifetime. Zero to compute the main sequence lifetime alone, or the main sequence turnoff mass when `which == "age"`.

which [str [case-insensitive] [default]["mass"]] The interpretation of `qty`: either `"mass"` or `"age"` (case-insensitive). If `which == "mass"`, then `qty` represents a stellar mass in $M_\odot$ and this function will compute a lifetime in Gyr. Otherwise, `qty` represents the age of a stellar population and the mass of a star with the specified lifetime will be calculated.

Returns

x [float] If `which == "mass"`, the lifetime of a star of that mass in Gyr according to the single power law. If `which == "age"`, the mass of a star in $M_\odot$ with the specified lifetime in Gyr.

Notes

This power-law is of the following form:

$$\tau = (1 + p_{\rm MS})\tau_\odot \left(\frac{M}{M_\odot}\right)^{-\gamma}$$

where $\tau_\odot$ is the main sequence lifetime of the sun (taken to be 10 Gyr), $p_{\rm MS}$ is the parameter `postMS`, and γ is the power-law index, taken to be 3.5.

This form of the mass-lifetime relation can be derived from the scaling relation:

$$\frac{\tau}{\tau_\odot} \sim \frac{M/M_\odot}{L/L_\odot}$$

where L is the luminosity of a star assuming $L \sim M^{4.5}$, a popular exercise in undergraduate astronomy courses.

The timescale τ quantifies only the main sequence lifetime of stars; the parameter `postMS` specifies the length of the post main sequence lifetime. This parameterization neglects the metallicity dependence of the mass-lifetime relation.

Example Code

```
>>> import vice
>>> vice.mlr.powerlaw(1) # the lifetime of the sun
11.0
>>> vice.mlr.powerlaw(1, postMS = 0) # main sequence lifetime only
10.0
>>> vice.mlr.powerlaw(1, which = "age") # what mass lives 1 Gyr?
1.9839958856298403
>>> vice.mlr.powerlaw(2, which = "age") # 2 Gyr?
1.627541971155844
>>> vice.mlr.powerlaw(3, which = "age") # 3 Gyr?
1.449507306037525
>>> vice.mlr.powerlaw(3, postMS = 0, which = "age") # MS turnoff mass
1.4105676750826
>>> vice.mlr.powerlaw(3, postMS = 0)
0.21383343303319474
```

vice.mlr.vincenzo2016

Compute either the lifetime or the mass of a dying star according to the mass-lifetime relation of Vincenzo et al. (2016)[1].

Signature: vice.mlr.vincenzo2016(qty, Z = 0.014, which = "mass")

New in version 1.3.0.

[1] Vincenzo et al. (2016), MNRAS, 460, 2238

Parameters

qty [float] Either the mass of a star in $M_\odot$ or the age of a stellar population in Gyr. Interpretion set by the keyword argument `which`.

Z [float [default][0.014]] The metallicity by mass of the stellar population.

which [str [case-insensitive] [default][“mass”]] The interpretation of `qty`: either `"mass"` or `"age"` (case-insensitive). If `which == "mass"`, then `qty` represents a stellar mass in $M_\odot$ and this function will compute a lifetime in Gyr. Otherwise, `qty` represents the age of a stellar population and the mass of a star with the specified lifetime will be calculated.

Returns

x [float] If `which == "mass"`, the lifetime of a star of that mass and metallicity in Gyr according to the Vincenzo et al. (2016) relation. If `which == "age"`, the mass of a star in $M_\odot$ with the specified lifetime in Gyr.

Notes

This relation is of the following functional form:

$$\tau = A \exp(B m^{-C})$$

where A, B, and C are functions of metallicity. Vincenzo et al. (2016) computed the values of these coefficients using the PARSEC stellar evolution code (Bressan et al. 2012[2]; Tang et al. 2014[3]; Chen et al. 2015[4]) which were then used in combination with a one-zone chemical evolution model to reproduce the color-magnitude diagram of the Sculptor dwarf galaxy.

VICE stores a table of values of A, B, and C as internal data, interpolating linearly between them in metallicity Z. This form of the mass-lifetime relation quantifies the *total* lifetimes of stars (i.e. the post main sequence lifetimes are included a priori, and calculating main sequence lifetimes only is not available).

Example Code

```
>>> import vice
>>> vice.mlr.vincenzo2016(1) # the lifetime of the sun
11.146605845086224
>>> vice.mlr.vincenzo2016(1, Z = 0.007) # at half solar metallicity
8.873217338534232
>>> vice.mlr.vincenzo2016(1, which = "age") # what mass lives 1 Gyr?
2.2090823821884733
>>> vice.mlr.vincenzo2016(2, which = "age") # 2 Gyr?
1.7081324509721378
>>> vice.mlr.vincenzo2016(2, Z = 0.007, which = "age")
1.6033524375573776
>>> vice.mlr.vincenzo2016(3)
0.48203178060452745
>>> vice.mlr.vincenzo2016(3, Z = 0.001)
```

(continues on next page)

[2] Bressan et al. (2012), MNRAS, 427, 127
[3] Tang et al. (2014), MNRAS, 445, 4287
[4] Chen et al. (2015), MNARS, 452, 1068

(continued from previous page)

```
0.33000930985434906
>>> vice.mlr.vincenzo2016(3, which = "age")
1.4878361243926437
```

vice.mlr.hpt2000

Compute either the lifetime or the mass of a dying star according to the mass-lifetime relation of Hurley, Pols & Tout (2000)[1].

Signature: vice.mlr.hpt2000(qty, postMS = 0.1, Z = 0.014, which = "mass")

New in version 1.3.0.

Parameters

qty [float] Either the mass of a star in $M_\odot$ or the age of a stellar population in Gyr. Interpretion set by the keyword argument `which`.

postMS [float [default][0.1]] The ratio of a star's post main sequence lifetime to its main sequence lifetime. Zero to compute the main sequence lifetime alone, or the main sequence turnoff mass when `which == "age"`.

Z [float [default][0.014]] The metallicity by mass of the stellar population.

which [str [case-insensitive] [default]["mass"]] The interpretation of `qty`: either `"mass"` or `"age"` (case-insensitive). If `which == "mass"`, then `qty` represents a stellar mass in $M_\odot$ and this function will compute a lifetime in Gyr. Otherwise, `qty` represents the age of a stellar population and the mass of a star with the specified lifetime will be calculated.

Returns

x [float] If `which == "mass"`, the lifetime of a star of that mass and metallicity in Gyr according to the Hurley, Pols & Tout (2000) relation. If `which == "age"`, the mass of a star in $M_\odot$ with the specified lifetime in Gyr.

Notes

The Hurley, Pols & Tout (2000) relation quantifies the main sequence lifetime according to (see their section 5.1):

$$t_{\text{MS}} = \max(\mu, x) t_{\text{BGB}}$$

where t_{BGB} is the time required for the star to reach the base of the giant branch (BGB), given by:

$$t_{\text{BGB}} = \frac{a_1 + a_2 M^4 + a_3 M^{5.5} + M^7}{a_4 M^2 + a_5 M^7}$$

where M is the mass of the star in solar masses and the coefficients a_n depend on metallicity in a manner described in their Appendix A. The quantities μ and x are given by

$$\mu = \max\left(0.5, 1.0 - 0.01\max\left(\frac{a_6}{M^{a_7}}, a_8 + \frac{a_9}{M^{a_{10}}}\right)\right)$$

[1] Hurley, Pols & Tout (2000), MNRAS, 315, 543

and

$$x = \max(0.95, \min(0.95 - 0.03(\zeta + 0.30103), 0.99))$$

where ζ is calculated from the metallicity by mass Z according to $\zeta = \log_{10}(Z/0.02)$.

VICE stores the necessary data with which to compute the coefficients a_n as internal data. Although this form takes into account the metallicity dependence of stellar lifetimes, this formalism quantifies *only* the main sequence lifetimes, with the parameter `postMS` quantifying the length of the post main sequence lifetime.

In calculating stellar masses from ages (i.e. when `which == "age"`), the equation must be solved numerically. For this, VICE makes use of the bisection root-finding algorithm described in chapter 9 of Press, Teukolsky, Vetterling & Flannery (2007)[2].

Example Code

```
>>> import vice
>>> vice.mlr.hpt2000(1) # the lifetime of the sun
10.949827652466094
>>> vice.mlr.hpt2000(1, postMS = 0) # main sequence lifetime only
9.954388774969177
>>> vice.mlr.hpt2000(1, Z = 0.007) # at half solar metallicity
9.160722377683282
>>> vice.mlr.hpt2000(1, postMS = 0, Z = 0.007)
8.32792943425753
>>> vice.mlr.hpt2000(1, which = "age") # what mass lives 1 Gyr?
2.1353209857940674
>>> vice.mlr.hpt2000(2, which = "age") # 2 Gyr
1.6775577716827392
>>> vice.mlr.hpt2000(2, postMS = 0, which = "age") # MS turnoff mass
1.6241520633697508
>>> vice.mlr.hpt2000(3)
0.39829399679015326
>>> vice.mlr.hpt2000(3, which = "age")
1.4629812650680543
>>> vice.mlr.hpt2000(3, postMS = 0, which = "age")
1.4181586170196532
```

vice.mlr.ka1997

Compute either the lifetime or the mass of a dying star according to the mass-lifetime relation of Kodama & Arimoto (1997)[1].

Signature: vice.mlr.ka1997(qty, Z = 0.014, which = “mass”)

New in version 1.3.0.

[2] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press
[1] Kodama & Arimoto (1997), A&A, 320, 41

Parameters

qty [float] Either the mass of a star in $M_\odot$ or the age of a stellar population in Gyr. Interpretion set by the keyword argument `which`.

Z [float [default][0.014]] The metallicity by mass of the stellar population.

which [str [case-insensitive] [default][“mass”]] The interpretation of `qty`: either `"mass"` or `"age"` (case-insensitive). If `which == "mass"`, then `qty` represents a stellar mass in $M_\odot$ and this function will compute a lifetime in Gyr. Otherwise, `qty` represents the age of a stellar population and the mass of a star with the specified lifetime will be calculated.

Returns

x [float] If `which == "mass"`, the lifetime of a star of that mass and metallicity in Gyr according to Kodama & Arimoto (1997). If `which == "age"`, the mass of a star in $M_\odot$ with the specified lifetime in Gyr.

Notes

Kodama & Arimoto (1997) quantified their mass-lifetime relation using stellar evolution tracks computed with the code presented in Iwamoto & Saio (1999)[2]. They report lifetimes on a table of stellar mass and metallicity, which VICE stores as internal data. To compute lifetimes at any mass and metallicity, it runs a 2-dimensional linear interpolation function between the appropriate elements of the mass-metallicity grid, linearly extrapolating to higher or lower masses or metallicities as needed.

This form of the mass-lifetime relation quantifies the *total* lifetimes of stars (i.e. the post main sequence lifetimes are included a priori, and calculating main sequence lifetimes only is not available).

Because an interpolation scheme is used to compute lifetimes, inverting the relationship to compute masses from ages must be done numerically. For this, VICE makes use of the bisection root-finding algorithm described in chapter 9 of Press, Teukolsky, Vetterling & Flannery (2007)[3].

Example Code

```
>>> import vice
>>> vice.mlr.ka1997(1) # the lifetime of the sun
11.225750000000001
>>> vice.mlr.ka1997(1, Z = 0.007) # at half solar metallicity
9.674165
>>> vice.mlr.ka1997(1, which = "age") # what mass lives 1 Gyr?
2.2440397491455073
>>> vice.mlr.ka1997(2, which = "age") # 2 Gyr?
1.6146153297424315
>>> vice.mlr.ka1997(2, Z = 0.007, which = "age")
1.5497655410766602
>>> vice.mlr.ka1997(3)
0.44375200000000004
>>> vice.mlr.ka1997(3, Z = 0.007)
0.40909550000000006
```

(continues on next page)

[2] Iwamoto & Saio (1999), ApJ, 521, 297
[3] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press

(continued from previous page)

```
>>> vice.mlr.ka1997(3, Z = 0.007, which = "age")
1.3685676021575928
```

vice.mlr.pm1993

Compute either the lifetime or the mass of a dying star according to the mass-lifetime relation of Padovani & Matteucci (1993)[1].

Signature: vice.mlr.pm1993(qty, postMS = 0.1, which = "mass")

New in version 1.3.0.

Note: This parameterization of the mass-lifetime relation predicts the lives of solar mass stars to be shorter than most other forms (~7.8 Gyr compared to ~10 Gyr).

Parameters

qty [float] Either the mass of a star in $M_\odot$ or the age of a stellar population in Gyr. Interpretion set by the keyword argument `which`.

postMS [float [default][0.1]] The ratio of a star's post main sequence lifetime to its main sequence lifetime. Zero to compute the main sequence lifetime alone, or the main sequence turnoff mass when `which == "age"`.

which [str [case-insensitive] [default]["mass"]] The interpretation of `qty`: either `"mass"` or `"age"` (case-insensitive). If `which == "mass"`, then `qty` represents a stellar mass in $M_\odot$ and this function will compute a lifetime in Gyr. Otherwise, `qty` represents the age of a stellar population and the mass of a star with the specified lifetime will be calculated.

Returns

x [float] If `which == "mass"`, the lifetime of a star of that mass and metallicity in Gyr according to Padovani & Matteucci (1993). If `which == "age"`, the mass of a star in $M_\odot$ with the specified lifetime in Gyr.

Notes

Padovani & Matteucci (1993) parameterize the mass-lifetime relation according to:

$$\log_{10}\tau = \frac{\alpha - \sqrt{\beta - \gamma(\eta - \log_{10}(M/M_\odot))}}{\mu}$$

for stellar masses below 6.6 $M_\odot$ with τ in Gyr, and

$$\tau = 1.2(M/M_\odot)^{-1.85} + 0.003 \text{ Gyr}$$

for masses above 6.6 $M_\odot$. Below 0.6 $M_\odot$, the lifetime flattens off at 160 Gyr. The coefficients α, β, γ, η, and μ are given below:

[1] Padovani & Matteucci (1993), ApJ, 416, 26

α	0.334
β	1.790
γ	0.2232
η	7.764
μ	0.1116

Though this form was originally published in Padovani & Matteucci (1993), in detail the form here is taken from Romano et al. (2005)[2].

The timescale τ quantifies only the main sequence lifetime of stars; the parameter `postMS` specifies the length of the post main sequence lifetime. This parameterization neglects the metallicity dependence of the mass-lifetime relation.

Example Code

```
>>> import vice
>>> vice.mlr.pm1993(1) # the lifetime of the sun
7.825388414293052
>>> vice.mlr.pm1993(1, postMS = 0) # main sequence lifetime only
7.113989467539137
>>> vice.mlr.pm1993(1, which = "age") # what mass lives 1 Gyr?
1.8113833345909132
>>> vice.mlr.pm1993(2, which = "age") # 2 Gyr?
1.4492132883063318
>>> vice.mlr.pm1993(2, postMS = 0, which = "age") # MS turnoff mass
1.4079983006192527
>>> vice.mlr.pm1993(3)
0.2571838372046172
>>> vice.mlr.pm1993(3, postMS = 0)
0.23380348836783377
>>> vice.mlr.pm1993(3, which = "age")
1.285731894196999
```

vice.mlr.mm1989

Compute either the lifetime or the mass of a dying star according to the mass-lifetime relation of Maeder & Meynet (1989)[1].

Signature: vice.mlr.mm1989(qty, postMS = 0.1, which = “mass”)

New in version 1.3.0.

[2] Romano et al. (2005), A&A, 430, 491
[1] Maeder & Meynet (1989), A&A, 210, 155

Parameters

qty [float] Either the mass of a star in $M_\odot$ or the age of a stellar population in Gyr. Interpretion set by the keyword argument `which`.

postMS [float [default][0.1]] The ratio of a star's post main sequence lifetime to its main sequence lifetime. Zero to compute the main sequence lifetime alone, or the main sequence turnoff mass when `which == "age"`.

which [str [case-insensitive] [default][“mass”]] The interpretation of `qty`: either `"mass"` or `"age"` (case-insensitive). If `which == "mass"`, then `qty` represents a stellar mass in $M_\odot$ and this function will compute a lifetime in Gyr. Otherwise, `qty` represents the age of a stellar population and the mass of a star with the specified lifetime will be calculated.

Returns

x [float] If `which == "mass"`, the lifetime of a star of that mass and metallicity in Gyr according to Maeder & Meynet (1989). If `which == "age"`, the mass of a star in $M_\odot$ with the specified lifetime in Gyr.

Notes

The mass-lifetime relation of Maeder & Meynet (1989) is given by:

$$\log_{10}\tau = \alpha\log_{10}(M/M_\odot) + \beta$$

for stellar masses below 60 $M_\odot$. Above this mass, the lifetime is given by:

$$\tau = 1.2\left(\frac{M}{M_\odot}\right)^{-1.85} + 0.003$$

and in both cases, τ is in Gyr.

Though this form was originally published in Maeder & Meynet (1989), in detail the form here is taken from Romano et al. (2005)[2].

The timescale τ quantifies only the main sequence lifetime of stars; the parameter `postMS` specifies the length of the post main sequence lifetime. This parameterization neglects the metallicity dependence of the mass-lifetime relation.

The values of the coefficients α and β vary with stellar mass according to ($m = M/M_\odot$):

Mass Range	α	β
$m \leq 1.3$	-0.6545	1
$1.3 < m \leq 3$	-3.7	1.35
$3 < m \leq 7$	-2.51	0.77
$7 < m \leq 15$	-1.78	0.17
$15 < m \leq 60$	-0.86	-0.94

In calculating stellar masses from ages (i.e. when `which == "age"`), the equation must be solved numerically. For this, VICE makes use of the bisection root-finding algorithm described in chapter 9 of Press, Teukolsky, Vetterling & Flannery (2007)[3].

[2] Romano et al. (2005), A&A, 430, 491
[3] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press

Example Code

```
>>> import vice
>>> vice.mlr.mm1989(1) # the lifetime of the sun
11.0
>>> vice.mlr.mm1989(1, postMS = 0) # main sequence lifetime only
10.0
>>> vice.mlr.mm1989(1, which = "age") # what mass lives 1 Gyr?
2.3775540199279783
>>> vice.mlr.mm1989(2, which = "age") # 2 Gyr?
1.9712891674041746
>>> vice.mlr.mm1989(2, postMS = 0, which = "age") # MS turnoff mass
1.9207444791793824
>>> vice.mlr.mm1989(3)
0.42271148013148074
>>> vice.mlr.mm1989(3, postMS = 0)
0.38428316375589155
>>> vice.mlr.mm1989(3, which = "age")
1.721426746368408
```

vice.mlr.larson1974

Compute either the lifetime or the mass of a dying star according to the mass-lifetime relation of Larson (1974)[1].

Signature: vice.mlr.larson1974(qty, postMS = 0.1, which = "mass")

New in version 1.3.0.

Parameters

qty [float] Either the mass of a star in $M_\odot$ or the age of a stellar population in Gyr. Interpretion set by the keyword argument `which`.

postMS [float [default][0.1]] The ratio of a star's post main sequence lifetime to its main sequence lifetime. Zero to compute the main sequence lifetime alone, or the main sequence turnoff mass when `which == "age"`.

which [str [case-insensitive] [default]["mass"]] The interpretation of `qty`: either `"mass"` or `"age"` (case-insensitive). If `which == "mass"`, then `qty` represents a stellar mass in $M_\odot$ and this function will compute a lifetime in Gyr. Otherwise, `qty` represents the age of a stellar population and the mass of a star with the specified lifetime will be calculated.

Returns

x [float] If `which == "mass"`, the lifetime of a star of that mass and metallicity in Gyr according to Larson (1974). If `which == "age"`, the mass of a star in $M_\odot$ with the specified lifetime in Gyr.

[1] Larson (1974), MNRAS, 166, 585

Notes

Larson (1974) present the following fit to the compilation of evolutionary lifetimes presented in Tinsley (1972)[2]:

$$\log_{10}\tau = \alpha + \beta\log_{10}(M/M_\odot) + \gamma(\log_{10}(M/M_\odot))^2$$

where α = 1 for τ measured in Gyr, β = -3.42, and γ = 0.88. Though this form was originally presented in Larson (1974), the values of the coefficients were taken from David, Forman & Jones (1990)[3] and Kobayashi (2004)[4].

The timescale τ quantifies only the main sequence lifetime of stars; the parameter `postMS` specifies the length of the post main sequence lifetime. This parameterization neglects the metallicity dependence of the mass-lifetime relation.

In solving the inverse function (i.e. mass as a function of lifetime), the solution proceeds analytically according to the quadratic formula where subtraction is chosen in the numerator over addition as this is the physical solution.

Example Code

```
>>> import vice
>>> vice.mlr.larson1974(1) # the lifetime of the sun
11.0
>>> vice.mlr.larson1974(1, postMS = 0) # main sequence lifetime only
10.0
>>> vice.mlr.larson1974(1, which = "age") # what mass lives 1 Gyr?
2.1529829084164525
>>> vice.mlr.larson1974(2, which = "age") # 2 Gyr?
1.698651828235309
>>> vice.mlr.larson1974(2, postMS = 0, which = "age") # MS lifetime only
1.6460010348842196
>>> vice.mlr.larson1974(3)
0.40734775084938435
>>> vice.mlr.larson1974(3, postMS = 0)
0.3703161371358039
>>> vice.mlr.larson1974(3, which = "age")
1.4882047037330677
```

vice.yields

Nucleosynthetic Yield Tools

Each sub-package stores built-in yield tables and user-presets for each element from each enrichment channel.

Signature: vice.yields

[2] Tinsley (1972), A&A, 20, 383
[3] David, Forman & Jones (1990), ApJ, 359, 29
[4] Kobayashi (2004), MNRAS, 347, 740

Contains

agb [<module>] Yields from asymptotic giant branch stars

ccsne [<module>] Yields from core collapse supernovae

sneia [<module>] Yields from type Ia supernovae

presets [<module>] Yield settings presets

test [<function>] Run the tests on this package

Notes

All yields built into VICE reflect stable isotopes *only*. The built-in tables reflect data which assume all important radioactive nuclides have fully decayed. Further details can be found in the `agb`, `ccsne`, and `sneia` modules and in the journal publications in which the data were originally published.

vice.yields.agb

Asymptotic Giant Branch (AGB) Star Nucleosynthetic Yield Tools

Analyze built-in yield tables and modify yield settings for use in simulations. This package provides tables from the following nucleosynthetic yield studies:

- Cristallo et al. (2011)[1]
- Karakas (2010)[2]
- Ventura et al. (2013, 2014, 2018, 2020)[3][4][5][6]
- Karakas & Lugaro (2016)[7], Karakas et al. (2018)[8]

Contents

grid [<function>] Return the stellar mass-metallicity grid of fractional nucleosynthetic yields for given element and study

interpolator [`object`] Linearly interpolates on the stellar mass-metallicity grid of yields for use in the global yield settings.

settings [`dataframe`] Stores current settings for these yields

cristallo11 [yield preset] Sets the yields according to the Cristallo et al. (2011, 2015) studies.

karakas10 [yield preset] Sets the yields according to the Karakas (2010) study.

ventura13 [yield preset] Sets the yields according to the Ventura et al. (2013, 2014, 2018, 2020) studies.

karakas16 [yield preset] Sets the yields according to the Karakas & Lugaro (2016), Karakas et al. (2018) studies.

New in version 1.3.0: The "ventura13" and "karakas16" yield models were introduced in version 1.3.0.

[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Karakas (2010), MNRAS, 403, 1413
[3] Ventura et al. (2013), MNRAS, 431, 3642
[4] Ventura et al. (2014), MNRAS, 437, 3274
[5] Ventura et al. (2018), MNRAS, 475, 2282
[6] Ventura et al. (2020), A&A, 641, A103
[7] Karakas & Lugaro (2016), ApJ, 825, 26
[8] Karakas et al. (2018), MNRAS, 477, 421

Notes

The data stored in this module are reported for each corresponding study *as published*. With the exception of converting the values to *fractional* yields (i.e. by dividing by progenitor initial mass), they were not modified in any way.

vice.yields.agb.grid

Obtain the stellar mass-metallicity grid of fractional net yields from asymptotic giant branch stars published in a nucleosynthesis study.

Signature: vice.yields.agb.grid(element, study = "cristallo11")

Parameters

element [`str` [case-insensitive]] The symbol of the element to obtain the yield grid for.

study [`str` [case-insensitive] [default]["cristallo11"]] A keyword denoting which study to pull the yield table from.

Recognized Keywords:

- "cristallo11" : Cristallo et al. (2011, 2015)[1][2]
- "karakas10" : Karakas (2010)[3]
- "ventura13" : Ventura et al. (2013)[4]
- **"karakas16": Karakas & Lugaro (2016)[5]; Karkas et al. (2018)** [6]

New in version 1.3.0: The "ventura13" and "karakas16" yield models were introduced in version 1.3.0.

Returns

grid [`tuple` (2-D)] A tuple of tuples containing the yield grid. The first axis is the stellar mass, and second is the metallicity

masses [`tuple`] The masses in units of $M_\odot$ that the yield grid is sampled on.

z [`tuple`] The metallicities by mass Z that the yield grid is sample on.

Raises

- **ValueError**
  - The study or the element are not built into VICE
- **LookupError**
  - `study == "karakas10"` and the atomic number of the element is ≥ 29. The Karakas (2010) study did not report yields for elements heavier the nickel.

[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Cristallo et al. (2015), ApJS, 219, 40
[3] Karakas (2010), MNRAS, 403, 1413
[4] Ventura et al. (2013), MNRAS, 431, 3642
[5] Kakaras & Lugaro (2016), ApJ, 825, 26
[6] Karakas et al. (2018), MNRAS, 477, 421

  - The Ventura et al. (2013) tables include yields only for the following elements: he, c, n, o, ne, na, mg, al, si. A request for a table for any other element with raise an exception.

- **IOError [Occur's only if VICE's file structure has been modified]**
  - The parameters passed to this function are allowed but the data file is not found.

Notes

The AGB star yields stored by VICE are reported *as published* by each corresponding study. With the exception of converting the values to *fractional* yields (i.e. by dividing by progenitor initial mass), they were not modified in any way.

Example Code

```
>>> y, m, z = vice.agb_yield_grid("sr")
>>> m
    (1.3, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 6.0)
>>> z
    (0.0001, 0.0003, 0.001, 0.002, 0.003, 0.006, 0.008, 0.01, 0.014, 0.02)
>>> # the fractional yield from 1.3 Msun stars at Z = 0.001
>>> y[0][2]
    2.32254e-09
```

vice.yields.agb.interpolator

A bi-linear interpolation scheme constructed from the yield tables built into the vice.yields.agb module.

Signature: vice.yields.agb.interpolator(element, study = "cristallo11")

New in version 1.2.0.

Parameters

element [`str` [case-insensitive]] The symbol of the element to obtain the yield grid for.

study [`str` [case-insensitive] [default]["cristallo11"]] A keyword denoting which study to pull the yield table from.

Recognized keywords:

- "cristallo11" : Cristallo et al. (2011, 2015)[1][2]
- "karakas10" : Karakas (2010)[3]
- "ventura13" : Ventura et al. (2013)[4]
- **"karakas16": Karakas & Lugaro (2016)[5]; Karkas et al. (2018)** [6]

New in version 1.3.0: The "ventura13" and "karakas16" yield models were introduced in version 1.3.0.

[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Cristallo et al. (2015), ApJS, 219, 40
[3] Karakas (2010), MNRAS, 403, 1413
[4] Ventura et al. (2013), MNRAS, 431, 3642
[5] Karakas & Lugaro (2016), ApJ, 825, 26
[6] Karakas et al. (2018), MNRAS, 477, 421

Attributes

masses [`list` [elements of type `float`]] The masses on which the yield table is sampled.

metallicities [`list` [elements of type `float`]] The metallicities on which the yield table is sampled.

yields [`list` [elements of type `list`]] The yields at each stellar mass and metallicity reported by the adopted study.

Calling

Call this object with stellar mass and metallicity to compute the fractional net yield for such an AGB star, estimated via bi-linear interpolation.

Parameters:

- **mass** [real number] The stellar mass of an AGB star in solar masses.
- **metallicity** [real number] The metallicity by mass Z of the AGB star.

Returns:

- **y** [real number] The fractional net yield, estimated via bi-linear interpolation. See *Notes* below.

Tip: This object can be used as a callable object to describe the AGB star yields of any given element. For the base class, it makes little sense to do so, since this is the same as setting the yield to the study itself. However, alternate functionality or interpolation schema can be achieved by subclassing this object and overriding the `__call__` function.

Notes

To conduct the interpolation, this object first finds the masses and metallicities on the grid that enclose the values passed as parameters when this object is called. It then interpolates linearly in metallicity at constant mass twice, then once in mass at constant metallicity to determine the value of the yield at the appropriate mass and metallicity. If the mass and/or metallicity are above or below the maximum or minimum values of the grid, the interpolation scheme falls back on linear extrapolation off of the two largest/smallest values.

This object inherits its functionality from `vice.toolkit.interpolation.interp_scheme_2d`, where the stellar masses are the x-coordinates, the metallicities the y-coordiantes, and the yields the z-coordinates. For further details, see the associated documentation.

Warning: VICE's AGB star yield interpolation routines force negative yields to zero below 1.5 $M_\odot$ in order to prevent numerical artifacts associated with extrapolation to stellar masses below the table of yields. This object does **not** include this functionality; numerical artifacts may be introduced as a consequence (see "Asymptotic Giant Branch Stars" under "Nucleosynthetic Yields" in VICE's science documentation: https://vice-astro.readthedocs.io/en/latest/science_documentation/index.html). If this correction is desired, users should subclass this object and add the following `if` statement to the `__call__` function for a value `y` returned from the inherited `__call__` function: `if mass < 1.5 and y < 0: return 0`.

Example Code

```
>>> import vice
>>> example = vice.yields.agb.interpolator('c')
>>> example.masses
[1.3, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 6.0]
>>> example.metallicities
[0.0001, 0.0003, 0.001, 0.002, 0.003, 0.006, 0.008, 0.01, 0.014, 0.02]
>>> example.yields[0]
[0.00233122,
 0.00206212,
 0.00163226,
 0.00150313,
 0.000781408,
 0.000406231,
 -5.03077e-05,
 -0.000150308,
 -0.000317615,
 -0.000422]
>>> example.yields[2]
[0.00760034,
 0.00650061,
 0.0060516,
 0.00610347,
 0.00510498,
 0.00443045,
 0.00347925,
 0.0035931,
 0.0026206,
 0.002503]
>>> # the yield at 2.2 solar masses and Z = 0.011
>>> example(2.2, 0.011)
0.004022337000000001
>>> example(5.4, 0.0036)
0.0004046900263999999
>>> example(1.8, 0.018)
0.0017274533333333335
```

vice.yields.agb.interpolator.masses

Type : `list` [elements of type `float`]

The stellar masses in $M_\odot$ on which the yield grid is sampled.

Example Code

```
>>> import vice
>>> example = vice.yields.agb.interpolator('c')
>>> example.masses
[1.3, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 6.0]
>>> example = vice.yields.agb.interpolator('c', study = "karakas10")
[1.0,
 1.25,
 1.5,
 1.75,
 1.9,
 2.25,
 2.5,
 3.0,
 3.5,
 4.0,
 4.5,
 5.0,
 5.5,
 6.0]
```

vice.yields.agb.interpolator.metallicities

Type : `list` [elements of type `float`]

The metallicities by mass $Z = M_z/M_\star$ of the AGB stars on which the yield grid is sampled.

Example Code

```
>>> import vice
>>> example = vice.yields.agb.interpolator('c')
>>> example.metallicities
[0.0001, 0.0003, 0.001, 0.002, 0.003, 0.006, 0.008, 0.01, 0.014, 0.02]
>>> example = vice.yields.agb.interpolator('c', study = "karakas10")
>>> example.metallicities
[0.0001, 0.004, 0.008, 0.02]
```

vice.yields.agb.interpolator.yields

Type : `list` [elements of type `list`, storing `float` types]

The fractional net yields of the given element on which the yield grid is sampled.

Example Code

```
>>> import vice
>>> example = vice.yields.agb.interpolator('c')
>>> example.yields[0]
[0.00233122,
 0.00206212,
 0.00163226,
 0.00150313,
 0.000781408,
 0.000406231,
 -5.03077e-05,
 -0.000150308,
 -0.000317615,
 -0.000422]
>>> example.yields[2]
[0.00760034,
 0.00650061,
 0.0060516,
 0.00610347,
 0.00510498,
 0.00443045,
 0.00347925,
 0.0035931,
 0.0026206,
 0.002503]
>>> example = vice.yields.agb.interpolator('c', study = "karakas10")
>>> example.yields[0]
[0.000250197, -3.861e-05, -6.944e-05, -0.0001502]
>>> example.yields[2]
[0.00773393, 0.00111333, -5.62667e-05, -0.0004974]
```

vice.yields.agb.settings

The VICE dataframe: global yield settings for AGB stars

For each chemical element, this object stores the current asymptotic giant branch (AGB) star nucleosynthetic yield setting. See *Notes* below for mathematical details.

New in version 1.2.0: In earlier versions, functions and classes within VICE accepted keyword arguments or attributes which encoded which model table of yields to adopt. This same functionality can be achieved by assigning a string as the yield setting for specific elements.

Note: Modifying yield settings through this dataframe is equivalent to going through the `vice.elements` module.

Indexing

- **str [case-insensitive]** [elemental symbols] This dataframe must be indexed by the symbol of an element recognized by VICE as it appears on the periodic table.

Item Assignment

For each chemical element, the AGB star yield can be assigned either:

- **str [case-insensitive]** [Adopt values published by a given study] Keywords correspond to yields calculated on a table of progenitor masses and metallicities which can be adopted directly.
  - "cristallo11" : Cristallo et al. (2011, 2015)[1][2]
  - "karakas10" : Karakas (2010)[3]
  - "ventura13" : Ventura et al. (2013)[4]
  - **"karakas16"** [Karakas & Lugaro (2016)[5] ; Karakas et al. (2018)] [6]

  New in version 1.3.0: The "ventura13" and "karakas16" yields models were introduced in version 1.3.0.
- **<function>** [Mathematical function describing the yield] Must accept progenitor zero age main sequence mass in $M_\odot$ as the first parameter and the metallicity by mass Z as the second.

Functions

- keys
- todict
- restore_defaults
- factory_settings
- save_defaults

Notes

VICE defines the yield from AGB stars as the fraction of a star's initial mass which is processed into some element. As with all other yields in VICE, these are *net* rather than *gross* yields in that they quantify only the mass of a given element which is newly produced. For a star of mass $M_\star$, the mass of the element ejected to the ISM, not counting previously produced nucleosynthetic material, is given by:

$$M = y_{\rm AGB}(M_\star, Z_\star)M_\star$$

where $y_{\rm AGB}$ is the yield and $Z_\star$ is the initial metallicity of the star.

This definition is retained in one- and multi-zone chemical evolution models as well. For further details, see VICE's science documentation: https://vice-astro.readthedocs.io/en/latest/science_documentation/index.html.

[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Cristallo et al. (2015), ApJS, 219, 40
[3] Karakas (2010), MNRAS, 403, 1413
[4] Ventura et al. (2013), MNRAS, 431, 3642
[5] Kakaras & Lugaro (2016), ApJ, 825, 26
[6] Karakas et al. (2018), MNRAS, 477, 421

Example Code

```
>>> import vice
>>> vice.yields.agb.settings["n"] = "cristallo11"
>>> vice.yields.agb.settings["N"]
        "cristallo11"
>>> vice.yields.agb.settings["N"] = "karakas10"
>>> vice.yields.agb.settings["n"]
        "karakas10"
>>> def f(m, z):
        return 0.001 * m * (z / 0.014)
>>> vice.yields.agb.settings["n"] = f
>>> vice.yields.agb.settings["N"]
        <function __main__.f(z)>
```

vice.yields.agb.settings.keys

Returns the keys of the AGB star yield settings dataframe.

Signature: vice.yields.agb.settings.keys()

Note: By nature, this function will simply return a list of all elements that are built into VICE - the same thing as `vice.elements.recognized`.

Example Code

```
>>> import vice
>>> elements = vice.yields.agb.settings.keys()
>>> tuple(elements) == vice.elements.recognized
        True
```

vice.yields.agb.settings.todict

Returns the AGB star yield settings dataframe as a dictionary.

Signature: vice.yields.agb.settings.todict()

Note: Modifications to the dictionary returned by this function will *not* affect the global yield settings.

Note: Python dictionaries are case-sensitive, and are thus less flexible than this class.

Example Code

```
>>> import vice
>>> example = vice.yields.agb.settings.todict()
>>> example["c"]
        "cristallo11"
>>> example["C"]
        Traceback (most recent call last):
          File "<stdin>", line 1, in <module>
        KeyError: 'C'
>>> example["c"] = "not a yield setting"
>>> example["c"]
        "not a yield setting"
>>> vice.yields.agb.settings["c"]
        "cristallo11"
```

vice.yields.agb.settings.restore_defaults

Restores the AGB star yield settings to their default values (i.e. undoes any changes since VICE was imported).

Signature: vice.yields.agb.settings.restore_defaults()

Example Code

```
>>> import vice
>>> vice.yields.agb.settings["c"] = "karakas10"
>>> vice.yields.agb.settings["n"] = "karakas10"
>>> vice.yields.agb.settings["o"] = "karakas10"
>>> vice.yields.agb.settings.restore_defaults()
>>> vice.yields.agb.settings["c"]
        "cristallo11"
>>> vice.yields.agb.settings["n"]
        "cristallo11"
>>> vice.yields.agb.settings["o"]
        "cristallo11"
```

vice.yields.agb.settings.factory_settings

Restores the AGB star yield settings to their factory defaults. This differs from `vice.yields.agb.settings.restore_defaults` in that users may modify their default values from those that VICE is distributed with.

Signature: vice.yields.agb.settings.factory_settings()

Tip: To revert your nucleosynthetic yield settings back to their production defaults *permanently*, simply call `vice.yields.agb.settings.save_defaults` immediately following this function.

Example Code

```
>>> import vice
>>> vice.yields.agb.settings["c"]
        "karakas10" # the user has modified their default yield for carbon
>>> vice.yields.agb.settings.factory_settings()
>>> vice.yields.agb.settings["c"]
        "cristallo11"
```

vice.yields.agb.settings.save_defaults

Saves the current AGB star yield settings as the default values.

Signature: vice.yields.agb.settings.save_defaults()

Note: Saving functional yields requires the package dill, an extension to `pickle` in the python standard library. It is recommended that VICE users install dill >= 0.2.0.

Example Code

```
>>> import vice
>>> vice.yields.agb.settings["c"]
        "cristallo11"
>>> vice.yields.agb.settings["c"] = "karakas10"
>>> vice.yields.agb.settings.save_defaults()
```

After re-launching the python interpreter:

```
>>> import vice
>>> vice.yields.agb.settings["c"]
        "karakas10"
```

vice.yields.agb.cristallo11

Cristallo et al. (2011, 2015) Asymptotic Giant Branch (AGB) star yields.

Signature: from vice.yields.agb import cristallo11

Importing this module will set the AGB star yield settings for all elements to "cristallo11", which combines the yields for 1 - 3 $M_\odot$ AGB star progenitors from Cristallo et al. (2011)[1] with the yields for 4 - 6 $M_\odot$ progenitors from Cristallo et al. (2015)[2]. AGB star yields will then be calculated using bi-linear interpolation in progenitor mass and metallicity using these data to determine AGB star yields in chemical evolution models.

Note: This module is not imported with a simple `import vice` statement.

---

[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Cristallo et al. (2015), ApJS, 219, 40

vice.yields.agb.karakas10

Karakas (2010), MNRAS, 403, 1413 Asymptotic Giant Branch (AGB) star yields.

Signature: from vice.yields.agb import karakas10

Importing this module will set the AGB star yield setting for all elements up to nickel to "karakas10". AGB star yields will then be calculated using bi-linear interpolation in progenitor mass and metallicity using these data to determine AGB star yields in chemical evolution models.

Note: This module is not imported with a simple `import vice` statement.

Raises

- **ScienceWarning** The Karakas (2010) study did not report yields for elements heavier than nickel. The settings for these elements will not be modified.

vice.yields.agb.ventura13

Ventura et al. (2013, 2014, 2018, 2020) Asymptotic Giant Branch (AGB) Star yields

Signature: from vice.yields.agb import ventura13

New in version 1.3.0.

Importing this module will set the AGB star yield settings for all elements where yield tables are available to "ventura13". AGB star yields will then be calculated using bi-linear interpolation in progenitor mass and metallicity using these data to determine AGB star yields in chemical evolution models. This module combines yields from various Ventura et al. publications into one yield table. At the following metallicities, the yields are adopted from the associated journal publication:

- Z = 0.0003 : Ventura et al. (2013)[1]
- Z = 0.001 : Unpublished
- Z = 0.002 : Unpublished
- Z = 0.004 : Ventura et al. (2014)[2]
- Z = 0.008 : Ventura et al. (2013)
- Z = 0.014 : Ventura et al. (2018)[3]
- Z = 0.04 : Ventura et al. (2020)[4]

Note: This module is not imported with a simple `import vice` statement.

[1] Ventura et al. (2013), MNRAS, 431, 3642
[2] Ventura et al. (2014), MNRAS, 437, 3274
[3] Ventura et al. (2018), MNRAS, 475, 2282
[4] Ventura et al. (2020), A&A, 641, A103

Raises

- **ScienceWarning** This module only provides tables for the following elements: he, c, n, o, ne, na, mg, al, si. The settings for other elements will not be modified.

vice.yields.agb.karakas16

Karakas & Lugaro (2016), Karakas et al. (2018) Asymptotic Giant Branch (AGB) star yields

Signature: from vice.yields.agb import karakas16

New in version 1.3.0.

Importing this module will set the AGB star yield setting for all elements to "karakas16". This module combines the yields for Z = 0.007, 0.014, and 0.03 progenitors from Karakas & Lugaro (2016)[1] with the yields for Z = 0.0028 progenitors from Karakas et al. (2018)[2] (a metallicity characteristic of stars in the Small Magellanic Cloud). AGB star yields will then be calculated using bi-linear interpolation in progenitor mass and metallicity using these data to determine AGB star yields in chemical evolution models.

Note: This module is not imported with a simple `import vice` statement.

---

vice.yields.ccsne

Core Collapse Supernovae (CCSNe) Nucleosynthetic Yield Tools

Calculate IMF-averaged yields and modify yield settings for use in simulations. This package provides tables from the following nucleosynthetic yield studies:

- Limongi & Chieffi (2018)[1]
- Sukhbold et al. (2016)[2]
- Chieffi & Limongi (2013)[3]
- Nomoto, Kobayashi & Tominaga (2013)[4]
- Chieffi & Limongi (2004)[5]
- Woosley & Weaver (1995)[6]

[1] Karakas & Lugaro (2016), ApJ, 825, 26
[2] Karakas et al. (2018), MNRAS, 477, 421
[1] Limongi & Chieffi (2018), ApJS, 237, 13
[2] Sukhbold et al. (2016), ApJ, 821, 38
[3] Chieffi & Limongi (2013), ApJ, 764, 21
[4] Nomoto, Kobayashi & Tominaga (2013), ARA&A, 51, 457
[5] Chieffi & Limongi (2004), ApJ, 608, 405
[6] Woosley & Weaver (1995), ApJ, 101, 181

Contents

fractional [<function>] Calculate an IMF-averaged yield for a given element.

table [<function>] Obtain the table of mass yields and progenitor masses for a given element from a given study.

settings [dataframe] Stores current settings for these yields.

engines [module] Models for massive star explodability as a function of progenitor mass for use in yield calculations.

LC18 [module] Sets yields according to the Limongi & Chieffi (2018) study.

S16 [module] Sets the yields according to one of the explosion engines in the Sukhbold et al. (2016) study.

CL13 [module] Sets yields according to the Chieffi & Limongi (2013) study.

NKT13 [module] Sets yields according to the Nomoto, Kobayashi & Tominaga (2013) study.

CL04 [module] Sets yields according to the Chieffi & Limongi (2004) study.

WW95 [module] Sets yields according to the Woosley & Weaver (1995) study.

Notes

The yield tables built into this module reflect a post-processing treatment for two radioactive nuclides. For the Limongi & Chieffi (2018), Sukhbold et al. (2016), and Chieffi & Limongi (2013) tables, we add the nickel-56 yield to the iron yields and the aluminum-26 yield to the magnesium yields. We also add the aluminum-26 yield to the magnesium yields for the Woosley & Weaver (1995) study. Otherwise, the data are included *as published*. The Chieffi & Limongi (2013) and Nomoto, Kobayashi & Tominaga (2013) yields report yields after radioactive isotopes have been decayed, so no further treatment is necessary.

vice.yields.ccsne.fractional

Calculate an IMF-integrated fractional nucleosynthetic yield of a given element from core-collapse supernovae.

Signature: vice.yields.ccsne.fractional(element, study = "LC18", MoverH = 0, rotation = 0, explodability = None, wind = True, net = True, IMF = "kroupa", method = "simpson", m_lower = 0.08, m_upper = 100, tolerance = 1e-3, Nmin = 64, Nmax = 2.0e+08)

Parameters

element [str [case-insensitive]] The symbol of the element to calculate the IMF-integrated fractional yield for.

study [str [case-insensitive] [default]["LC18"]] A keyword denoting which study to adopt the yields from

Keywords and their Associated Studies:

- "LC18": Limongi & Chieffi (2018)[1]
- "S16/W18": Sukhbold et al. (2016)[2] (W18 explosion engine)
- "S16/W18F": Sukhbold et al. (2016) (W18 engine, forced explosions)
- "S16/N20": Sukhbold et al. (2016) (N20 explosion engine)
- "CL13": Chieffi & Limongi (2013)[3]

[1] Limongi & Chieffi (2018), ApJS, 237, 13
[2] Sukhbold et al. (2016), ApJ, 821, 38
[3] Chieffi & Limongi (2013), ApJ, 764, 21

- "NKT13": Nomoto, Kobayashi & Tominaga (2013)[4]
- "CL04": Chieffi & Limongi (2004)[5]
- "WW95": Woosley & Weaver (1995)[6]

Note: The S16/W18F yields force a supernova explosion in the progenitors which otherwise did not explode in the S16/W18 set, producing a yield sample for which all progenitors explode. This allows new black hole landscapes to be used in computing yields under similar explosion physics; the IMF-averaged S16/W18 yields are similar to what is computed with the S16/W18F yields and S16/W18 explodability function (see below). For details on the forced explosion models, see discussion in Griffith et al. (2021)[7].

MoverH [real number [default][0]] The total metallicity [M/H] = $\log_{10}(Z/Z_\odot)$ of the exploding stars. There are only a handful of metallicities recognized by each study.

Keywords and their Associated Metallicities:

- "LC18": [M/H] = -3, -2, -1, 0
- "S16/*": [M/H] = 0
- "CL13": [M/H] = 0
- "NKT13": [M/H] = -inf, -1.15, -0.54, -0.24, 0.15, 0.55
- "CL04": [M/H] = -inf, -4, -2, -1, -0.37, 0.15
- "WW95": [M/H] = -inf, -4, -2, -1, 0

rotation [real number [default][0]] The rotational velocity of the exploding stars in km/s. There are only a handful of rotational velocities recognized by each study.

Keywords and their Associated Rotational Velocities:

- "LC18": v = 0, 150, 300
- "S16/*": v = 0
- "CL13": v = 0, 300
- "NKT13": v = 0
- "CL04": v = 0
- "WW95": v = 0

explodability [<function> or `None` [default][`None`]] Stellar explodability as a function of mass. This function is expected to take stellar mass in $M_\odot$ as the only numberical parameter, and to return a number between 0 and 1 denoting the fraction of stars at that mass which explode as a CCSN.

New in version 1.2.0.

Tip: The vice.yields.ccsne.engines module provides a number of popular mathematical forms for the black hole landscape, both simple and complex.

Note: Explodability criteria will be overspecified when calculating yields from the Limongi & Chieffi (2018) study, in which stars above 25 $M_\odot$ were not forced to explode. The same applies to the W18 and N20 yield sets

[4] Nomoto, Kobayashi & Tominaga (2013), ARA&A, 51, 457
[5] Chieffi & Limongi (2004), ApJ, 608, 405
[6] Woosley & Weaver (1995), ApJ, 101, 181
[7] Griffith et al. (2021), ApJ, 921, 73

from the Sukhbold et al. (2016) study, for which the reported yields already reflect the more complicated black hole landscape predicted by the explosion engine.

wind [bool [default][`True`]] If True, the stellar wind contribution to the yield will be included in the yield calculation. If False, the calculation will run considering only the supernova explosion yield.

New in version 1.2.0.

Note: Wind and explosive yields are only separated for the Limongi & Chieffi (2018) and Sukhbold et al. (2016) studies. Wind yields are not separable from explosive yields or are not included for other studies supported by this function.

net [bool [default][`True`]] If True, the initial abundance of each simulated CCSN progenitor star will be subtracted from the gross yield to convert the reported value to a net yield.

New in version 1.2.0.

Note: Net yields cannot be calculated for the Woosley & Weaver (1995) study as they do not report birth abundances. Conversely, gross yields cannot be calculated for the Nomoto, Kobayashi & Tominaga (2013) study, because their data is already reported as net yields for each individual progenitor.

IMF [`str` [case-insensitive] or <function> [default][“kroupa”]] The stellar initial mass function (IMF) to assume. Strings denote built-in IMFs, which must be either “Kroupa”[8] or “Salpeter”[9]. Functions must accept stellar mass in $M_\odot$ as the only numerical paraneter and will be interpreted as a custom, arbitrary stellar IMF.

New in version 1.2.0: Prior to version 1.2.0, only the built-in Kroupa and Salpeter IMFs were supported.

method [`str` [case-insensitive] [default][“simpson”]] The method of quadrature.

Recognized Methods:

- “simpson”
- “trapezoid”
- “midpoint”
- “euler”

Note: These methods of quadrature are implemented according to Chapter 4 of Press, Teukolsky, Vetterling & Flannery (2007)[10].

m_lower [real number [default][0.08]] The lower mass limit on star formation in $M_\odot$.

m_upper [real number [default][100]] The upper mass limit on star formation in $M_\odot$.

tolerance [real number [default][0.001]] The numerical tolerance. VICE will not return a result until the fractional change between two successive integrations is smaller than this value.

Nmin [real number [default][64]] The minimum number of bins in quadrature.

Nmax [real number [default][2.0e+08]] The maximum number of bins in quadrature. Included as a failsafe against solutions that don’t converge numerically.

[8] Kroupa (2001), MNRAS, 231, 322
[9] Salpeter (1955), ApJ, 121, 161
[10] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press

Returns

y [real number] The numerically calculated yield.

error [real number] The estimated numerical error.

Raises

- **ValueError**
  - The element is not built into VICE
  - The study is not built into VICE
  - The tolerance is not between 0 and 1
  - m_lower > m_upper
  - Custom IMF does not accept exactly 1 positional argument
  - Built-in IMF is not recognized
  - The method of quadrature is not built into VICE
  - Nmin > Nmax
- **LookupError**
  - The study did not report yields at the specified metallicity
  - The study did not report yields at the specified rotational velocity.
- **ScienceWarning**
  - m_upper is larger than the largest mass on the grid reported by the specified study. VICE extrapolates to high masses in this case.

    The upper mass limits of each study:

    * Limongi & Chiefii (2018) : 120 $M_\odot$
    * Sukhbold et al (2016) : 120 $M_\odot$
    * Nomoto, Kobayash & Tominaga (2013) : 40 $M_\odot$
    * Chieffi & Limongi (2013) : 120 $M_\odot$
    * Chieffi & Limongi (2004) : 35 $M_\odot$
    * Woosley & Weaver (1995) : 40 $M_\odot$
  - study is either "CL04" or "CL13" and the atomic number of the element is between 24 and 28 (inclusive). VICE warns against adopting these yields for iron peak elements.
  - Numerical quadrature did not converge within the maximum number of allowed quadrature bins to within the specified tolerance.
  - `explodability` is not `None` and `study == "LC18"`, `"S16/N20"`, `"S16/W18"`. The mass yields these studies report are already under a given model for the black hole landscape.
  - `wind = False` and `study` is anything other than LC18 or S16. These are the only studies for which wind yields were reported separate from explosive yields.
  - `net == True` and `study == "WW95"`. The Woosley & Weaver (1995) study did not report a detailed initial composition of their model CCSN progenitors; VICE can only calculate gross yields in this instance.

- `net == False` and `study == NKT13`. The Nomoto, Kobayashi & Tominaga (2013) study reported net yields in their model core collapse supernova ejeta. VICE can only calculate net yields in this instance.

Notes

This function evaluates the solution to the following equation:

$$y_x^{\text{CC}} = \frac{\int_8^u (E(m)m_x + w_x - Z_{x,\text{prog}}m)\frac{dN}{dm}dm}{\int_l^u m\frac{dN}{dm}dm}$$

where $E(m)$ is the stellar explodability for progenitors of initial mass m, m_x is the mass of the element x produced in the explosion, w_x is the mass of the element x ejected in the wind, dN/dm is the assumed stellar IMF, and $Z_{x,\text{prog}}$ is the abundance by mass of the element x in the CCSN progenitor stars, whose values are stored in VICE's internal data. If the keyword arg `net = False`, $Z_{x,\text{prog}}$ is simply set to zero to calculate a gross yield.

If a study does not report wind yields, or doesn't separate them from the explosive yields (i.e. anything other than LC18 or S16/* yield sets), then w_x = 0 everywhere by definition. In these cases, VICE weights the term correcting for the birth abundance $Z_{x,\text{prog}}m$ by the explodability $E(m)$ for net yield calculations. Because high mass stars tend to return their birth abundance with the wind yield (see discussion in Griffith et al. 2021), w_x = 0 is unrealistic in these cases. Weighting $Z_{x,\text{prog}}m$ by $E(m)$ prevents this function from returning very negative net yields in this scenario, approximately correcting for the neglected return of the birth abundance in the wind. For gross yield calculations, this is unnecessary because VICE simply sets $Z_{x,\text{prog}}m$ = 0 anyway.

The above equation is evaluated always for stable isotopes *only*. See the notes in the `vice.yields.ccsne` docstring for details on the studies to which VICE applies a treatment of radioactive isotopes in its built-in tables.

Example Code

```
>>> y, err = vice.yields.ccsne.fractional("o")
>>> y
        0.004859197708207693
>>> err
        5.07267151987336e-06
>>> y, err = vice.yields.ccsne.fractional("mg", study = "CL13")
>>> y
        0.0009939371276697314
```

vice.yields.ccsne.table

Look up the mass yield of a given element as a function of stellar mass as reported by a given study.

Signature: vice.yields.ccsne.table(element, study = "LC18", MoverH = 0, rotation = 0, wind = True, isotopic = False)

New in version 1.2.0.

Parameters

element [`str` [case-insensitive]] The symbol of the element to look up the yield table for.

study [`str` [case-insensitive] [default]["LC18"]] A keyword denoting which study to pull the table from

Keywords and their Associated Studies:

- "LC18": Limongi & Chieffi (2018)[1]
- "S16/W18": Sukhbold et al. (2016)[2] (W18 explosion engine)
- "S16/W18F": Sukhbold et al. (2016) (W18 engine, forced explosions)
- "S16/N20": Sukhbold et al. (2016) (N20 explosion engine)
- "CL13": Chieffi & Limongi (2013)[3]
- "NKT13": Nomoto, Kobayashi & Tominaga (2013)[4]
- "CL04": Chieffi & Limongi (2004)[5]
- "WW95": Woosley & Weaver (1995)[6]

MoverH [real number [default][0]] The total metallicity [M/H] = $\log_{10}(Z/Z_\odot)$ of the exploding stars. There are only a handful of metallicities recognized by each study.

Keywords and their Associated Metallicities:

- "LC18": [M/H] = -3, -2, -1, 0
- "S16/*": [M/H] = 0
- "CL13": [M/H] = 0
- "NKT13": [M/H] = -inf, -1.15, -0.54, -0.24, 0.15, 0.55
- "CL04": [M/H] = -inf, -4, -2, -1, -0.37, 0.15
- "WW95": [M/H] = -inf, -4, -2, -1, 0

rotation [real number [default][0]] The rotational velocity of the exploding stars in km/s. There are only a handful of rotational velocities recognized by each study.

Keywords and their Associated Rotational Velocities:

- "LC18": v = 0, 150, 300
- "S16/*": v = 0
- "CL13": v = 0, 300
- "NKT13": v = 0
- "CL04": v = 0
- "WW95": v = 0

wind [bool [default][`True`]] If True, the stellar wind contribution to the yield will be included in the reported table. If False, the table will include only the supernova explosion yields.

[1] Limongi & Chieffi (2018), ApJS, 237, 13
[2] Sukhbold et al. (2016), ApJ, 821, 38
[3] Chieffi & Limongi (2013), ApJ, 764, 21
[4] Nomoto, Kobayashi & Tominaga (2013), ARA&A, 51, 457
[5] Chieffi & Limongi (2004), ApJ, 608, 405
[6] Woosley & Weaver (1995), ApJ, 101, 181

Note: Wind and explosive yields are only separated for the Limongi & Chieffi (2018) and Sukhbold et al. (2016) studies. Wind yields are not separable from explosive yields for other studies supported by this function.

isotopic [`bool` [default][`False`]] If `True`, the full-breakdown of isotopic mass yields is returned. If `False`, only the total mass yield of the given element is returned.

Returns

yields [`ccsn_yield_table` [VICE `dataframe` derived class]] A dataframe designed to hold a CCSN yield table. It can be indexed via stellar mass in $M_\odot$ or the isotopes of the requested element (if `isotopic == True`).

Raises

- **ValueError**
  - The element is not built into VICE
  - The study is not built into VICE
- **LookupError**
  - The study did not report yields for the requested element
  - The study did not report yields at the specified metallicity
  - The study did not report yields at the specified rotational velocity.
- **ScienceWarning**
  - `wind = False` and `study` is anything other than LC18 or S16. These are the only studies for which wind yields were reported separate from explosive yields.

Notes

The tables returned by this function will include stable isotopes *only*. See the notes in the `vice.yields.ccsne` docstring for details on the studies to which VICE applies a treatment of radioactive isotopes in its built-in tables.

Example Code

```
>>> import vice
>>> example = vice.yields.ccsne.table('o')
>>> example
vice.dataframe{
        13.0 -----------> 0.247071034
        15.0 -----------> 0.585730308
        20.0 -----------> 1.256452301
        25.0 -----------> 2.4764558329999997
        30.0 -----------> 0.073968147
        40.0 -----------> 0.087475695
        60.0 -----------> 0.149385561
        80.0 -----------> 0.24224373600000002
```

(continues on next page)

(continued from previous page)

```
        120.0 ----------> 0.368598602
}
>>> example.masses
(13.0, 15.0, 20.0, 25.0, 30.0, 40.0, 60.0, 80.0, 120.0)
>>> example[20.0]
1.256452301
>>> [example[i] for i in example.masses]
[0.2470691117,
 0.5857306186,
 1.256464291,
 2.476488843,
 0.073968147,
 0.087475695,
 0.149385561,
 0.24224373600000002,
 0.368598602]
>>> vice.yields.ccsne.table('o', isotopic = True)
vice.dataframe{
        13 -------------> {'o16': 0.24337, 'o17': 5.5634e-05, 'o18': 0.0036454}
        15 -------------> {'o16': 0.58234, 'o17': 5.5608e-05, 'o18': 0.0033347}
        20 -------------> {'o16': 1.2501, 'o17': 4.6201e-05, 'o18': 0.0063061}
        25 -------------> {'o16': 2.4724, 'o17': 4.7633e-05, 'o18': 0.0040082}
        30 -------------> {'o16': 0.073782, 'o17': 4.0707e-05, 'o18': 0.00014544}
        40 -------------> {'o16': 0.087253, 'o17': 4.1705e-05, 'o18': 0.00018099}
        60 -------------> {'o16': 0.1491, 'o17': 5.3011e-05, 'o18': 0.00023255}
        80 -------------> {'o16': 0.24192, 'o17': 6.1316e-05, 'o18': 0.00026242}
        120 ------------> {'o16': 0.36819, 'o17': 7.9192e-05, 'o18': 0.00032941}
}
>>> example[20]
vice.dataframe{
    o16 ------------> 1.250112
    o17 ------------> 4.6201e-05
    o18 ------------> 0.0063060899999999994
}
>>> example[20]['o16']
1.250112
>>> example['o16']
vice.dataframe{
    13.0 -----------> 0.2433681
    15.0 -----------> 0.5823402999999999
    20.0 -----------> 1.250112
    25.0 -----------> 2.472433
    30.0 -----------> 0.073782
    40.0 -----------> 0.087253
    60.0 -----------> 0.1491
    80.0 -----------> 0.24192
    120.0 ----------> 0.36819
}
```

vice.yields.ccsne.settings

The VICE dataframe: global settings for CCSN yields

For each chemical element, this object stores the current core collapse supernova (CCSN) nucleosynthetic yield setting. See *Notes* below for mathematical details.

Note: Modifying yield settings through this dataframe is equivalent to going through the `vice.elements` module.

Indexing

- **`str` [case-insensitive]** [elemental symbols] This dataframe must be indexed by the symbol of an element recognized by VICE as it appears on the periodic table.

Item Assignment

For each chemical element, the CCSN yield can be assigned either:

- real number : denotes a constant, metallicity-independent yield.
- **<function>** [Mathematical function describing the yield.] Must accept the metallicity by mass Z as the only parameter.

Functions

- keys
- todict
- restore_defaults
- factory_settings
- save_defaults

Notes

In all instances, VICE approximates CCSN enrichment to occur instantaneously according to an IMF-averaged yield. As with all other yields in VICE, these are *net* rather than *gross* yields in that they quantify only the mass of a given element which is newly produced. In one- and multi-zone chemical evolution models, the rate of enrichment due to CCSNe proceeds according to the following equation:

$$\dot{M}_{\rm CC} = y_{\rm CC}\dot{M}_\star$$

where $y_{\rm CC}$ is the yield assigned to a given element and $\dot{M}_\star$ is the star formation rate. For single stellar populations, the mass of some element produced by all CCSNe associated with that stellar population is given by:

$$M = y_{\rm CC}M_\star$$

where $M_\star$ is the total initial mass of the stellar population. For further details, see VICE's science documentation: https://vice-astro.readthedocs.io/en/latest/science_documentation/index.html.

Example Code

```
>>> import vice
>>> vice.yields.ccsne.settings["fe"] = 0.001
>>> vice.yields.ccsne.settings["Fe"]
        0.001
>>> vice.yields.ccsne.settings["FE"] = 0.0012
>>> vice.yields.ccsne.settings["fe"]
        0.0012
>>> def f(z):
        return 0.005 + 0.002 * (z / 0.014)
>>> vice.yields.ccsne.settings["fe"] = f
>>> vice.yields.ccsne.settings["FE"]
        <function __main__.f(z)>
```

vice.yields.ccsne.settings.keys

Returns the keys of the CCSN yield settings dataframe.

Signature: vice.yields.ccsne.settings.keys()

Note: By nature, this function will simply return a list of all elements that are built into VICE - the same thing as `vice.elements.recognized`.

Example Code

```
>>> import vice
>>> elements = vice.yields.ccsne.settings.keys()
>>> tuple(elements) == vice.elements.recognized
        True
```

vice.yields.ccsne.settings.todict

Returns the CCSN yield settings as a dictionary.

Signature: vice.yields.ccsne.settings.todict()

Note: Modifications to the dictionary returned by this function will *not* affect the global yield settings.

Note: Python dictionaries are case-sensitive, and are thus less flexible than this class.

Example Code

```
>>> import vice
>>> example = vice.yields.ccsne.settings.todict()
>>> example["c"]
        0.00236
>>> example["C"]
        Traceback (most recent call last):
          File "<stdin>", line 1, in <module>
        KeyError: 'C'
>>> example["c"] = "not a yield setting"
>>> example["c"]
        "not a yield setting"
>>> vice.yields.ccsne.settings["c"]
        0.00236
```

vice.yields.ccsne.settings.restore_defaults

Restores the CCSN yield settings to their default values (i.e. undoes any changes since VICE was imported).

Signature: vice.yields.ccsne.settings.restore_defaults()

Example Code

```
>>> import vice
>>> vice.yields.ccsne.settings["c"] = 0.0
>>> vice.yields.ccsne.settings["n"] = 0.0
>>> vice.yields.ccsne.settings["o"] = 0.0
>>> vice.yields.ccsne.settings.restore_defaults()
>>> vice.yields.ccsne.settings["c"]
        0.00236
>>> vice.yields.ccsne.settings["n"]
        0.000578
>>> vice.yields.ccsne.settings["o"]
        0.00564
```

vice.yields.ccsne.settings.factory_settings

Restores the CCSN yield settings to their factory defaults. This differs from `vice.yields.ccsne.settings.restore_defaults` in that users may modify their default values from those that VICE is distributed with.

Signature: vice.yields.ccsne.settings.factory_settings()

Tip: To revert your nucleosynthetic yield settings back to their production defaults *permanently*, simply call `vice.yields.ccsne.settings.save_defaults` immediately following this function.

Example Code

```
>>> import vice
>>> vice.yields.ccsne.settings["c"]
        0.001 # the user has modified their default yield for carbon
>>> vice.yields.ccsne.settings.factory_settings()
>>> vice.yields.ccsne.settings["c"]
        0.00236
```

vice.yields.ccsne.settings.save_defaults

Saves the current CCSN yield settings as the default values.

Signature: vice.yields.ccsne.settings.save_defaults()

Note: Saving functional yields requires the package dill, an extension to `pickle` in the python standard library. It is recommended that VICE users install dill >= 0.2.0.

Example Code

```
>>> import vice
>>> vice.yields.ccsne.settings["c"]
        0.00236
>>> vice.yields.ccsne.settings["c"] = 0.001
>>> vice.yields.ccsne.settings.save_defaults()
```

After re-launching the python interpreter:

```
>>> import vice
>>> vice.yields.ccsne.settings["c"]
        0.001
```

vice.yields.ccsne.engines

Core collapse supernova explosion engines: explodability as a function of progenitor mass in solar masses.

Signature: from vice.yields.ccsne import engines

New in version 1.2.0.

Tip: Instances of the `engine` class can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming a particular black hole landscape. The impact of these assumptions is explored in Griffith et al. (2021)[1].

---

[1] Griffith et al. (2021), ApJ, 921, 73

Contents

engine [type] The base class implemented in this module.

cutoff [engine] An engine characterized by a threshold mass, below which stars explode as CCSNe, and above which stars collapse directly to a black hole.

E16 [engine] An engine implementing the Ertl et al. (2016)[2] model, characterized by the two parameters M_4 and μ_4.

S16 [module] Explodability maps as reported by the various models in Sukhbold et al. (2016)[3].

vice.yields.ccsne.engines.engine

Core collapse supernova explosion engines: explodability as a function of progenitor mass in $M_\odot$.

Signature: vice.yields.ccsne.engines.engine(masses, frequencies)

New in version 1.2.0.

Tip: These objects can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming a particular black hole landscape.

Parameters

masses [list] The attribute `masses`. Though this isn't enforced, this is assumed to be sorted from least to greatest. See below.

frequencies [list] The attribute `frequencies`. See below.

Attributes

masses [list] The initial masses of core collapse supernova progenitors in $M_\odot$ on which the explosion engine is sampled.

frequencies [list] The fraction of stars at the sampled masses that explode as a core collapse supernova. Though this number may be anywhere between 0 and 1, the built-in engines in the current version are binary.

Note: The attributes `masses` and `frequencies` will not be modifiable after constructing an instance of this class.

---

[2] Ertl et al. (2016), ApJ, 818, 124
[3] Sukhbold et al. (2016), ApJ, 821, 38

Calling

Call this object with progenitor mass as the only argument, and the explodability as a float between 0 and 1 will be returned.

Parameters:

- **mass** [float] Progenitor zero age main sequence mass in $M_\odot$.

Returns:

- **explodability** [float] The fraction of stars at that mass which produce a core collapse supernova event according to the given explosion engine.

Note: The return value will be calculated via linear interpolation between masses and frequencies on the grid. With frequencies which are binary in the current version, non-binary explosion frequencies are only found at masses between a grid element which did explode in the supernova study and another which did not.

Indexing

Performs the same function as *Calling*.

See also:

Built-in instances of derived classes

- vice.yields.ccsne.engines.cutoff
- vice.yields.ccsne.engines.E16
- vice.yields.ccsne.engines.S16.N20
- vice.yields.ccsne.engines.S16.S19p8
- vice.yields.ccsne.engines.S16.W15
- vice.yields.ccsne.engines.S16.W18
- vice.yields.ccsne.engines.S16.W20

Example Code

```
>>> from vice.yields.ccsne.engines.S16 import W18
>>> W18.masses
[9.0,
 9.25,
 9.5,
 ...,
 80.0,
 100.0,
 120.0]
>>> W18.frequencies
[1.0,
 1.0,
 1.0,
```

(continues on next page)

(continued from previous page)

```
 ...,
 0.0,
 0.0,
 1.0]
>>> W18(20)
0.0
>>> W18(20.1)
1.0
>>> W18(20.05)
0.5
```

vice.yields.ccsne.engines.engine.masses

Type : `list`

The stellar masses in $M_\odot$ on which the explosion engine is sampled.

Example Code

```
>>> from vice.yields.ccsne.engines.S16 import W18
>>> W18.masses
[9.0,
 9.25,
 9.5,
 ...,
 80.0,
 100.0,
 120.0]
```

vice.yields.ccsne.engines.engine.frequencies

Type : `list`

The frequencies with which stars of a given mass explode; the progenitor zero age main sequence masses in $M_\odot$ are stored in the attribute `masses`. Though this number can be anywhere between 0 and 1, in the current version they are binary.

Example Code

```
>>> from vice.yields.ccsne.engines.S16 import W18
>>> W18.frequencies
[1.0,
 1.0,
 1.0,
 ...,
 0.0,
 0.0,
 1.0]
```

vice.yields.ccsne.engines.cutoff

A core collapse supernova explosion engine characterized by a threshold mass below which stars produce core collapse supernovae, and above which they collapse directly to a black hole.

Signature: from vice.yields.ccsne.engines import cutoff

New in version 1.2.0.

Tip: This object can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate the IMF-averaged yields assuming this black hole landscape.

Explosion models such as these have been explored by recent supernova nucleosynthesis studies (e.g. Limongi & Chieffi 2018[1]), and were compared to more sophisticated explosion models (e.g. Ertl et al. 2016[2]; Sukhbold et al. 2016[3]) by Griffith et al. (2021)[4].

Attributes

collapse_mass [`float` [default][40.0]] The progenitor ZAMS mass in $M_\odot$ above which stars collapse to a black hole. Must be positive.

Calling

Call this object with progenitor mass as the only argument, and the explodability as either a 0 or 1 will be returned.

Parameters:

- **mass** [`float`] Progenitor zero age main sequence mass in $M_\odot$.

Returns:

- **explodability** [`float`] 1 if 8 <= `mass` <= `collapse_mass`. 0 otherwise.

Example Code

```
>>> from vice.yields.ccsne.engines import cutoff
>>> cutoff.collapse_mass
40
>>> [cutoff(35), cutoff(45), cutoff(55)]
[1.0, 0.0, 0.0]
>>> cutoff.collapse_mass = 50
>>> [cutoff(35), cutoff(45), cutoff(55)]
[1.0, 1.0, 0.0]
```

[1] Limongi & Chieffi (2018), ApJS, 237, 13
[2] Ertl et al. (2016), ApJ, 818, 124
[3] Sukhbold et al. (2016), ApJ, 821, 38
[4] Griffith et al. (2021), MNRAS, 921, 73

vice.yields.ccsne.engines.cutoff.collapse_mass

Type : `float` [default : 40.0]

The progenitor zero age main sequence mass in $M_\odot$ above which stars collapse to a black hole, and below which they produce a core collapse supernova. Must be positive.

Example Code

```
>>> from vice.yields.ccsne.engines import cutoff
>>> cutoff.collapse_mass
40.0
>>> cutoff.collapse_mass = 50
>>> cutoff.collapse_mass
50.0
```

vice.yields.ccsne.engines.E16

Core collapse supernova explosion engine as calculated by Ertl et al. (2016)[1] and studied in Griffith et al. (2021)[2].

Signature: from vice.yields.ccsne.engines import E16

New in version 1.2.0.

Tip: This object can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming this black hole landscape.

Attributes

masses [`list`] The stellar masses in $M_\odot$ on which the explosion engine is sampled.

m4 [`list`] The quantity M_4 at the time of core collapse for stars with ZAMS masses given by the attribute `masses`. Values are taken from Sukhbold et al. (2016)[3]. See *Notes*.

mu4 [`list`] The quantity μ_4 at the time of core collapse for stars with ZAMS masses given by the attribute `masses`. Values are taken from Sukhbold et al. (2016). See *Notes*.

slope [`float` [default][0.283]] The slope of the line in $\mu_4 - \mu_4 M_4$ space dividing progenitors which explode and collapse.

intercept [`float` [default][0.043]] The intercept of the line in $\mu_4 - \mu_4 M_4$ space dividing progenitors which explode and collapse.

frequencies [`list`] The fraction of stars at the sampled masses that explode as a core collapse supernova. Though this attribute has meaning for other `engine` objects, here it simply returns the binary 0 or 1 describing whether or not each element of the attribute `masses` explodes under the current parameters.

[1] Ertl et al. (2016), ApJ, 818, 124
[2] Griffith et al. (2021), ApJ, 921, 73
[3] Sukhbold et al. (2016), ApJ, 821, 38

Calling

Call this object with progenitor mass as the only argument, and either 0 or 1 will be returned based on whether or not such a progenitor should explode based on the current attributes of this object.

Parameters:

- **mass** [float] Progenitor zero age main sequence mass in $M_\odot$.

Returns:

- **explodability** [float] 1 if the star would produce a core collapse supernova under the current assumptions, and 0 if it would collapse to a black hole.

Note: The base class `engine` will return non-binary values for masses between grid elements where the explosion frequency is 0 and 1. This derived class, however, will always return a binary value.

Notes

The base class `engine` determines explodability by linearly interpolating between elements of the grid defined by the attributes `masses` and `frequencies`. This class, however, linearly interpolates on the `masses-m4` and `masses-mu4` grids to determine M_4 and μ_4 at a given progenitor zero age main sequence (ZAMS) mass. Whether or not this progenitor would explode is then determined according to the following criterion:

$$\mu_4 \leq aM_4\mu_4 + b$$

where a and b are the attributes `slope` and `intercept`. Default values of a = 0.283 and b = 0.043 are adopted from Ertl et al. (2016), while the `m4` and `mu4` tables are taken from Sukhbold et al. (2016). The attribute `frequencies` plays no role in this calculation.

Example Code

```
>>> from vice.yields.ccsne.engines import E16
>>> E16.masses
[9.0,
 9.25,
 9.5,
 ...,
 80.0,
 100.0,
 120.0]
>>> E16.m4
[1.357,
 1.36955,
 1.38349,
 ...,
 1.66232,
 1.81718,
 1.60252]
>>> E16.mu4
```

(continues on next page)

(continued from previous page)

```
[1.79891e-05,
 0.000933825,
 0.00210795,
 ...,
 0.0896075,
 0.103897,
 0.0759102]
>>> E16(20)
0.0
>>> E16(21)
1.0
>>> E16(22)
0.0
```

vice.yields.ccsne.engines.E16.m4

Type : `list` [elements of type `float`]

The quantity M_4 at the time of core collapse for stars with ZAMS masses given by the attribute `masses`. M_4 is defined as the normalized mass enclosed in a region defined by the dimensionless energy per nucleon of s = 4.

Ertl et al. (2016)[1] conclude that the quantities M_4 and μ_4 can predict whether or not a massive star will produce a core collapse supernova based on the following criterion with only a few exceptions (~1 - 2.5%):

$$\mu_4 \leq aM_4\mu_4 + b$$

where they report a = 0.283 and b = 0.043 as best fit values. The values of a and b are controlled by the attributes `slope` and `intercept` respectively.

Note: This attribute's values are adopted from Sukhbold et al. (2016)[2].

Example Code

```
>>> from vice.yields.ccsne.engines import E16
>>> E16.m4
[1.357,
 1.36955,
 1.38349,
 ...,
 1.66232,
 1.81781,
 1.60252]
```

[1] Ertl et al. (2016), ApJ, 818, 124
[2] Sukhbold et al. (2016), ApJ, 821, 38

vice.yields.ccsne.engines.E16.mu4

Type : `list` [elements of type `float`]

The quantity μ_4 at the time of core collapse for stars with ZAMS masses given by the attribute `masses`. μ_4 is defined as the normalized mass derivative in a region defined by the dimensionless energy per nucleon of s = 4:

$$\mu_4 \equiv (dm/M_\odot)/(dr/1000km)|_{s=4}$$

Ertl et al. (2016)[1] conclude that the quantities M_4 and μ_4 can predict whether or not a massive star will produce a core collapse supernova based on the following criterion with only a few exceptions (~1 - 2.5%):

$$\mu_4 \leq aM_4\mu_4 + b$$

where they report a = 0.283 and b = 0.043 as best fit values. The values of a and b are controlled by the attributes `slope` and `intercept`, respectively.

Note: This attribute's values are adopted from Sukhbold et al. (2016)[2].

Example Code

```
>>> from vice.yields.ccsne.engines import E16
>>> E16.mu4
[1.79891e-05,
 0.000933825,
 0.00210795,
 ...,
 0.0896075,
 0.103897,
 0.0759102]
```

vice.yields.ccsne.engines.E16.slope

Type : `float` [default : 0.283]

The slope of the line in $\mu_4 - M_4\mu_4$ space dividing progenitors which explode and collapse. Ertl et al. (2016)[1] argue that the quantities M_4 and μ_4 can predict whether or not a massive star will produce a core collapse supernova based on the following criterion with only a few exceptions (~1 - 2.5%):

$$\mu_4 \leq aM_4\mu_4 + b$$

where this attribute encodes the value of a.

[1] Ertl et al. (2016), ApJ, 818, 124
[2] Sukhbold et al. (2016), ApJ, 821, 38
[1] Ertl et al. (2016), ApJ, 818, 124

Example Code

```
>>> from vice.yields.ccsne.engines import E16
>>> E16.slope
0.283
>>> E16.slope = 0.3
>>> E16.slope
0.3
>>> E16.slope = 0.25
>>> E16.slope
0.25
```

vice.yields.ccsne.engines.E16.intercept

Type : `float` [default : 0.043]

The intercept of the line in $\mu_4 - M_4\mu_4$ space dividing progenitors which explode and collapse. Ertl et al. (2016)[1] argue that the quantities M_4 and μ_4 can predict whether or not a massive star will produce a core collapse supernova based on the following criterion with only a few exceptions (~1 - 2.5%):

$$\mu_4 \leq aM_4\mu_4 + b$$

where this attribute encodes the value of b.

Example Code

```
>>> from vice.yields.ccsne.engines import E16
>>> E16.intercept
0.043
>>> E16.intercept = 0.04
>>> E16.intercept
0.04
>>> E16.intercept = 0.045
>>> E16.intercept
0.045
```

vice.yields.ccsne.engines.S16

Core collapse supernova explosion engines: explodability as a function of progenitor mass in solar masses as reported by the Sukhbold et al. (2016)[1] models.

Signature: from vice.yields.ccsne.engines import S16

New in version 1.2.0.

Tip: Instances of the `engine` class can be passed the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming a particular black hole landscape. The impact of these as-

[1] Ertl et al. (2016), ApJ, 818, 124
[1] Sukhbold et al. (2016), ApJ, 821, 38

sumptions is explored in Griffith et al. (2021)[2].

Note: For all explosion engines, progenitors with zero age main sequence masses between 9 and 12 $M_\odot$ proceed according to the Z9.6 engine, while remaining masses explode or collapse according to the associated engine. (See: Section 2.2.2 of Sukhbold et al. 2016)

Contents

N20 [engine] An engine characterized by the N20 explosion model.

S19p8 [engine] An engine characterized by the S19p8 explosion model.

W15 [engine] An engine characterized by the W15 explosion model.

W18 [engine] An engine characterized by the W18 explosion model.

W20 [engine] An engine characterized by the W20 explosion model.

vice.yields.ccsne.engines.S16.N20

N20 core collapse supernova explosion engine as reported by Sukhbold et al. (2016)[1].

Signature: from vice.yields.ccsne.engines.S16 import N20

New in version 1.2.0.

Tip: This object can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming this black hole landscape.

This object inherits its functionality from the base class `vice.yields.ccsne.engines.engine` with the attribute `frequencies` assigned according to the N20 explosion engine. See the associated documentation for further details.

vice.yields.ccsne.engines.S16.S19p8

S19.8 core collapse supernova explosion engine as reported by Sukhbold et al. (2016)[1].

Signature: from vice.yields.ccsne.engines.S16 import S19p8

New in version 1.2.0.

Tip: This object can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming this black hole landscape.

This object inherits its functionality from the base class `vice.yields.ccsne.engines.engine` with the attribute `frequencies` assigned according to the S19.8 explosion engine. See the associated documentation for further details.

[2] Griffith et al. (2021), ApJ, 921, 73
[1] Sukhbold et al. (2016), ApJ, 821, 38
[1] Sukhbold et al. (2016), ApJ, 821, 38

vice.yields.ccsne.engines.S16.W15

W15 core collapse supernova explosion engine as reported by Sukhbold et al. (2016)[1].

Signature: from vice.yields.ccsne.engines.S16 import W15

New in version 1.2.0.

Tip: This object can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming this black hole landscape.

This object inherits its functionality from the base class `vice.yields.ccsne.engines.engine` with the attribute `frequencies` assigned according to the W15 explosion engine. See the associated documentation for further details.

vice.yields.ccsne.engines.S16.W18

W18 core collapse supernova explosion engine as reported by Sukhbold et al. (2016)[1].

Signature: from vice.yields.ccsne.engines.S16 import W18

New in version 1.2.0.

Tip: This object can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming this black hole landscape.

This object inherits its functionality from the base class `vice.yields.ccsne.engines.engine` with the attribute `frequencies` assigned according to the W18 explosion engine. See the associated documentation for further details.

vice.yields.ccsne.engines.S16.W20

W20 core collapse supernova explosion engine as reported by Sukhbold et al. (2016)[1].

Signature: from vice.yields.ccsne.engines.S16 import W20

New in version 1.2.0.

Tip: This object can be passed as the keyword argument `explodability` to `vice.yields.ccsne.fractional` to calculate IMF-averaged yields assuming this black hole landscape.

This object inherits its functionality from the base class `vice.yields.ccsne.engines.engine` with the attribute `frequencies` assigned according to the W20 explosion engine. See the associated documentation for further details.

[1] Sukhbold et al. (2016), ApJ, 821, 38
[1] Sukhbold et al. (2016), ApJ, 821, 38
[1] Sukhbold et al. (2016), ApJ, 821, 38

vice.yields.ccsne.WW95

Woosley & Weaver (1995), ApJ, 101, 181 core collapse supernova (CCSN) yields

Signature: from vice.yields.ccsne import WW95

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Woosley & Weaver (1995) yield table for [M/H] = 0 stars. This will adopt an upper mass limit of 40 $M_\odot$.

We provide core collapse supernova yields for non-rotating progenitors as reported by Woosley & Weaver (1995) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -inf
- -4
- -2
- -1
- 0

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.ccsne.WW95.set_params

Update the parameters with which the yields are calculated from the Woosley & Weaver (1995)[1] data.

Signature: vice.yields.ccsne.WW95.set_params(**kwargs)

[1] Woosley & Weaver (1995), ApJ, 101, 181

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "WW95" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

Notes

We provide core collapse supernova yields for non-rotating progenitors as reported by Woosley & Weaver (1995) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -inf
- -4
- -2
- -1
- 0

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import WW95
>>> vice.yields.ccsne.settings['o']
0.011213287529967898
>>> WW95.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.01031925181253855
>>> WW95.set_params(IMF = "salpeter", m_upper = 80)
>>> vice.yields.ccsne.settings['o']
0.009732682861255278
>>> from vice.yields.ccsne.engines.S16 import W18
>>> from vice.yields.ccsne.engines import E16
>>> # Sukhbold et al. (2016) W18 black hole landscape with WW95 yields
... WW95.set_params(explodability = W18)
>>> vice.yields.ccsne.settings['o']
0.004076427850896676
>>> # Ertl et al. (2016) black hole landscape with WW95 yields
... WW95.set_params(explodability = E16)
>>> vice.yields.ccsne.settings['o']
0.0026048168655224023
```

See also:

vice.yields.ccsne.fractional

vice.yields.ccsne.CL04

Chieffi & Limongi (2014), ApJ, 608, 405 core collapse supernova (CCSN) yields

Signature: from vice.yields.ccsne import CL04

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Chieffi & Limongi (2004) yield table for [M/H] = 0.15 stars. This will adopt an upper mass limit of 35 $M_\odot$.

We provide core collapse supernova yields for non-rotating progenitors as reported by Chieffi & Limongi (2004) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -inf
- -4
- -2
- -1
- -0.37
- 0.15

assuming $Z_\odot$ = 0.014 according to Asplund et al. (2009)[1].

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.ccsne.CL04.set_params

Update the parameter with which the yields are calculated from the Chieffi & limongi (2004)[1] data.

Signature: vice.yields.ccsne.CL04.set_params(**kwargs)

[1] Asplund et al. (2009), ARA&A, 47, 481
[1] Chieffi & Limongi (2004), ApJ, 608, 405

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "CL04" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

Notes

We provide core collapse supernova yields for non-rotating progenitors as reported by Chieffi & Limongi (2004) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -inf
- -4
- -2
- -1
- -0.37
- 0.15

assuming $Z_\odot$ = 0.014 according to Asplund et al. (2009)[2].

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import CL04
>>> # negative yields simply imply a net loss
>>> vice.yields.ccsne.settings['o']
-0.07258767478609592
>>> CL04.set_params(MoverH = -4)
>>> vice.yields.ccsne.settings['o']
0.01835902419240956
>>> CL04.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
-0.056090913652505486
>>> CL04.set_params(IMF = "salpeter", m_upper = 60)
>>> vice.yields.ccsne.settings['o']
-0.0512909951164916
>>> from vice.yields.ccsne.engines.S16 import W18
>>> from vice.yields.ccsne.engines import E16
>>> # Sukhbold et al. (2016) W18 black hole landscape with CL04 yields
... CL04.set_params(explodability = W18)
>>> vice.yields.ccsne.settings['o']
```

(continues on next page)

[2] Asplund et al. (2009), ARA&A, 47, 481

(continued from previous page)

```
-0.049690600129938464
>>> # Ertl et al. (2016) black hole landscape with CL04 yields
... CL04.set_params(explodability = E16, MoverH = -4)
>>> vice.yields.ccsne.settings['o']
0.001936110535019444
```

See also:

- vice.yields.ccsne.fractional
- vice.yields.ccsne.table

vice.yields.ccsne.CL13

Chieffi & Limongi (2013), ApJ, 764, 21 core collapse supernova (CCSN) yields

Signature: from vice.yields.ccsne import CL13

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Chieffi & Limongi (2013) yield table for [M/H] = 0 stars. This will adopt an upper mass limit of 100 $M_\odot$.

We provide core collapse supernova yields for progenitors at rotational velocities of 0 and 300 km/s as reported by Chieffi & Limongi (2013) at solar metallicity.

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.ccsne.CL13.set_params

Update the parameters with which the yields are calculated from the Chieffi & Limongi (2013)[1] data.

Signature: vice.yields.ccsne.CL13.set_params(**kwargs)

[1] Chieffi & Limongi (2013), ApJ, 764, 21

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "CL13" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

Notes

We provide core collapse supernova yields for progenitors at rotational velocities of 0 and 300 km/s as reported by Chieffi & Limongi (2013) at solar metallicity.

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import CL13
>>> vice.yields.ccsne.settings['o']
0.0155974093648906 2
>>> CL13.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.009261587578694157
>>> CL13.set_params(rotation = 300)
>>> vice.yields.ccsne.settings['o']
0.024372870613029975
>>> CL13.set_params(rotation = 300, IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.014655521027938309
>>> from vice.yields.ccsne.engines.S16 import W18
>>> from vice.yields.ccsne.engines import E16
>>> # Sukhbold et al. (2016) W18 black hole landscape with CL13 yields
... CL13.set_params(explodability = W18)
>>> vice.yields.ccsne.settings['o']
0.0029448935889672184
>>> # Ertl et al. (2016) black hole landscape with CL13 yields
... CL13.set_params(explodability = E16, rotation = 300)
>>> vice.yields.ccsne.settings['o']
0.005392906178396455
```

See also:

vice.yields.ccsne.fractional

vice.yields.ccsne.NKT13

Nomoto, Kobayashi & Tominaga (2013), ARA&A, 51, 457 core collapse supernova yields

Signature: from vice.yields.ccsne import NKT13

New in version 1.2.0.

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Nomoto, Kobayashi & Tominaga (2013) yield table for [M/H] = 0.15 stars. This will adopt an upper mass limit of 40 $M_\odot$.

We provide core collapse supernova yields for non-rotating progenitors as reported Nomoto, Kobayashi & Tominaga (2013) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -inf
- -1.15
- -0.54
- -0.24
- 0.15
- 0.55

assuming $Z_\odot$ = 0.014 according to Asplund et al. (2009)[1].

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calcualted.

vice.yields.ccsne.NKT13.set_params

Update the parameters with which the yields are calculated from the Nomoto, Kobayashi & Tominaga (2013)[1] data.

Signature: vice.yields.ccsne.NKT13.set_params(**kwargs)

[1] Asplund et al. (2009), ARA&A, 47, 481
[1] Nomoto, Kobayashi & Tominaga (2013), ARA&A, 51, 457

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "NKT13" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

Notes

We provide core collapse supernova yields for non-rotating progenitors as reported Nomoto, Kobayashi & Tominaga (2013) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -inf
- -1.15
- -0.54
- -0.24
- 0.15
- 0.55

assuming $Z_\odot$ = 0.014 according to Asplund et al. (2009)[2].

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import NKT13
>>> vice.yields.ccsne.settings['o']
0.00849047694119129
>>> NKT13.set_params(MoverH = 0.55)
>>> vice.yields.ccsne.settings['o']
0.02069201811708860 6
>>> NKT13.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.013645466478054947
>>> NKT13.set_params(IMF = "salpeter", MoverH = -1.15)
>>> vice.yields.ccsne.settings['o']
0.015010918224009764
>>> from vice.yields.ccsne.engines.S16 import W18
>>> from vice.yields.ccsne.engines import E16
>>> # Sukhbold et al. (2016) W18 black hole landscape with NKT13 yields
... NKT13.set_params(explodability = W18)
>>> vice.yields.ccsne.settings['o']
0.003980895857851888
```

(continues on next page)

[2] Asplund et al. (2009), ARA&A, 47, 481

(continued from previous page)

```
>>> # Ertl et al. (2016) black hole landscape with NKT13 yields
... NKT13.set_params(explodability = E16, MoverH = -1.15)
>>> vice.yields.ccsne.settings['o']
0.0023433951760400543
```

See also:

vice.yields.ccsne.fractional

vice.yields.ccsne.S16

Sukhbold et al. (2016), ApJ, 821, 38 core collapse supernova yields

Signature: from vice.yields.ccsne import S16

New in version 1.2.0.

This yield module provides a number of sub-modules with which to update CCSN yield settings according to the Sukhbold et al. (2016) study. This module, however, does not import them as doing so would update the yields multiple times. Users should instead import one of the modules listed below under *Contents*.

Tip: By importing this module or any of its sub-modules, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Contents

N20 [module] Sets yields according to the N20 explosion engine.

W18 [module] Sets yields according to the W18 explosion engine.

W18F [module] Sets yields according to the W18 engine with forced explosions.

For details on the W18F engine, see discussion in Griffith et al. (2021)[1].

vice.yields.ccsne.S16.N20

Sukhbold et al. (2016), ApJ, 821, 38 N20 Explosion Engine

Signature: from vice.yields.ccsne.S16 import N20

New in version 1.2.0.

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Sukhbold et al. (2016) yield table under the N20 explosion engine for [M/H] = 0 stars. This will adopt an upper mass limit of 120 $M_\odot$.

We provide core collapse supernova yields for non-rotating progenitors at solar metallicity only as reported by Sukhbold et al. (2016) under the N20 explosion engine.

[1] Griffith et al. (2021), ApJ, 921, 73

Tip: By importing this module the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.ccsne.S16.N20.set_params

Update the parameters with which the yields are calculated from the Sukhbold et al. (2016)[1] N20 explosion engine data.

Signature: vice.yields.ccsne.S16.N20.set_params(**kwargs)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "S16/N20" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

Notes

We provide core collapse supernova yields for non-rotating progenitors at solar metallicity only as reported by Sukhbold et al. (2016) under the N20 explosion engine.

[1] Sukhbold et al. (2016), ApJ, 821, 38

Example Code

```
>>> import vice
>>> from vice.yields.ccsne.S16 import N20
>>> vice.yields.ccsne.settings['o']
0.007286488814947039
>>> N20.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.004247585262396765
>>> N20.set_params(IMF = "salpeter", m_upper = 80)
>>> vice.yields.ccsne.settings['o']
0.003960345954142228
```

See also:

vice.yields.ccsne.fractional

vice.yields.ccsne.S16.W18

Sukhbold et al. (2016), ApJ, 821, 38 W18 Explosion Engine

Signature: from vice.yields.ccsne.S16 import W18

New in version 1.2.0.

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Sukhbold et al. (2016) yield table under the W18 explosion engine for [M/H] = 0 stars. This will adopt an upper mass limit of 120 $M_\odot$.

We provide core collapse supernova yields for non-rotating progenitors at solar metallicity only as reported by Sukhbold et al. (2016) under the W18 explosion engine.

Tip: By importing this module the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.ccsne.S16.W18.set_params

Update the parameters with which the yields are calculated from the Sukhbold et al. (2016)[1] W18 explosion engine data.

Signature: vice.yields.ccsne.S16.W18.set_params(**kwargs)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "S16/W18" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

Notes

We provide core collapse supernova yields for non-rotating progenitors at solar metallicity only as reported by Sukhbold et al. (2016) under the W18 explosion engine.

Example Code

```
>>> import vice
>>> from vice.yields.ccsne.S16 import W18
>>> vice.yields.ccsne.settings['o']
0.00574509645756458
>>> W18.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.0033812925649179715
>>> W18.set_params(IMF = "salpeter", m_upper = 80)
>>> vice.yields.ccsne.settings['o']
0.0032910365772603626
```

See also:

vice.yields.ccsne.fractional

[1] Sukhbold et al. (2016), ApJ, 821, 38

vice.yields.ccsne.S16.W18F

Sukhbold et al. (2016), ApJ, 821, 38 W18 Engine with Forced Explosions

Signature: from vice.yields.ccsne.S16 import W18F

New in version 1.2.0.

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Sukhbold et al. (2016) yield table under the W18F explosion engine for [M/H] = 0 stars. This will adopt an upper mass limit of 120 $M_\odot$.

We provide core collapse supernova yields for non-rotating progenitors at solar metallicity only as reported by Sukhbold et al. (2016) under the W18F explosion engine.

For details on the nature of the forced explosion model, see discussion in Griffith et al. (2021)[1].

Tip: By importing this module the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.ccsne.S16.W18F.set_params

Update the parameters with which the yields are calculated from the Sukhbold et al. (2016)[1] W18F explosion engine data.

Signature: vice.yields.ccsne.S16.W18F.set_params(**kwargs)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

[1] Griffith et al. (2021), ApJ, 921, 73
[1] Sukhbold et al. (2016), ApJ, 821, 38

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "S16/W18F" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

Notes

We provide core collapse supernova yields for non-rotating progenitors at solar metallicity only as reported by Sukhbold et al. (2016) under the W18F explosion engine.

Example Code

```
>>> import vice
>>> from vice.yields.ccsne.S16 import W18F
>>> vice.yields.ccsne.settings['o']
0.018063484040223107
>>> W18F.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.010770572437600351
>>> W18F.set_params(IMF = "salpeter", m_upper = 80)
>>> vice.yields.ccsne.settings['o']
0.010523432438733342
>>> from vice.yields.ccsne.engines import E16
>>> from vice.yields.ccsne.engines.S16 import W20
>>> # Ertl et al. (2016) black hole landscape with W18F yields
... W18F.set_params(explodability = E16)
>>> vice.yields.ccsne.settings['o']
0.0044613656957736055
>>> # Sukhbold et al. (2016) W20 black hole landscape with W18F yields
... W18F.set_params(explodability = W20)
>>> vice.yields.ccsne.settings['o']
0.0034641626751717044
```

See also:

vice.yields.ccsne.fractional

vice.yields.ccsne.LC18

Limongi & Chieffi (2018), ApJS, 237, 13 core collapse supernova (CCSN) yields

Signature: from vice.yields.ccsne import LC18

Importing this module will automatically set the CCSN yield settings for all elements to the IMF-averaged yields calculated with the Limongi & Chieffi (2018) yield table for [M/H] = 0 stars. This will adopt an upper mass limit of 100 $M_\odot$.

We provide core collapse supernova yields for progenitors at rotational velocities of 0, 150, and 300 km/s as reported by Limongi & Chieffi (2018) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -3
- -2
- -1
- 0

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Note: When this module is imported, the yields will be updated with a maximum of 10^5 bins in quadrature to decrease computational overhead. For some elements, the yield calculation may not converge. To rerun the yield calculation with higher numerical precision, simply call `set_params` with a new value for the keyword `Nmax` (see below).

Contents

set_params [<function>] Update the parameters with which the yields are calculated

vice.yields.ccsne.LC18.set_params

Update the parameters with which the yields are calculated from the Limongi & Chieffi (2018)[1] data.

Signature: vice.yields.ccsne.LC18.set_params(**kwargs)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "LC18" when called from this module.

Other exceptions are raised by vice.yields.ccsne.fractional.

[1] Limongi & Chieffi (2018), ApJS, 237, 17

Notes

We provide core collapse supernova yields for progenitors at rotational velocities of 0, 150, and 300 km/s as reported by Limongi & Chieffi (2018) at metallicities relative to solar of $\log_{10}(Z/Z_\odot)$ =

- -3
- -2
- -1
- 0

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import LC18
>>> vice.yields.ccsne.settings['o']
0.0036512768277795864
>>> LC18.set_params(IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.002255625777096071 3
>>> LC18.set_params(rotation = 150)
>>> vice.yields.ccsne.settings['o']
0.010851696623329273
>>> LC18.set_params(rotation = 150, IMF = "salpeter")
>>> vice.yields.ccsne.settings['o']
0.006744726174198793
>>> LC18.set_params(rotation = 150, IMF = "salpeter", MoverH = -2)
>>> vice.yields.ccsne.settings['o']
0.0081235534350932
```

See also:

vice.yields.ccsne.fractional

vice.yields.sneia

Type Ia Supernovae (SNe Ia) Nucleosynthetic Yield Tools

Calculate IMF-averaged yields and modify yield settings for use in simulations. This package provides tables from the following nucleosynthetic yield studies:

- Seitenzahl et al. (2013)[1]
- Iwamoto et al. (1999)[2]
- Gronow et al. (2021a, b)[3][4]

[1] Seitenzahl et al. (2013), MNRAS, 429, 1156
[2] Iwamoto et al. (1999), ApJ, 124, 439
[3] Gronow et al. (2021a), A&A, 649, 155
[4] Gronow et al. (2021b), arxiv:2103.14050

Contents

fractional [<function>] Calculate an IMF-averaged yield for a given element.

single [<function>] Look up the mass yield of a given element from a single type Ia supernova from a specified study.

settings [dataframe] Stores current settings for these yields.

gronow21 [yield preset] Sets the yields according to one of the models published in the Gronow et al. (2021a, b) studies.

seitenzahl13 [yield preset] Sets the yields according to one of the models published in the Seitenzahl et al. (2013) study.

iwamoto99 [yield preset] Sets the yields according to one of the models published in the Iwamoto et al. (1999) study.

New in version 1.3.0: The "gronow21" yield model was introduced in version 1.3.0.

Notes

The data stored in this module are reported for each corresponding study *as published*. The Seitenzahl et al. (2013) and Gronow et al. (2021a, b) models reported mass yields after complete decay of all radioactive nuclides with half-lives less than 2 Gyr, and the Iwamoto et al. (1999) study fully decayed *all* unstable isotopes; any additional treatment for radioactive isotopes is thus unnecessary.

vice.yields.sneia.single

Lookup the mass yield of a given element from a single instance of a type Ia supernova (SN Ia) as determined by a specified study and explosion model.

Signature: vice.yields.sneia.single(element, study = "seitenzahl13", model = "N1")

Parameters

element [`str` [case-insensitive]] The symbol of the element to look up the yield for.

study [`str` [case-insensitive] [default]["seitenzahl13"]] A keyword denoting which study to adopt the yield from

Keywords and their Associated Studies:

- "seitenzahl13" : Seitenzahl et al. (2013)[1]
- "iwamoto99" : Iwamoto et al. (1999)[2]
- "gronow21" : Gronow et al. (2021a, b)[3][4]

model [`str` [case-insensitive] [default][N1]] A keyword denoting the explosion model from the associated study to adopt.

Keywords and their Associated Models:

- **"seitenzahl13"** [N1, N3, N5, N10, N20, N40, N100H, N100,] N100L, N150, N200, N300C, N1600, N1600C, N100_Z0.5, N100_Z0.1, N100_Z0.01
- "iwamoto99" : W7, W70, WDD1, WDD2, WDD3, CDD1, CDD2

[1] Seitenzahl et al. (2013), MNRAS, 429, 1156
[2] Iwamoto et al. (1999), ApJ, 124, 439
[3] Gronow et al. (2021a), A&A, 649, 155
[4] Gronow et al. (2021b), arxiv:2103.14050

- "gronow21" : M08_03_001, M08_03_01, M08_03_1, M08_03_3,
  M08_05_001, M08_05_01, M08_05_1, M08_05_3,
  M08_10_001, M08_10_01, M08_10_1, M08_10_3,
  M09_03_001, M09_03_01, M09_03_1, M09_03_3,
  M09_05_001, M09_05_01, M09_05_1, M09_05_3,
  M09_10_001, M09_10_01, M09_10_1, M09_10_3,
  M10_02_001, M10_02_01, M10_02_1, M10_02_3,
  M10_03_001, M10_03_01, M10_03_1, M10_03_3,
  M10_05_001, M10_05_01, M10_05_1, M10_05_3,
  M10_10_001, M10_10_01, M10_10_1, M10_10_3,
  M11_05_001, M11_05_01, M11_05_1, M11_05_3

Returns

y [real number] The mass yield of the given element in $M_\odot$ under the specified explosion model as reported by the nucleosynthesis study.

Raises

- **ValueError**
  - The element is not built into VICE
  - The study is not built into VICE
- **LookupError**
  - The study is recognized, but the model is not recognized for that particular study.
- **IOError [Occurs only if VICE's file structure has been modified]**
  - The data file is not found.

Notes

The data stored in this module are reported for each corresponding study *as published*. The Seitenzahl et al. (2013) and Gronow et al. (2021a, b) models reported mass yields after complete decay of all radioactive nuclides with half-lives less than 2 Gyr, and the Iwamoto et al. (1999) study fully decayed *all* unstable isotopes; any additional treatment for radioactive isotopes is thus unnecessary.

The Gronow et al. (2021a, b) models are named for the mass of the carbon-oxygen core, the mass of the helium shell, and the metallicity of the progenitor relative to solar, in that order. For example, the "M09_05_01" model refers to one with a 0.9 $M_\odot$ carbon-oxygen core and a 0.05 $M_\odot$ helium shell produced by a star that was initially at a metallicity of 0.1 $Z_\odot$.

Example Code

```
>>> import vice
>>> vice.yields.sneia.single("fe")
1.17390714
>>> vice.yields.sneia.single("fe", study = "iwamoto99", model = "W70")
0.77516
>>> vice.yields.sneia.single("fe", study = "iwamoto99", model = "CDD1")
0.6479629999999998
>>> vice.yields.sneia.single("ni", model = "n1001")
0.0391409000000526
>>> vice.yields.sneia.single("ni", model = "N150")
0.0749891244
>>> vice.yields.sneia.single("co", study = "gronow21", model = "M10_10_1")
0.001058
>>> vice.yields.sneia.single("co", study = "gronow21", model = "M09_05_001")
0.0001572
```

See also:

- vice.yields.sneia.fractional
- vice.yields.sneia.gronow21
- vice.yields.sneia.iwamoto99
- vice.yields.sneia.seitenzahl13

vice.yields.sneia.fractional

Calculate a delay-time distribution integrated fractional nucleosynthetic yield of a given element from type Ia supernovae.

Signature: vice.yields.sneia.fractional(element, study = "seitenzahl13", model = "N1", n = 2.2e-03)

Parameters

element [`str` [case-insensitive]] The symbol of the element to calculate the yield for.

study [`str` [case-sensitive] [default]["seitenzahl13"]] A keyword denoting which study to adopt SN Ia mass yields from.

Keywords and their Associated Studies:

- "seitenzahl13" : Seitenzahl et al. (2013)[1]
- "iwamoto99" : Iwamoto et al. (1999),[2]
- "gronow21" : Gronow et al. (2021a, b)[3][4]

model [`str` [case-insensitive] [default]["N1"]] The model from the associated study to adopt.

Keywords and their Associated Models:

[1] Seitenzahl et al. (2013), MNRAS, 429, 1156
[2] Iwamoto et al. (1999), ApJ, 124, 439
[3] Gronow et al. (2021a), A&A, 649, 155
[4] Gronow et al. (2021b), arxiv:2103.14050

- **"seitenzahl13"** [N1, N3, N5, N10, N20, N40, N100H, N100,] N100L, N150, N200, N300C, N1600, N1600C, N100_Z0.5, N100_Z0.1, N100_Z0.01
- "iwamoto99" : W7, W70, WDD1, WDD2, WDD3, CDD1, CDD2
- "gronow21" : M08_03_001, M08_03_01, M08_03_1, M08_03_3,
  M08_05_001, M08_05_01, M08_05_1, M08_05_3,
  M08_10_001, M08_10_01, M08_10_1, M08_10_3,
  M09_03_001, M09_03_01, M09_03_1, M09_03_3,
  M09_05_001, M09_05_01, M09_05_1, M09_05_3,
  M09_10_001, M09_10_01, M09_10_1, M09_10_3,
  M10_02_001, M10_02_01, M10_02_1, M10_02_3,
  M10_03_001, M10_03_01, M10_03_1, M10_03_3,
  M10_05_001, M10_05_01, M10_05_1, M10_05_3,
  M10_10_001, M10_10_01, M10_10_1, M10_10_3,
  M11_05_001, M11_05_01, M11_05_1, M11_05_3

n [real number [default][2.2e-03]] The average number of type Ia supernovae produced per unit stellar mass formed $N_{\text{Ia}}/M_\star$ in $M_\odot^{-1}$.

Note: The default value for this parameter is adopted from Maoz & Mannucci (2012)[5].

Returns

y [real number] The delay-time distribution integrated yield. This quantity represents the mass of some element produced over all SN Ia associated with a given stellar population in units of that stellar population's mass. This quantity is thus unitless ($M_\odot$ per $M_\odot$).

Note: Unlike vice.yields.ccsne.fractional, there is no associated numerical error with this function, because the solution is analytic.

Raises

- **ValueError**
  - The element is not built into VICE.
  - The study is not built into VICE.
  - n < 0
- **LookupError**
  - The model is not recognized for the given study.
- **IOError [Occurs only if VICE's file structure has been modified]**
  - The parameters passed to this function are allowed but the data file is not found.

[5] Maoz & Mannucci (2012), PASA, 29, 447

Notes

This function evaluates the solution to the following equation:

$$y_x^{\text{Ia}} = \left(\frac{N_{\text{Ia}}}{M_\star}\right) M_x$$

where M_x is the value returned by vice.yields.sneia.single, and $N_{\text{Ia}}/M_\star$ is specified by the parameter `n`.

The data stored in this module are reported for each corresponding study *as published*. The Seitenzahl et al. (2013) and Gronow et al. (2021a, b) models reported mass yields after complete decay of all radioactive nuclides with half-lives less than 2 Gyr, and the Iwamoto et al. (1999) study fully decayed *all* unstable isotopes; any additional treatment for radioactive isotopes is thus unnecessary.

The Gronow et al. (2021a, b) models are named for the mass of the carbon-oxygen core, the mass of the helium shell, and the metallicity of the progenitor relative to solar, in that order. For example, the "M09_05_01" model refers to one with a 0.9 $M_\odot$ carbon-oxygen core and a 0.05 $M_\odot$ helium shell produced by a star that was initially at a metallicity of 0.1 $Z_\odot$.

Example Code

```
>>> import vice
>>> vice.yields.sneia.fractional("fe")
0.0025825957080000002
>>> vice.yields.sneia.fractional("fe", study = "iwamoto99", model = "W70")
0.001705352
>>> vice.yields.sneia.fractional("fe", study = "iwamoto99", model = "CDD1")
0.0014255185999999997
>>> vice.yields.sneia.fractional("ni", model = "n1001")
8.610998000011574e-05
>>> vice.yields.sneia.fractional("ni", model = "N150")
0.00016497607368
>>> vice.yields.sneia.fractional("co", study = "gronow21",
        model = "M10_10_1")
2.3276e-06
>>> vice.yields.sneia.fractional("co", study = "gronow21",
        model = "M09_05_001")
3.4584000000000003e-07
```

See also:

- vice.yields.sneia.single
- vice.yields.sneia.gronow21
- vice.yields.sneia.iwamoto99
- vice.yields.sneia.seitenzahl13

vice.yields.sneia.settings

The VICE dataframe: global settings for SN Ia yields

For each chemical element, this object stores the current type Ia supernova (SN Ia) nucleosynthetic yield setting. See *Notes* below for mathematical details.

Note: Modifying yield settings through this dataframe is equivalent to going through the `vice.elements` module.

Indexing

- **`str` [case-insensitive]** [elemental symbols] This dataframe must be indexed by the symbol of an element recognized by VICE as it appears on the periodic table.

Item Assignment

For each chemical element, the SN Ia yield can be assigned either:

- real number : denotes a constant, metallicity-independent yield.
- **<function>** [Mathematical function describing the yield.] Must accept the metallicity by mass Z as the only parameter.

  Note: Functions of metallicity for yields of delayed enrichment channels like SNe Ia can significantly increase the required integration time in simulations, especially for fine timestepping.

  New in version 1.2.0: In earlier versions, VICE did not support SN Ia yields which vary with metallicity.

Functions

- keys
- todict
- restore_defaults
- factory_settings
- save_defaults

Notes

VICE implements the rate of enrichment from a single stellar population (SSP) according to a delay-time distribution $R_{\rm Ia}$, which here has units of Gyr^{-1}. For an SSP of age τ, the rate of production of some element due to SNe Ia is given by:

$$\dot{M}_{\rm Ia} = y_{\rm Ia} M_\star R_{\rm Ia}(\tau)$$

where $M_\star$ is the mass of the SSP and $y_{\rm Ia}$ is the yield of the element. As with all other yields in VICE, these are *net* rather than *gross* yields in that they quantify only the mass of a given element which is newly produced. In a one-zone

model, all stellar populations must be taken into account, which necessitates an integral over the star formation history:

$$\dot{M}_{\rm Ia} = \int_0^t \dot{M}_\star R_{\rm Ia}(\tau) dt$$

where $\dot{M}_\star$ is the star formation rate and t is the current time in the model. In a multi-zone model, the rate is a simple summation over all stellar populations in a given zone at some time:

$$\dot{M}_{\rm Ia} = \sum_i M_{\star,i} R_{\rm Ia}(\tau_i)$$

where the index i refers to the i'th SSP in a given zone. For futher details, see VICE's science documentation: https://vice-astro.readthedocs.io/en/latest/science_documentation/index.html.

Note: The normalization of $R_{\rm Ia}$ is irrelevant here, because VICE will always normalize it such that its integral from the minimum delay time up to 15 Gyr is equal to 1. Other sections of VICE's documentation refer to $R_{\rm Ia}$ as having units of $M_\odot^{-1}{\rm Gyr}^{-1}$ as published in the literature; here we note it as having units of ${\rm Gyr}^{-1}$ for simplicity.

Example Code

```
>>> import vice
>>> vice.yields.sneia.settings["fe"] = 0.001
>>> vice.yields.sneia.settings["Fe"]
        0.001
>>> vice.yields.sneia.settings["FE"] = 0.0012
>>> vice.yields.sneia.settings["fe"]
        0.0012
>>> def f(z):
        return 0.005 + 0.002 * (z / 0.014)
>>> vice.yields.sneia.settings["Fe"] = f
>>> vice.yields.sneia.settings["fe"]
        <function __main__.f(z)>
```

vice.yields.sneia.settings.keys

Returns the keys of the SN Ia yield settings dataframe.

Signature: vice.yields.sneia.settings.keys()

Note: By nature, this function will simply return a list of all elements that are built into VICE - the same thing as `vice.elements.recognized`.

Example Code

```
>>> import vice
>>> elements = vice.yields.sneia.settings.keys()
>>> tuple(elements) == vice.elements.recognized
        True
```

vice.yields.sneia.settings.todict

Returns the SN Ia yield settings dataframe as a dictionary.

Signature: vice.yields.sneia.settings.todict()

Note: Modifications to the dictionary returned by this function will *not* affect the global yield settings.

Note: Python dictionaries are case-insensitive, and are thus less flexible than this class.

Example

```
>>> import vice
>>> example = vice.yields.sneia.settings.todict()
>>> example["c"]
        5.74e-06
>>> example["C"]
        Traceback (most recent call last):
          File "<stdin>", line 1, in <module>
        KeyError: 'C'
>>> example["c"] = "not a yield setting"
>>> example["c"]
        "not a yield setting"
>>> vice.yields.sneia.settings["c"]
        5.74e-06
```

vice.yields.sneia.settings.restore_defaults

Restores the SN Ia yield settings to their default values (i.e. undoes any changes since VICE was imported).

Signature: vice.yields.sneia.settings.restore_defaults()

Example Code

```
>>> import vice
>>> vice.yields.sneia.settings["c"] = 0.0
>>> vice.yields.sneia.settings["n"] = 0.0
>>> vice.yields.sneia.settings["o"] = 0.0
>>> vice.yields.sneia.settings.restore_defaults()
>>> vice.yields.sneia.settings["c"]
        5.74e-06
>>> vice.yields.sneia.settings["n"]
        6.43e-09
>>> vice.yields.sneia.settings["o"]
        5.79e-05
```

vice.yields.sneia.settings.factory_settings

Restores the SN Ia yield settings to their factory defaults. This differs from `vice.yields.sneia.settings.restore_defaults` in that users may modify their default values from those that VICE is distributed with.

Signature: vice.yields.sneia.settings.factory_settings()

Tip: To revert your nucleosynthetic yield settings back to their production defaults *permanently*, simply call `vice.yields.sneia.settings.save_defaults` immediately following this function.

Example Code

```
>>> import vice
>>> vice.yields.sneia.settings["c"]
        0.0 # the user has modified their default yield for carbon
>>> vice.yields.sneia.settings.factory_settings()
>>> vice.yields.sneia.settings["c"]
        5.74e-06
```

vice.yields.sneia.settings.save_defaults

Saves the current SN Ia yield settings as the default values.

Signature: vice.yields.sneia.save_defaults()

Note: Saving functional yields requires the package dill, an extension to `pickle` in the python standard library. It is recommended that VICE users install dill >= 0.2.0.

Example Code

```
>>> import vice
>>> vice.yields.sneia.settings["c"]
        5.74e-06
>>> vice.yields.sneia.settings["c"] = 0.0
>>> vice.yields.sneia.settings.save_defaults()
```

After re-launching the python interpreter

```
>>> import vice
>>> vice.yields.sneia.settings["c"]
        0.0
```

vice.yields.sneia.iwamoto99

Iwamoto et al. (1999), ApJ, 124, 439 Type Ia supernova (SN Ia) yields

Signature: from vice.yields.sneia import iwamoto99

Importing this module will automatically set the SN Ia yield settings for all elements to the delay-time distribution integrated yields calculated with the Iwamoto et al. (1999) yield table under the W70 explosion model. This study reports yields for Chandrasekhar Mass progenitors (1.4 $M_\odot$) with a variety of deflagration speeds and ignition densities.

We provide type Ia supernova yields from the Iwamoto et al. (1999) study under the following explosion models presented in their journal publication:

- W7
- W70
- WDD1
- WDD2
- WDD3
- CDD1
- CDD2

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.sneia.iwamoto99.set_params

Update the parameters with which the yields are calculated from the Iwamoto et al. (1999)[1] data.

Signature: vice.yields.sneia.iwamoto99.set_params(**kwargs)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.sneia.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "iwamoto99" when called from this module.

Other exceptions are raised by vice.yields.sneia.fractional.

Notes

We provide type Ia supernova yields from the Iwamoto et al. (1999) study under the following explosion models presented in their journal publication:

- W7
- W70
- WDD1
- WDD2
- WDD3
- CDD1
- CDD2

Example Code

```
>>> import vice
>>> from vice.yields.sneia import iwamoto99
>>> vice.yields.sneia.settings['fe']
0.001705352
>>> iwamoto99.set_params(n = 1.5e-3)
>>> vice.yields.sneia.settings['fe']
0.00116274
```

(continues on next page)

[1] Iwamoto et al. (1999), ApJ, 124, 439

(continued from previous page)

```
>>> iwamoto99.set_params(n = 1.8e-3, model = "CDD1")
>>> vice.yields.sneia.settings['fe']
0.0011663333999999998
>>> iwamoto99.set_params(model = "cdd2")
>>> vice.yields.sneia.settings['fe']
0.001835812
```

See also:

- vice.yields.sneia.fractional
- vice.yields.sneia.single

vice.yields.sneia.seitenzahl13

Seitenzahl et al. (2013), MNRAS, 429, 1156 Type Ia supernova (SN Ia) yields

Signature: from vice.yields.sneia import seitenzahl13

Importing this module will automatically set the SN Ia yield settings for all elements to the delay-time distribution integrated yields calculated with the Seitenzahl et al. (2013) yield table under the N1 explosion model. This study reported yields for delayed detonation explosion models of Chandrasekhar mass progenitors (1.4 $M_\odot$).

We provide type Ia supernova yields from the Seitenzahl et al. (2013) study under the following explosions models presented in their journal publication:

- N1
- N3
- N5
- N10
- N20
- N40
- N100H
- N100
- N100L
- N150
- N200
- N300C
- N1600
- N1600C
- N100_Z0.5
- N100_Z0.1
- N100_Z0.01

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.sneia.seitenzahl13.set_params

Update the parameters with which the yields are calculated from the Seitenzahl et al. (2013)[1] data.

Signature: vice.yields.sneia.seitenzahl13.set_params(**kwargs)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.sneia.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "seitenzahl13" when called from this module.

Other exceptions are raised by vice.yields.sneia.fractional.

Notes

We provide type Ia supernova yields from the Seitenzahl et al. (2013) study under the following explosions models presented in their journal publication:

- N1
- N3
- N5
- N10
- N20
- N40
- N100H
- N100
- N100L
- N150
- N200
- N300C

[1] Seitenzahl et al. (2013), ApJ, 124, 439

- N1600
- N1600C
- N100_Z0.5
- N100_Z0.1
- N100_Z0.01

Example Code

```
>>> import vice
>>> from vice.yields.sneia import seitenzahl13
>>> vice.yields.sneia.settings['fe']
0.0025825957080000002
>>> seitenzahl13.set_params(n = 1.5e-3)
>>> vice.yields.sneia.settings['fe']
0.0017608607100000001
>>> seitenzahl13.set_params(n = 1.8e-3, model = "N100L")
>>> vice.yields.sneia.settings['fe']
0.0010877402286
>>> seitenzahl13.set_params(model = "N1600")
>>> vice.yields.sneia.settings['fe']
0.001158315444
```

See also:

- vice.yields.sneia.fractional
- vice.yields.sneia.single

vice.yields.sneia.gronow21

Gronow et al. (2021a, b) Type Ia supernova (SN Ia) yields

Signature: from vice.yields.sneia import gronow21

New in version 1.3.0.

Importing this module will automatically set the SN Ia yield settings for all elements to the delay-time distribution integrated yields calculated with the Gronow et al. (2021a, b)[1][2] yield tables under the M10_10_1 progenitor model. These studies reports yields for double detonations of sub-Chandrasekhar mass (1.4 $M_\odot$) white dwarfs at various progenitor metallicities, with the solar metallicity yields published in Gronow et al. (2021a) and those for remaining metallicities ($Z/Z_\odot$ = 0.01, 0.1, and 3) in Gronow et al. (2021b).

We provide type Ia supernova yields from the Gronow et al. (2021a, b) studies under the following explosion models presented in their journal publications:

- M08_03_001, M08_03_01, M08_03_1, M08_03_3
- M08_05_001, M08_05_01, M08_05_1, M08_05_3
- M08_10_001, M08_10_01, M08_10_1, M08_10_3
- M09_03_001, M09_03_01, M09_03_1, M09_03_3

[1] Gronow et al. (2021a), A&A, 649, 155
[2] Gronow et al. (2021b), arxiv:2103.14050

- M09_05_001, M09_05_01, M09_05_1, M09_05_3
- M09_10_001, M09_10_01, M09_10_1, M09_10_3
- M10_02_001, M10_02_01, M10_02_1, M10_02_3
- M10_03_001, M10_03_01, M10_03_1, M10_03_3
- M10_05_001, M10_05_01, M10_05_1, M10_05_3
- M10_10_001, M10_10_01, M10_10_1, M10_10_3
- M11_05_001, M11_05_01, M11_05_1, M11_05_3

These models are named for the mass of the carbon-oxygen core, the mass of the helium shell, and the metallicity of the progenitor relative to solar, in that order. For example, the "M09_05_01" model refers to one with a 0.9 $M_\odot$ carbon-oxygen core and a 0.05 $M_\odot$ helium shell produced by a star that was initially at a metallicity of 0.1 $Z_\odot$.

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: This module is not imported with a simple `import vice` statement.

Contents

set_params [<function>] Update the parameters with which the yields are calculated.

vice.yields.sneia.gronow21.set_params

Update the parameters with which the yields are calculated from the Gronow et al. (2021a, b)[1][2] data.

Signature: vice.yields.sneia.gronow21.set_params(**kwargs)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.sneia.fractional.

Raises

- **TypeError**
  - Received a keyword argument "study". This will always be "gronow21" when called from this module.

Other exceptions are raised by vice.yields.sneia.fractional.

See also:

vice.yields.sneia.fractional

[1] Gronow et al. (2021a), A&A, 649, 155
[2] Gronow et al. (2021b), arxiv:2103.14050

Notes

We provide type Ia supernova yields from the Gronow et al. (2021a, b) studies under the following explosion models presented in their journal publications:

- M08_03_001, M08_03_01, M08_03_1, M08_03_3
- M08_05_001, M08_05_01, M08_05_1, M08_05_3
- M08_10_001, M08_10_01, M08_10_1, M08_10_3
- M09_03_001, M09_03_01, M09_03_1, M09_03_3
- M09_05_001, M09_05_01, M09_05_1, M09_05_3
- M09_10_001, M09_10_01, M09_10_1, M09_10_3
- M10_02_001, M10_02_01, M10_02_1, M10_02_3
- M10_03_001, M10_03_01, M10_03_1, M10_03_3
- M10_05_001, M10_05_01, M10_05_1, M10_05_3
- M10_10_001, M10_10_01, M10_10_1, M10_10_3
- M11_05_001, M11_05_01, M11_05_1, M11_05_3

These models are named for the mass of the carbon-oxygen core, the mass of the helium shell, and the metallicity of the progenitor relative to solar, in that order. For example, the "M09_05_01" model refers to one with a 0.9 $M_\odot$ carbon-oxygen core and a 0.05 $M_\odot$ helium shell produced by a star that was initially at a metallicity of 0.1 $Z_\odot$.

Example Code

```
>>> import vice
>>> from vice.yields.sneia import gronow21
>>> vice.yields.sneia.settings['fe']
0.0017619157670400003
>>> gronow21.set_params(n = 1.5e-3)
>>> vice.yields.sneia.settings['fe']
0.0012013062048000002
>>> gronow21.set_params(n = 1.8e-3, model = "M09_10_001")
>>> vice.yields.sneia.settings['fe']
0.0009757080218934002
>>> gronow21.set_params(model = "M11_05_01")
>>> vice.yields.sneia.settings['fe']
0.0019527508063624003
```

See also:

- vice.yields.sneia.fractional
- vice.yields.sneia.single

vice.yields.presets

Nucleosynthetic Yield Presets

New in version 1.1.0.

Save copies of user-constructed yield settings for loading into VICE. Users can create external yield scripts which modify VICE's nucleosynthetic yield settings, then make these settings available to import statements.

Note: These features may not function properly if VICE is installed locally (i.e. if it was installed with a `--user` flag). Please speak with your administrator about installing VICE globally if this is an issue.

Contents

save [<function>] Save a copy of the yield settings declared in external python code. This will make the yield settings available to import statements for future simulations.

remove [<function>] Remove a copy of yield presets previously saved.

JW20 [yield preset] The yield presets associated with Johnson & Weinberg (2020)[1].

vice.yields.presets.save

Save a permanent copy of yields stored in a given file for loading back into VICE at any time via an import statement.

Signature: vice.yields.presets.save(filename)

New in version 1.1.0.

Parameters

filename [`str`] The full or relative path to the script containing the yields to be saved. The name of this file will become the name of the preset to use in import statements.

Raises

- **RuntimeError**
  - An exception occurs in attempting to import the file
  - The file is named JW20.py. This will always be the Johnson & Weinberg (2020) preset file.
- **IOError**
  - The file does not exist
- **TypeError**
  - filename is not of type str

[1] Johnson & Weinberg (2020), MNRAS, 498, 1364

Example Code

The following in a file named "example.py":

```
import vice
vice.yields.ccsne.settings['o'] = 0.015
vice.yields.ccsne.settings['fe'] = 0.0012
vice.yields.sneia.settings['o'] = 0.0
vice.yields.sneia.settings['fe'] = 0.0017
```

And the following in the same directory as that file:

```
>>> import vice
>>> vice.yields.presets.save("example.py")
```

This will enable the following from any directory:

```
>>> import vice
>>> from vice.yields.presets import example
>>> vice.yields.ccsne.settings['o']
0.015
```

vice.yields.presets.remove

Delete a copy of yield presets previously saved by a call to vice.yields.presets.save.

Signature vice.yields.presets.remove(name, force = False)

New in version 1.1.0.

Parameters

name [`str`] The name of the preset.

force [`bool` [default][`False`]] If `True`, will not stop for user confirmation before removing the yield file once it's found.

Raises

- **RuntimeError**
  - The preset module is not found
  - Another exception occurs in attempting to remove the yield file.
- **IOError**
  - The file does not exist

Example Code

```
>>> import vice
>>> vice.yields.presets.remove("example")
>>> from vice.yields.presets import example
Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
ImportError: cannot import name 'example' from 'vice.yields.presets'
(/anaconda3/lib/python3.7/site-packages/vice/yields/presets/__init__.py)
```

See also:

vice.yields.presets.save

vice.yields.presets.JW20

Johnson & Weinberg (2020), MNRAS, 498, 1364 Nucleosynthetic Yield Settings

Signature: from vice.yields.presets import JW20

New in version 1.1.0.

Importing this module sets the yields of oxygen, iron, and strontium to that adopted in the Johnson & Weinberg (2020) paper on starburst scenarios.

Note: This module is not imported with a simple "import vice" statement.

CCSNe

- $y_{\mathrm{O}}^{\mathrm{CC}} = 0.015$
- $y_{\mathrm{Fe}}^{\mathrm{CC}} = 0.0012$
- $y_{\mathrm{Sr}}^{\mathrm{CC}} = 3.5 \times 10^{-8}$

SNe Ia

- $y_{\mathrm{O}}^{\mathrm{Ia}} = 0$
- $y_{\mathrm{Fe}}^{\mathrm{Ia}} = 0.0017$
- $y_{\mathrm{Sr}}^{\mathrm{Ia}} = 0$

AGB

All three elements described by the Cristallo et al. (2011)[1] yields.

Other Contents

alt_cc_sr_linear [<function>] The functional form of the alternative CCSN Sr yield which is linear in metallicity Z.

alt_cc_sr_limitexp [<function>] The functional form of the alternative CCSN Sr yield which is a limited exponential in metallicity Z.

vice.yields.presets.JW20.alt_cc_sr_linear

The functional form of the alternative CCSN Sr yield explored in Johnson & Weinberg (2020)[1] which is linear in metallicity Z.

Signature: vice.yields.presets.JW20.alt_cc_sr_linear(Z, Z_solar = 0.014)

New in version 1.1.0.

Parameters

Z [real number] The metallicity by mass $M_Z/M_\star$.

Z_solar [real number [default][0.014]] The metallicity by mass of the Sun. Default value is take from Asplund et al. (2009)[2].

Returns

y [real number] The IMF-averaged CCSN Sr yield as a function of metallicity Z.

Notes

The yield is defined by:

$$y_{\text{Sr}}^{\text{CC}} = 3.5 \times 10^{-8} \left(\frac{Z}{Z_\odot}\right)$$

[1] Cristallo et al. (2011), ApJS, 197, 17
[1] Johnson & Weinberg (2020), MNRAS, 498, 1364
[2] Asplund et al. (2009), ARA&A, 47, 481

Example Code

```
>>> import vice
>>> from vice.yields.presets import JW20
>>> vice.yields.ccsne.settings['sr'] = JW20.alt_cc_sr_linear
>>> modified = lambda z: JW20.alt_cc_sr_linear(z, Z_solar = 0.018)
>>> vice.yields.ccsne.settings['sr'] = modified
```

vice.yields.presets.JW20.alt_cc_sr_limitexp

The functional form of the alternative CCSN Sr yield explored in Johnson & Weinberg (2020)[1] which is a limited exponential in Z.

Signature: vice.yields.presets.JW20.alt_cc_sr_limitexp(Z, Z_solar = 0.014)

New in version 1.1.0.

Parameters

Z [real number] The metallicity by mass $M_Z/M_\star$.

Z_solar [real number [default][0.014]] The metallicity by mass of the Sun. Default value is take from Asplund et al. (2009)[2].

Returns

y [real number] The IMF-averaged CCSN Sr yield as a function of metallicity Z.

Notes

The yield is defined by:

$$y_{\rm Sr}^{\rm CC} = 10^{-7}\left[1 - e^{-10(Z/Z_\odot)}\right]$$

Example Code

```
>>> import vice
>>> from vice.yields.presets import JW20
>>> vice.yields.ccsne.settings['sr'] = JW20.alt_cc_sr_limitexp
>>> modified = lambda z: JW20.alt_cc_sr_limitexp(z, Z_solar = 0.018)
>>> vice.yields.ccsne.settings['sr'] = modified
```

[1] Johnson & Weinberg (2020), MNRAS, 498, 1364
[2] Asplund et al. (2009), ARA&A, 47, 481

vice.elements

Chemical Elements

Provides a means of accessing nucleosynthetic yield information on an element-by-element basis.

New in version 1.1.0.

Contents

recognized [tuple of strings] The symbols of all elements that VICE recognizes as they appear on the periodic table.

element [type] Provides a means of accessing and modifying relevant information for different elements as well nucleosynthetic yields.

yields [type] Provides a means of accessing and modifying nucleosynthetic yield settings.

Element objects can be created from their symbols, or accessed directly through VICE's namespace. For example:

```
>>> import vice
>>> vice.elements.Fe
        vice.element{
                symbol ------------ > Fe
                name -------------- > Iron
                atomic number ----- > 26
                primordial -------- > 0
                solar abundance --- > 0.00129
                sources ----------- > ['CCSNE', 'SNEIA']
                stable isotopes --- > [54, 56, 57, 58]
                yields.ccsne ------ > 0.000246
                yields.sneia ------ > 0.00258
                yields.agb -------- > cristallo11
        }
>>> example = vice.elements.element("sr")
>>> example
        vice.element{
                symbol ------------ > Sr
                name -------------- > Strontium
                atomic number ----- > 38
                primordial -------- > 0
                solar abundance --- > 4.74e-08
                sources ----------- > ['CCSNE', 'AGB']
                stable isotopes --- > [84, 86, 87, 88]
                yields.ccsne ------ > 1.34e-08
                yields.sneia ------ > 0
                yields.agb -------- > cristallo11
        }
>>> example.symbol = 'fe'
>>> example
        vice.element{
                symbol ------------ > Fe
                name -------------- > Iron
                atomic number ----- > 26
                primordial -------- > 0
                solar abundance --- > 0.00129
```

(continues on next page)

(continued from previous page)

```
                sources ----------- > ['CCSNE', 'SNEIA']
                stable isotopes --- > [54, 56, 57, 58]
                yields.ccsne ------ > 0.000246
                yields.sneia ------ > 0.00258
                yields.agb -------- > cristallo11
        }
```

See also:

- vice.yields
- vice.atomic_number
- vice.primordial
- vice.solar_z
- vice.sources
- vice.stable_isotopes

vice.elements.recognized

Type : `tuple` [elements of type `str`]

The [lower-cased] symbols of all elements recognized by VICE as they appear on the periodic table.

New in version 1.1.0.

Example Code

```
>>> import vice
>>> "fe" in vice.elements.recognized
True
>>> "mg" in vice.elements.recognized
True
>>> "li" in vice.elements.recognized # VICE cannot do lithium yet
False
>>> "sr" in vice.elements.recognized
True
>>> "foo" in vice.elements.recognized
False
```

vice.elements.element

An object describing an element on the periodic table and its astrophysical nucleosynthetic sources and their associated yields.

Signature: vice.elements.element(symbol)

New in version 1.1.0.

Parameters

symbol [str [case-insensitive]] The symbol of the element as it appears on the periodic table.

Attributes

symbol [str] The symbol of the element as it appears on the periodic table.

name [str] The full name of the element in English.

yields [yields] The yields object containing the nucleosynthetic yield settings for this element.

atomic_number [int] The atomic number (protons only) of this element.

primordial [float] The primordial abundance by mass of this element according to the standard model[3][4][5].

solar_z [float] The abundance by mass of this element in the sun as determined by Asplund et al. (2009)[1].

sources [list of strings] The dominant astrophysical sources of this element as reported by Johnson (2019)[2].

stable_isotopes [list of integers] The mass numbers (protons and neutrons) of the stable isotopes of this element.

See also:

- vice.yields.agb.settings
- vice.yields.ccsne.settings
- vice.yields.sneia.settings
- vice.atomic_number
- vice.primordial
- vice.solar_z
- vice.sources
- vice.stable_isotopes

Example Code

```
>>> import vice
>>> vice.elements.Fe
        vice.element{
                symbol ------------ > Fe
                name --------------- > Iron
                atomic number ----- > 26
                primordial -------- > 0
                solar abundance --- > 0.00129
                sources ------------ > ['CCSNE', 'SNEIA']
                stable isotopes --- > [54, 56, 57, 58]
                yields.ccsne ------ > 0.000246
                yields.sneia ------ > 0.00258
```

(continues on next page)

[3] Planck Collaboration et al. (2016), A&A, 594, A13
[4] Pitrou et al. (2018), Phys. Rep., 754, 1
[5] Pattie et al. (2018), Science, 360, 627
[1] Asplund et al. (2009), ARA&A, 47, 481
[2] Johnson (2019), Science, 363, 474

(continued from previous page)

```
                yields.agb -------- > cristallo11
        }
>>> example = vice.elements.element("sr")
>>> example
        vice.element{
                symbol ------------ > Sr
                name -------------- > Strontium
                atomic number ----- > 38
                primordial -------- > 0
                solar abundance --- > 4.74e-08
                sources ----------- > ['CCSNE', 'AGB']
                stable isotopes --- > [84, 86, 87, 88]
                yields.ccsne ------ > 1.34e-08
                yields.sneia ------ > 0
                yields.agb -------- > cristallo11
        }
>>> example.symbol = 'fe'
>>> example
        vice.element{
                symbol ------------ > Fe
                name -------------- > Iron
                atomic number ----- > 26
                primordial -------- > 0
                solar abundance --- > 0.00129
                sources ----------- > ['CCSNE', 'SNEIA']
                stable isotopes --- > [54, 56, 57, 58]
                yields.ccsne ------ > 0.000246
                yields.sneia ------ > 0.00258
                yields.agb -------- > cristallo11
        }
```

vice.elements.element.symbol

Type : `str`

The one- or two-letter symbol of this element as it appears on the periodic table.

Example Code

```
>>> import vice
>>> vice.elements.Fe.symbol
        'Fe'
>>> example = vice.elements.element("sr")
>>> example.symbol = "Mg"
```

vice.elements.element.name

Type : str

The full name of the element in English.

Example Code

```
>>> import vice
>>> vice.elements.Mg.name
        'Magnesium'
>>> vice.elements.Sr.name
        'Strontium'
>>> vice.elements.Ne.name
        'Neon'
```

vice.elements.element.yields

The current yield settings from core collapse and type Ia supernovae and asymptotic giant branch stars. See ach attribute's docstring for more information.

Attributes

agb [str [case-insensitive] or <function>] The current setting for asymptotic giant branch stars.

ccsne [float or <function>] The current setting for core collapse supernovae.

sneia [float or <function>] The current setting for type Ia supernovae.

Example Code

```
>>> import vice
>>> vice.elements.Fe.yields.agb
        "cristallo11"
>>> vice.elements.Fe.yields.ccsne = 0.0012
>>> vice.yields.ccsne.settings['fe']
        0.0012
```

vice.elements.element.atomic_number

Type : int

The atomic number (protons only) of the element.

Example Code

```
>>> import vice
>>> vice.elements.Fe.atomic_number
        26
>>> vice.elements.Sr.atomic_number
        38
```

vice.elements.element.primordial

Type :: `float`

The abundance of this element by mass following big bang nucleosynthesis, according to the standard model[1][2][3]. This is zero for all elements with the exception of helium, for which it is 0.24672.

Example Code

```
>>> import vice
>>> vice.elements.Fe.primordial
        0
>>> vice.elements.He.primordial
        0.24672
```

vice.elements.element.solar_z

Type : `float`

The abundance by mass of this element in the sun as reported by Asplund et al. (2009)[1].

New in version 1.2.0: As of version 1.2.0, users can modify the assumed solar chemical composition.

Example Code

```
>>> import vice
>>> vice.elements.Fe.solar_z
        0.00129
>>> vice.elements.O.solar_z
        0.00572
```

[1] Planck Collaboration et al. (2016), A&A, 594, A13
[2] Pitrou et al. (2018), Phys. Rep., 754, 1
[3] Pattie et al. (2018), Science, 360, 627
[1] Asplund et al. (2009), ARA&A, 47, 481

vice.elements.element.sources

Type : `list` of strings

Strings denoting the dominant sources of enrichment for this element as reported by Johnson (2019)[1].

Example Code

```
>>> import vice
>>> vice.elements.Fe.sources
        ['CCSNE', 'SNEIA']
>>> vice.elements.Mg.sources
        ['CCSNE']
```

Note: This parameter does not impact simulations in any way. It is purely a look-up function.

vice.elements.element.stable_isotopes

Type : `list` of integers

The mass numbers (protons and neutrons) of the stable isotopes of this element.

New in version 1.1.0.

Example Code

```
>>> import vice
>>> vice.elements.Fe.stable_isotopes
        [54, 56, 57, 58]
>>> vice.elements.Mg.stable_isotopes
        [24, 25, 26]
```

vice.elements.yields

Current Nucleosynthetic yield settings for a given element.

Signature: vice.elements.yields(symbol)

New in version 1.1.0.

[1] Johnson (2019), Science, 363, 474

Parameters

symbol [`str` [case-insensitive]] The symbol of an element as it appears on the periodic table.

Attributes

agb [`str` [case-insensitive] or <function>] The asymptotic giant branch star yield setting.

ccsne [`float` or <function>] The core collapse supernova yield setting.

sneia [`float` or <function>] The type Ia supernova yield setting.

Note: modifying yields here is equivalent to modifying them through the vice.yields module.

---

vice.elements.yields.agb

Type : `str` [case-insensitive] or <function>

New in version 1.2.0: Prior to version 1.2.0, individual functions and objects required an attribute or keyword argument `agb_model`. With version 1.2.0, this was changed to a global yield setting like the supernova yields.

The current yield setting for asymptotic giant branch stars. If this is a string, it will be interpreted as a keyword denoting the built-in table from a nucleosynthesis study to adopt. If this is a <function>, it must accept stellar mass in $M_\odot$ as the first parameter and the metallicity by mass Z as the second.

Keywords and their Associated Studies:

- "cristallo11" : Cristallo et al. (2011)[1]
- "karakas10" : Karakas (2010)[2]
- **"ventura13"** [Ventura et al. (2013, 2014, 2018, 2020)[3]] [4][5][6]
- **"karakas16"** [Karakas & Lugaro (2016)[7], Karakas et al. (2018) [8]

New in version 1.3.0: The "ventura13" and "karakas16" yield models were introduced in version 1.3.0.

Internal yield tables can be analyzed by calling vice.yields.agb.grid.

Note: Modifying yield settings here is equivalent to modifying vice.yields.agb.settings.

See also:

vice.yields.agb.settings vice.yields.agb.grid

[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Karakas (2010), MNRAS, 403, 1413
[3] Ventura et al. (2013), MNRAS, 431, 3642
[4] Ventura et al. (2014), MNRAS, 437, 3274
[5] Ventura et al. (2018), MNRAS, 475, 2282
[6] Ventura et al. (2020), A&A, 641, A103
[7] Karakas & Lugaro (2016), ApJ, 825, 26
[8] Karakas et al. (2018), MNRAS, 477, 421

Example Code

```
>>> import vice
>>> vice.elements.C.yields.agb = "cristallo11"
>>> vice.elements.N.yields.agb = "cristallo11"
>>> vice.elements.Ne.yields.agb = "ventura13"
>>> vice.elements.Mg.yields.agb = "karakas16"
>>> vice.elements.O.yields.agb = "karakas10"
>>> def my_n_yield(m, z):
        return 9.0e-4 * m * (z / 0.014)
>>> vice.elements.N.yields.agb = my_n_yield
```

vice.elements.yields.ccsne

Type : real number or <function>

The current yield setting for core collapse supernovae (CCSNe). If this is a real number, it will be interpreted as a constant, metallicity-independent yield. If it is a function, it must accept the metallicity by mass Z as the only parameter.

These values can be calculated by calling vice.yields.ccsne.fractional.

Note: Modifying yield settings here is equivalent to modifying vice.yields.ccsne.settings.

See also:

- vice.yields.ccsne.settings
- vice.yields.ccsne.fractional
- vice.yields.ccsne.table

Example Code

```
>>> import vice
>>> vice.elements.Fe.yields.ccsne = 0.0012
>>> vice.elements.O.yields.ccsne = 0.015
>>> def z_dep_o_yield(z):
        return 0.015 * (z / 0.014)**0.5
>>> vice.elements.O.yields.ccsne = z_dep_o_yield
```

vice.elements.yields.sneia

Type : real number or <function>

The current yield setting for type Ia supernovae (SNe Ia). If this is a real number, it will be interpreted as a constant, metallicity-independent yield. If it is a function, it must accept the metallicity by mass Z as the only parameter.

These values can be calculated by calling vice.yields.sneia.fractional.

Note: Modifying yield settings here is equivalent to modifying vice.yields.sneia.settings.

See also:

- vice.yields.sneia.settings
- vice.yields.sneia.fractional
- vice.yields.sneia.single

Example Code

```
>>> import vice
>>> vice.elements.Fe.yields.sneia = 0.0017
>>> vice.elements.O.yields.sneia = 0
>>> def z_dep_fe_yield(z):
        return 0.0017 * (z / 0.014)**0.5
>>> vice.elements.Fe.yields.sneia = z_dep_fe_yield
```

vice.imf

Built-in functional forms of popular stellar initial mass functions (IMFs).

New in version 1.1.0.

Contains

Kroupa [<function>] The Kroupa (2001) IMF[1].

Salpeter [<function>] The Salpeter (1955) IMF[2].

vice.imf.kroupa

The (unnormalized) Kroupa (2001)[1] stellar initial mass function (IMF).

Signature: vice.imf.kroupa(mass)

New in version 1.1.0.

[1] Kroupa (2001), MNRAS, 322, 231
[2] Salpeter (1955), ApJ, 121, 161
[1] Kroupa (2001), MNRAS, 322, 231

Parameters

mass [real number] The stellar mass in solar masses.

Returns

dndm [real number] The unnormalized value of the Kroupa IMF at that stellar mass, defined by:

$$\frac{dN}{dm} \propto m^{-\alpha}$$

where α = 2.3, 1.3, and 0.3 for m > 0.5, 0.08 $\leq m \leq$ 0.5, and m < 0.08, respectively.

Raises

- **TypeError**
  - mass is not a real number
- **ValueError**
  - mass is non-positive

Example Code

```
>>> vice.imf.kroupa(1)
        0.04
>>> vice.imf.kroupa(0.5)
        0.1969831061351866
>>> vice.imf.kroupa(2)
        0.008122523963562356
```

vice.imf.salpeter

The (unnormalized) Salpeter (1955)[1] stellar initial mass function (IMF).

Signature: vice.imf.salpeter(mass)

New in version 1.1.0.

Parameters

mass [real number] The stellar mass in solar masses.

[1] Salpeter (1955), ApJ, 121, 161

Returns

dndm [real number] The unnormalized value of the Salpeter IMF at that stellar mass, defined by:

$$\frac{dN}{dm} \propto m^{-\alpha}$$

where α = 2.35 always.

Raises

- **TypeError**
  - mass is not a real number
- **ValueError**
  - mass is non-positive

Example Code

```
>>> vice.imf.salpeter(1)
        1.0
>>> vice.imf.salpeter(0.5)
        5.098242509277049
>>> vice.imf.salpeter(2)
        0.19614602447418766
```

vice.singlezone

An object designed to run simulations of chemical enrichment under the single-zone approximation for user-specified parameters. The parameters of the simulation are implemented as attributes of this class.

Signature: vice.singlezone(**kwargs)

Parameters

kwargs [varying types] Every attribute of this class can be assigned via a keyword argument.

Attributes

name [`str` [default]["onezonemodel"]] The name of the simulation. Output will be stored in a directory under this name.

func [<`function`> [default][vice._globals._DEFAULT_FUNC_]] A function of time describing some evolutionary parameter. Physical interpretation set by the attribute `mode`.

mode [`str` [default]["ifr"]] The interpretation of the attribute `func`. Either "ifr" for infall rate, "sfr" for star formation rate, or "gas" for the mass of gas.

verbose [`bool` [default][False]] Whether or not to print to the console as the simulation runs.

New in version 1.1.0.

elements [tuple [default][("fe", "sr", "o")]] A tuple of strings holding the symbols of the elements to be simulated.

IMF [str [case-insensitive] or <function> [default]["kroupa"]] The stellar initial mass function (IMF) to adopt. Either a string denoting a built-in IMF or a function containing a user-constructed IMF.

Recognized built-in IMFs:

- "kroupa"[1]
- "salpeter"[2]

New in version 1.2.0: Prior to version 1.2.0, only the built-in Kroupa and Salpeter IMFs were supported.

eta [real number [default][2.5]] The mass-loading parameter: the ratio of outflow to star formation rates. This changes when the attribute smoothing is nonzero.

enhancement [real number or <function> [default][1]] The ratio of outflow to ISM metallicities. Numbers are interpreted as constants. Functions must accept time in Gyr as a parameter.

Zin [real number, <function>, or dataframe [default][0]] The infall metallicity, which can be a constant, time-vary, or have element-by-element specifications.

recycling [str [case-insensitive] or real number] [default : "continuous"] Either the string "continuous" or a real number between 0 and 1. Denotes the prescription for recycling of previously produced heavy nuclei.

bins [array-like [default][[-3.0, -2.95, -2.9, ... , 0.9, 0.95, 1.0]]] The binspace within which to sort the normalized stellar metallicity distribution function in each [X/H] and [X/Y] abundance ratio measurement.

delay [real number [default][0.15]] The minimum delay time in Gyr before the onset of type Ia supernovae associated with a single stellar population

RIa [str [case-insensitive] or <function> [default]["plaw"]] The SN Ia delay-time distribution (DTD) to adopt. Strings denote built-in DTDs and functions must accept time in Gyr as a parameter.

Mg0 [real number [default][6.0e+09]] The initial gas supply of the galaxy in solar masses. This is only relevant when the simulation is ran in infall mode (i.e. mode == "ifr").

smoothing [real number [default][0]] The outflow smoothing timescale in Gyr.[3]

tau_ia [real number [default][1.5]] The e-folding timescale of type Ia supernovae in gyr when the attribute RIa == "exp".

tau_star [real number or <function> [default][2.0]] The star formation rate per unit gas mass in the galaxy in Gyr. This can be either a number which will be treated as a constant, or a function of time in Gyr, whose behavior can be modified when the attribute schmidt == True. Can also be a function which accepts a second parameter in addition to time in Gyr; when mode == "ifr" or "gas", this will be interpreted as the gas mass in $M_\odot$, and when mode == "sfr", it will be interpreted as the star formation rate in $M_\odot/yr$.

New in version 1.2.0: Prior to version 1.2.0, functions could only accept time in Gyr as the new parameter.

dt [real number [default][0.01]] The timestep size in Gyr.

schmidt [bool [default][False]] A boolean describing whether or not to implement a gas-dependent star formation efficiency. Overridden when the attribute tau_star is a function of two variables.

schmidt_index [real number [default][0.5]] The power-law index of gas-dependent star formation efficiency. Overridden when the attribute tau_star is a function of two variables.

MgSchmidt [real umber [default][6.0e+09]] The normalization of the gas-supply when the attribute schmidt = True. Overridden when the attribute tau_star is a function of two variables.

m_upper [real number [default][100]] The upper mass limit on star formation in $M_\odot$.

[1] Kroupa (2001), MNRAS, 231, 322
[2] Salpeter (1955), ApJ, 121, 161
[3] Johnson & Weinberg (2020), MNRAS, 498, 1364

m_lower [real number [default][0.08]] The lower mass limit on star formation in $M_\odot$.

postMS [real number [default][0.1]] The lifetime ratio of the post main sequence to main sequence phases of stellar evolution.

New in version 1.1.0.

Z_solar [real number [default][0.014]] The adopted metallicity by mass of the sun.

agb_model [`str` [case-insensitive] [default][None]] **[DEPRECATED]**

A keyword denoting which table of nucleosynthetic yields from AGB stars to adopt.

Recognized Keywords:

- "cristallo11"[4]
- "karakas10"[5]

Deprecated since version 1.2.0: Users should instead modify their AGB star yield settings through `vice.yields.agb.settings`. Users may specify either a built-in study or a function of stellar mass and metallicity.

Functions

run [[instancemethod]] Run the simulation.

from_output [[classmethod]] Obtain a `singlezone` object with the parameters of the one that produced an output.

Notes

Implementation VICE uses a forward Euler approach to handle its timestepping. Although this isn't the highest numerical resolution timestepping method, the dominant source of error in VICE is not in the numerics but in the approximations built into the model itself. Solutions in which the numerical error is adequately small can be achieved with reasonable timestep sizes. Furthermore, the forward Euler approach allows VICE to treat the discretization of timesteps to correspond directly to a discretization of stellar populations, simplifying its implementation and allowing fast numerical solutions. The exact timestamps at which functions of time describing evolutionary parameters will be evaluated is also a simple calculation as a result, since they will all be integer multiples of the timestep size. For further details, see VICE's science documentation: https://vice-astro.readthedocs.io/en/latest/science_documentation/index.html

Computational Overhead In general, the `singlezone` object is not memory limited, requiring only ~700 MB of RAM and ~20 seconds to compute abundances for 3 elements across 10,000 timesteps with default parameters. With 1,000 timesteps, it takes only 500 MB of RAM and finishes in ~1/4 second. For quantified measurements of the `singlezone` object's required integration time, see "Timed Runs" under VICE's science documentation: https://vice-astro.readthedocs.io/en/latest/science_documentation/implementation.html#timed-runs

Relationship to ``vice.multizone`` The `multizone` object makes use of composition. At its core, it is an array of `singlezone` objects.

[4] Cristallo et al. (2011), ApJS, 197, 17
[5] Karakas (2010), MNRAS, 403, 1413

Example Code

```
>>> import vice
>>> sz = vice.singlezone()
>>> sz
        vice.singlezone{
                name -----------> onezonemodel
                func -----------> <function _DEFAULT_FUNC_ at 0x112180ae8>
                mode -----------> ifr
                verbose --------> False
                elements -------> ('fe', 'sr', 'o')
                IMF ------------> kroupa
                eta ------------> 2.5
                enhancement ----> 1.0
                entrainment ----> <entrainment settings>
                Zin ------------> 0.0
                recycling ------> continuous
                delay ----------> 0.15
                RIa ------------> plaw
                Mg0 ------------> 6000000000.0
                smoothing ------> 0.0
                tau_ia ---------> 1.5
                tau_star -------> 2.0
                schmidt --------> False
                schmidt_index --> 0.5
                MgSchmidt ------> 6000000000.0
                dt -------------> 0.01
                m_upper --------> 100.0
                m_lower --------> 0.08
                postMS ---------> 0.1
                Z_solar --------> 0.014
                bins -----------> [-3, -2.95, -2.9, ... , 0.9, 0.95, 1]
        }
```

vice.singlezone.run

Run the simulation.

Signature: x.run(output_times, capture = False, overwrite = False)

Parameters

x [`singlezone`] An instance of this class.

output_times [array-like [elements are real numbers]] The times in Gyr at which VICE should record output from the simulation. These need not be sorted from least to greatest.

capture [`bool` [default][False]] If `True`, an output object containing the results of the simulation will be returned.

overwrite [`bool` [default][False]] If `True`, will force overwrite any files with the same name as the simulation output files.

Returns

out [`output` [only returned if `capture == True`]] An `output` object produced from this simulation's output.

Raises

- **TypeError**
  - Any functional attribute evaluates to a non-numerical value.
- **ValueError**
  - Any element of output_times is negative.
  - An inflow metallicity evaluates to a negative value.
- **ArithmeticError**
  - Any functional attribute evaluates to NaN or inf.
- **UserWarning**
  - Any yield settings or class attributes are callable and the user does not have dill installed.
  - Output times are more finely spaced than the timestep size.
- **ScienceWarning**
  - Any element tracked by the simulation is enriched in signifcant part by r-process nucleosynthesis.
  - Any element tracked by the simulation has a weakly constrained solar abundance measurement.
- **VisibleRuntimeWarning**
  - The attribute `RIa` is a user-defined function.
  - Any of the elements tracked by the simulation have AGB star yields described by a user-defined function.
  - The model is running with a mass-lifetime relation which requires numerical solutions to the inverse function (i.e. mass as a function of lifetime).

Notes

Note: Calling this function only causes VICE to produce the output files. The `output` class handles the reading and storing of the simulation results.

Note: Saving functional attributes with VICE outputs requires the package dill, an extension to `pickle` in the python standard library. It is recommended that VICE users install dill >= 0.2.0.

Note: When `overwrite == False`, and there are files under the same name as the output produced, this acts as a halting function. VICE will wait for the user's approval to overwrite existing files in this case. If users are running multiple simulations and need their integrations not to stall, they must specify `overwrite = True`.

Note: VICE will always write output at the final timestep of the simulation. This may be one timestep beyond the last element of the specified `output_times` array.

Example Code

```
>>> import numpy as np
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> outtimes = np.linspace(0, 10, 1001)
>>> sz.run(outtimes)
```

vice.singlezone.from_output

Obtain an instance of the `singlezone` class given either the path to an output or an output itself.

Signature: vice.singlezone.from_output(arg)

New in version 1.1.0.

Parameters

arg [`str` or `output`] The full or relative path to the output directory; the '.vice' extension is not necessary. Alternatively, an output object.

Returns

sz [`singlezone`] A `singlezone` object with the same parameters as the one which produced the output.

Raises

- **TypeError**
  - arg is neither an output object nor a string
- **IOError [Only occurs if the output has been altered]**
  - The output is missing files

Notes

Note: If arg is either a `multizone` output or a `multioutput` object, a `multizone` object will be returned.

Note: In versions before 1.1.0, this function had the call signature `vice.mirror` (now deprecated).

Note: This function serving as the reader, the writer is the vice.core.singlezone._singlezone.c_singlezone.pickle function, implemented in Cython.

Example Code

```
>>> import numpy as np
>>> import vice
>>> vice.singlezone(name = "example").run(np.linspace(0, 10, 1001))
>>> sz = vice.singlezone.from_output("example")
>>> sz
        vice.singlezone{
                name -----------> example
                func -----------> <function _DEFAULT_FUNC_ at 0x10d0c8e18>
                mode -----------> ifr
                verbose --------> False
                elements -------> ('fe', 'sr', 'o')
                IMF ------------> kroupa
                eta ------------> 2.5
                enhancement ----> 1.0
                entrainment ----> <entrainment settings>
                Zin ------------> 0.0
                recycling ------> continuous
                delay ----------> 0.15
                RIa ------------> plaw
                Mg0 ------------> 6000000000.0
                smoothing ------> 0.0
                tau_ia ---------> 1.5
                tau_star -------> 2.0
                schmidt --------> False
                schmidt_index --> 0.5
                MgSchmidt ------> 6000000000.0
                dt -------------> 0.01
                m_upper --------> 100.0
                m_lower --------> 0.08
                postMS ---------> 0.1
                Z_solar --------> 0.014
                bins -----------> [-3, -2.95, -2.9, ... , 0.9, 0.95, 1]
        }
```

vice.singlezone.name

Type : `str`

Default : "onezonemodel"

The name of the simulation. The output will be stored in a directory under this name with the extension ".vice". This can also be of the form `./path/to/directory/name` and the output will be stored there.

Tip: Users need not interact with any of the output files. The `output` object is designed to read in all of the results

automatically.

Tip: By forcing a ".vice" extension on the output directory, users can run `<command> \*.vice` in a terminal to run commands over all VICE outputs in a given directory.

Note: The outputs of this class include the full time evolution of the interstellar abundances, the resulting stellar metallicity distribution, and pickled objects that allow a singlezone object to construct itself from the output. By separating the output into a handful of files, the full time evolution data and the resulting stellar metallicity distribution can be stored in pure ascii text files. This allows users to analyze their simulations in languages other than python with ease. Most of the relevant information is stored in the history.out and mdf.out files within the output directory.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.name = "another_name"
```

vice.singlezone.func

Type : `<function>`

Default : vice._globals._DEFAULT_FUNC_

A callable object which must accept time in Gyr as the only parameter. The value returned by this function will represent either the gas infall history in $M_\odot\ yr^{-1}$ (`mode` == "ifr"), the star formation history in $M_\odot\ yr^{-1}$ (`mode` == "sfr"), or the ISM gas supply in $M_\odot$ (`mode` == "gas).

Note: The default function returns the value of 9.1 always. With a default `mode` of "ifr", this corresponds to an infall rate of 9.1 $M_\odot\ yr^{-1}$ at all times.

Note: Saving this functional attribute with VICE outputs requires the package dill, an extension to `pickle` in the Python standard library. It is recommended that VICE user's install dill >= 0.2.0.

Note: This attribute will always be expected to accept time in Gyr as the only parameter. However, infall and star formation rates will be interpreted as having units of $M_\odot\ yr^{-1}$ according to convention.

See also:

vice.singlezone.mode

Example Code

```
>>> import math as m
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> def f(t):
        if t <= 1:
                return 10
        else:
                return 10 * m.exp(-(t - 1) / 3)
>>> sz.func = f
>>> sz.func = lambda t: 10. * m.exp(-t / 3)
```

vice.singlezone.mode

Type : `str` [case-insensitive]

Default : "ifr"

The interpretation of the attribute `func`.

- mode = "ifr" : The value returned from the attribute `func` represents the rate of gas infall into the interstellar medium in $M_\odot\ yr^{-1}$.
- mode = "sfr" : The value returned from the attribute `func` represents the star formation rate of the galaxy in $M_\odot\ yr^{-1}$.
- mode = "gas" : The value returned from the attribute `func` represents the mass of the ISM gas in $M_\odot$.

Note: The attribute `func` will always be expected to accept time in Gyr as the only parameter. However, infall and star formation rates will be interpreted as having units of $M_\odot\ yr^{-1}$ according to convention.

See also:

vice.singlezone.func

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.mode = "sfr"
>>> sz.mode = "gas"
```

vice.singlezone.verbose

Type : bool

Default : False

If True, the simulation will print to the console as it evolves.

New in version 1.1.0.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.verbose = True
```

vice.singlezone.elements

Type : tuple [elements of type str [case-insensitive]]

Default : ("fe", "sr", "o")

The symbols for the elements to track the enrichment for (case-insensitive). The more elements that are tracked, the longer the simulation will take, but the better calibrated is the total metallicity of the ISM in handling metallicity-dependent yields.

Tip: The order in which the elements appear in this tuple will dictate the abundance ratios that are quoted in the final stellar metallicity distribution function. That is, if element X appears before element Y, then VICE will determine the MDF in $dN/d[Y/X]$ as opposed to $dN/d[X/Y]$. The elements that users intend to use as "reference elements" should come earliest in this list.

Note: All versions of VICE support the simulation of all 76 astrophysically produced elements between carbon ("c") and bismuth ("bi"). Versions >= 1.1.0 also support helium ("he").

Note: Some of the heaviest elements that VICE recognizes have statistically significant enrichment from r-process nucleosynthesis[1]. Simulations of these elements with realistic parameters and realistic nucleosynthetic yields will underpredict the absolute abundances of these elements. However, if these nuclei are assumed to be produced promptly following the formation of a single stellar population, the yield can be added to the yield from core collapse supernovae, which in theory can describe the total yield from all prompt sources[2].

---

[1] Johnson (2019), Science, 363, 474
[2] Johnson & Weinberg (2020), MNRAS, 498, 1364

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.elements
("fe", "sr", "o")
>>> sz.elements = ["mg", "fe", "n", "c", "o"]
>>> sz.elements
("mg", "fe", "n", c", "o")
```

vice.singlezone.IMF

Type : `str` [case-insensitive] or <`function`>

Default : "kroupa"

New in version 1.2.0: In versions >= 1.2.0, users may construct a function of mass to describe the IMF.

The assumed stellar initial mass function (IMF). If assigned a string, VICE will adopt a built-in IMF. Functions must accept stellar mass as the only parameter and are expected to return the value of the IMF at that mass (it need not be normalized).

Built-in IMFs:

- "kroupa"[1]
- "salpeter"[2]

Note: VICE has analytic solutions to the *cumulative return fraction* and the *main sequence mass fraction* for built-in IMFs. If assigned a function, VICE will calculate these quantities numerically, increasing the required integration time.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.IMF = "kroupa"
>>> def f(m):
        if m < 0.5:
                return m**-1.2
        else:
                return m**-2.2
>>> sz.IMF = f
```

[1] Kroupa (2001), MNRAS, 322, 231
[2] Salpeter (1955), ApJ, 121, 161

vice.singlezone.eta

Type : real number or <function>

Default : 2.5

The mass loading factor, defined as the ratio of the mass outflow rate to the star formation rate.

$$\eta \equiv \frac{\dot{M}_{\text{out}}}{\dot{M}_\star}$$

Note: If the attribute `smoothing` is nonzero, this relationship generalizes to

$$\dot{M}_{\text{out}} = \eta(t)\langle\dot{M}_\star\rangle_{\tau_s} = \begin{cases} \frac{\eta(t)}{t}\int_0^t \dot{M}_\star(t')dt' (t < \tau_s) \\ \frac{\eta(t)}{\tau_s}\int_{t-\tau_s}^t \dot{M}_\star(t')dt' \ (t \geq \tau_s) \end{cases}$$

where τ_s is the value of the attribute, the outflow smoothing time.

Note also that the time-average is over the star formation rate only, and not the mass-loading factor.

Note: Saving this functional attribute with VICE outputs requires the package dill, an extension to `pickle` in the Python standard library. It is recommended that VICE user's install dill >= 0.2.0.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.eta = 2
>>> def f(t):
        if t <= 2:
                return 5
        else:
                return 1
>>> sz.eta = f
```

vice.singlezone.enhancement

Type : real number or <function>

Default : 1.0

The ratio of the outflow to ISM metallicities. Real numbers will be taken as constant. Functions must accept time in Gyr as the only parameter. This will apply to all elements tracked by the simulation.

Note: Saving this functional attribute with VICE outputs requires the package dill, an extension to `pickle` in the Python standard library. It is recommended that VICE user's install dill >= 0.2.0.

See also:

- vice.singlezone.eta

- vice.singlezone.smoothing

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.enhancement = 3
>>> def f(t):
        if t <= 1:
                return 5
        else:
                return 1
>>> sz.enhancement = f
```

vice.singlezone.entrainment

Type : <entrainment object>

Default : all elements from all enrichment channels assigned a value of 1.

Each element from each enrichment channel assigned a value of 1. These values denote the mass fraction of nucleosynthetic yields that are retained by the interstellar medium, the remainder of which is added directly to outflows. These must always be numerical values between 0 and 1.

Attributes

agb [dataframe] The entrainment fraction of each element from AGB stars

ccsne [dataframe] The entrainment fraction of each element from CCSNe

sneia [dataframe] The entrainment fraction of each element fron SNe Ia

See also:

vice.dataframe

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> # set the entrainment of CCSN ejecta to 80 percent
>>> for i in sz.elements:
        sz.entrainment.ccsne[i] = 0.8
>>> # set the entrainment of SN Ia ejecta to 90 percent
>>> for i in sz.elements:
        sz.entrainment.sneia[i] = 0.9
```

vice.singlezone.entrainment.agb

Type : dataframe

Default : All elements map to a value of 1.0

The entrainment fraction of each element from asymptotic giant branch stars.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.entrainment.agb['c'] = 0.9
>>> sz.entrainment.agb['n'] = 0.95
```

vice.singlezone.entrainment.ccsne

Type : dataframe

Default : All recognized elements map to a value of 1.0

The entrainment fraction of each element from core collapse supernovae.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.entrainment.ccsne['o'] = 0.8
>>> sz.entrainment.ccsne['mg'] = 0.85
```

vice.singlezone.entrainment.sneia

Type : dataframe

Default : All recognized elements map to a value of 1.0

The entrainment fraction of each element from type Ia supernovae.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.entrainment.sneia['fe'] = 0.7
>>> sz.entrainment.sneia['ni'] = 0.75
```

vice.singlezone.Zin

Type : real number, <function>, or dataframe

Default : 0.0

The metallicity of gas inflow. Numbers and functions apply to all elements tracked by the simulation. Functions must accept time in Gyr as the only parameter. A dictionary or a dataframe can also be passed, allowing real numbers and functions to be assigned on an element-by-element basis.

Tip: The easiest way to switch this attribute to a dataframe is by passing an empty python dictionary {}.

Note: Dictionaries will be automatically converted into a dataframe.

Note: Saving functional attributes with VICE outputs requires the package dill, an extension to pickle in the Python standard library. It is recommended that VICE user's install dill >= 0.2.0.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.Zin = 0.001
>>> def f(t):
        return 0.001 * (t / 5)
>>> sz.Zin = lambda t: 0.001 * (t / 5)
>>> sz.Zin = {}
>>> sz.Zin
vice.dataframe{
        sr --------------> 0.0
        fe --------------> 0.0
        o ---------------> 0.0
}
>>> sz.Zin["o"] = 0.001
>>> sz.Zin["fe"] = lambda t: 1.0e-04 * (t / 5)
>>> sz.Zin
vice.dataframe{
        sr --------------> 0.0
        fe --------------> <function main.<__lambda__>(t)>
        o ---------------> 0.001
}
```

vice.singlezone.recycling

Type : real number or `str` [case-insensitive]

Default : "continuous"

The *cumulative return fraction* $r(t)$. This is the mass fraction of a single stellar population returned to the interstellar medium as gas at the birth metallicity of the stars.

The only allowed string is "continuous" [case-insensitive]. In this case VICE will implement time-dependent recycling from each episode of star formation via a treatment of the stellar initial mass function and the initial-final remnant mass model of Kalirai at al. (2008)[1].

Numbers must be between 0 and 1 (inclusive), and will be interpreted as the instantaneous recycling fraction: the fraction of a stellar population's mass that is returned to the interstellar medium immediately following its formation.

Note: In the case of instantaneous recycling, it is recommened that users adopt r = 0.4 with the Kroupa[2] IMF and r = 0.2 with the Salpeter[3] IMF based on the findings of Weinberg, Andrews & Freudenburg (2017)[4].

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example", IMF = "kroupa")
>>> sz.recycling = 0.4
>>> sz.IMF = "salpeter"
>>> sz.recycling = 0.2
>>> sz.recycling = "continuous"
```

vice.singlezone.bins

Type : array-like [elements must be real numbers]

Default : [-3, -2.95, -2.9, ... , 0.9, 0.95, 1.0]

The bins in each [X/H] abundance and [X/Y] abundance ratio to sort the normalized stellar metallicity distribution function into. By default, VICE sorts everything into 0.05-dex bins between [X/H] and [X/Y] = -3 and +1.

Note: The metallicity distributions reported by VICE are normalized to probability distribution functions (i.e. the integral over all bins is equal to 1).

---

[1] Kalirai et al. (2008), ApJ, 676, 594
[2] Kroupa (2001), MNRAS, 231, 322
[3] Salpeter (1955), ApJ, 131, 161
[4] Weinberg, Andrews & Freudenburg (2017), ApJ, 837, 183

Example Code

```
>>> import numpy as np
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> # 400 bins between -3 and 1
>>> sz.bins = np.linspace(-3, 1, 401)
>>> # 800 bins between -2 and +2
>>> sz.bins = np.linspace(-2, 2, 801)
```

vice.singlezone.delay

Type : real number

Default : 0.15

The minimum delay time in Gyr before the onset of type Ia supernovae associated with a single stellar population. Default value is adopted from Weinberg, Andrews & Freudenburg (2017)[1].

See also:

vice.singlezone.RIa

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.delay
0.15
>>> sz.delay = 0.1
>>> sz.delay
0.1
```

vice.singlezone.RIa

Type : <function> or `str` [case-insensitive]

Default : "plaw"

The delay-time distribution (DTD) for typa Ia supernovae to adopt. If type `str`, VICE will use a built-in DTD:

- "exp" : $R_{\rm Ia} \sim e^{-t}$
- "plaw" : $R_{\rm Ia} \sim t^{-1.1}$

When using the exponential DTD, the e-folding timescale is set by the attribute `tau_ia`.

Functions must accept time in Gyr as the only parameter.

Tip: A custom DTD does not need to be normalized by the user. VICE will take care of this automatically.

[1] Weinberg, Andrews & Freudenburg (2017), ApJ, 837, 183

Note: Saving functional attributes with VICE outputs requires the package dill, an extension to `pickle` in the Python standard library. It is recommended that VICE users install dill >= 0.2.0.

Example Code

```
>>> import math as m
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.RIa = "exp"
>>> def f(t):
        if t < 0.2:
                return 1
        else:
                return m.exp(-(t - 0.2) / 1.4)
>>> sz.RIa = f
```

vice.singlezone.Mg0

Type : real number

Default : 6.0e+09

The mass of the ISM gas at time = 0 in $M_\odot$ when ran in infall mode.

Note: This parameter only matters when the simulation is ran in infall mode (i.e. `mode` == "ifr"). In gas mode, `func(0)` specifies the initial gas supply, and in star formation mode, it is `func(0) * tau_star(0)` (modulo the prefactors imposed by gas-dependent star formation efficiency, if applicable).

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.Mg0 = 5.0e+09
>>> sz.Mg0 = 0.
```

vice.singlezone.smoothing

Type : real number

Default : 0.0

The outflow smoothing timescale in Gyr (Johnson & Weinberg 2020[1]). This is the timescale on which the star formation rate is time-averaged before determining the outflow rate via the mass loading factor (attribute `eta`). For an outflow rate $\dot{M}_{\rm out}$ and a star formation rate $\dot{M}_\star$ with a smoothing time $\tau_{\rm s}$:

$$\dot{M}_{\rm out} = \eta(t)\langle\dot{M}_\star\rangle_{\tau_{\rm s}}$$

[1] Johnson & Weinberg (2020), MNRAS, 498, 1364

The traditional relationship of $\dot{M}_{\rm out} = \eta\dot{M}_\star$ is recovered when the user specifies a smoothing time that is smaller than the timestep size.

Note: While this parameter time-averages the star formation rate, it does NOT time-average the mass-loading factor.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.smoothing = 0.0
>>> sz.smoothing = 0.5
>>> sz.smoothing = 1.0
```

vice.singlezone.tau_ia

Type : real number

Default : 1.5

The e-folding timescale in Gyr of an exponentially decaying delay-time distribution in type Ia supernovae.

Note: Because this is an e-folding timescale, it only matter when the attribute `RIa` == "exp".

See also:

vice.singlezone.RIa

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example", RIa = "exp")
>>> sz.tau_ia = 1.0
>>> sz.tau_ia = 1.5
>>> sz.tau_ia = 2.0
```

vice.singlezone.tau_star

Type : real number or <function>

Default : 2.0

The star formation rate per unit gas supply in Gyr, defined by

$$\tau_\star \equiv M_{\rm g}/\dot{M}_\star$$

where $M_{\rm g}$ is the ISM gas mass and $\dot{M}_\star$ is the star formation rate. Numbers will be interpreted as a constant value. Functions must accept either one or two parameters, the first of which will always be time in Gyr. In infall and gas modes, the second parameter will always be interpreted as the gas mass in $M_\odot$, but in star formation mode, it will be

interpreted as the star formation rate in $M_\odot/yr$. This approach allows this attribute to vary with either the gas mass or the star formation rate in simulation (depending on which mode the model is ran in).

New in version 1.2.0: Prior to version 1.2.0, a functional form for this attribute had to accept only one numerical parameter, always interpreted as time in Gyr.

Tip: In infall and gas modes, this parameter can be set to infinity to forcibly shut off star formation.

Tip: When adopting a functional form for this attribute which depends on the gas supply itself via a pure power-law, we recommend users make use of the attributes `schmidt`, `schmidt_index`, and `MgSchmidt`. These control the parameters of the power-law and allow VICE to calculate the values internally, resulting in shorter integration times.

Note: When the attribute `schmidt == True`, this is interpreted as the prefactor on gas-dependent star formation efficiency:

$$\tau_\star^{-1} = \tau_{*,\text{specified}}^{-1}\left(\frac{M_\text{g}}{M_{\text{g,Schmidt}}}\right)^\alpha$$

where α is the power-law index on gas-dependent star formation efficiency, set by the attribute `schmidt_index`, and $\tau_{*,\text{specified}}$ is the value of this attribute.

Note: Saving functional attributes with VICE outputs requires the package dill, an extension to `pickle` in the Python standard library. It is recommended that VICE user's install dill >= 0.2.0.

Note: In the interstellar medium and star formation literature, this parameter is often referred to as the depletion timescale. In this documentation and in much of the galactic chemical evolution literature, it is often referred to as the "star formation efficiency timescale."

Note: If the user assigns this attribute a function which is ran through a Cython compiler, the corresponding Cython source code must be compiled with the `binding = True` directive. This allows VICE to inspect the signature of the compiled function; otherwise, assigning the function to this attribute will raise a `ValueError`.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.tau_star = 1
>>> def f(t):
        if 5 <= t <= 6:
                return 1
        else:
                return 2
>>> sz.tau_star = f
```

vice.singlezone.dt

Type : real number

Default : 0.01

The timestep size in Gyr to use in the integration.

Note: For fine timestepping, this affects the total integration time with a dt^{-2} dependence. For coarse timestepping, the integration time is approximately constant, due to it being dominated not by timestepping but by write-out.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.dt = 0.02
>>> sz.dt = 0.005
```

vice.singlezone.schmidt

Type : bool

Default : False

If True, the simulation will adopt a gas-dependent scaling of the star formation efficiency timescale $\tau_\star$. At each timestep, $\tau_\star$ is determined via:

$$\tau_\star(t) = \tau_{\star,\text{specified}}(t) \left(\frac{M_g}{M_{g,\text{Schmidt}}}\right)^{-\alpha}$$

where $\tau_{\star,\text{specified}}(t)$ is the user-specified value of the attribute `tau_star`, M_g is the mass of the interstellar medium, $M_{g,\text{Schmidt}}$ is the normalization thereof (attribute `MgSchmidt`), and α is the power-law index set by the attribute `schmidt_index`.

This is an application of the Kennicutt-Schmidt star formation law to the single-zone approximation (Kennicutt 1998[1]; Schmidt 1959[2], 1963[3]).

If False, this parameter does not impact the star formation efficiency that the user has specified.

Note: This attribute is irrelevant when the attribute `tau_star` is a function of two variables.

See also:

- vice.singlezone.tau_star
- vice.singlezone.schmidt_index
- vice.singlezone.MgSchmidt

[1] Kennicutt (1998), ApJ, 498, 541
[2] Schmidt (1959), ApJ, 129, 243
[3] Schmidt (1963), ApJ, 137, 758

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.schmidt = True
>>> sz.schmidt = False
```

vice.singlezone.schmidt_index

Type : real number

Default : 0.5

The power-law index on gas-dependent star formation efficiency, if applicable:

$$\tau_\star^{-1} \sim M_g^\alpha$$

Note: This attribute is irrelevant when the attribute `tau_star` is a function of two variables.

Note: This number should be 1 less than the power law index which describes the scaling of star formation with the surface density of gas.

See also:

- vice.singlezone.tau_star
- vice.singlezone.schmidt
- vice.singlezone.schmidt_index

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.schmidt_index = 0.5
>>> sz.schmidt_index = 0.4
```

vice.singlezone.MgSchmidt

Type : real number

Default : 6.0e+09

The normalization of the gas supply in $M_\odot$ when star formation efficiency is dependent on the gas supply:

$$\tau_\star \sim \left(\frac{M_g}{M_{g,\text{Schmidt}}}\right)^{-\alpha}$$

where α is specified by the attribute `schmidt_index`.

Note: This attribute is irrelevant when the attribute `tau_star` is a function of two variables.

Tip: In practice, this quantity should be comparable to a typical gas supply of the simulated zone so that the actual star formation efficiency at a given timestep is near the user-specified value.

See also:

- vice.singlezone.tau_star
- vice.singlezone.schmidt
- vice.singlezone.schmidt_index

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.MgSchmidt = 5.0e+09
```

vice.singlezone.m_upper

Type : real number

Default : 100

The upper mass limit on star formation in $M_\odot$.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.m_upper = 120
```

vice.singlezone.m_lower

Type : real number

Default : 0.08

The lower mass limit on star formation in $M_\odot$.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.m_lower = 0.1
```

vice.singlezone.postMS

Type : real number

Default : 0.1

New in version 1.1.0.

The ratio of a star's post main sequence lifetime to its main sequence lifetime.

Note: This parameter has no impact when the stellar mass-lifetime relations of either Vincenzo et al. (2016)[1] or Kodama & Arimoto (1997)[2] are adopted (i.e. when `vice.mlr.setting` is either `"vincenzo2016"` or `"ka1997"`). These parameterizations as they are built into VICE quantify the *total* lifetimes of stars, making a prescription for the post main sequence lifetimes superfluous.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.postMS = 0.12
```

vice.singlezone.Z_solar

Type : real number

Default : 0.014

The metallicity by mass of the sun $M_Z/M_\odot$. This is used in calibrating the total metallicity of the ISM, which is necessary when there are only a few elements tracked by the simulation with metallicity dependent yields. This scaling is implemented as follows:

$$Z_{\rm ISM} = Z_\odot \left[\sum_i Z_i\right] \left[\sum_i Z_i^\odot\right]^{-1}$$

where the summation is taken over the elements tracked by the simulation.

Note: The default value is the metallicity calculated by Asplund et al. (2009)[1]; VICE by default adopts the Asplund et al. (2009) measurements on their element-by-element basis in calculating [X/H] and [X/Y] in simulations. Users who wish to adopt a different model for the composition of the sun should modify **both** this value **and** the element-by-element entires in `vice.solar_z`.

---

[1] Vincenzo et al. (2016), MNRAS, 460, 2238
[2] Kodama & Arimoto (1997), A&A, 320, 41
[1] Asplund et al. (2009), ARA&A, 47, 481

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.Z_solar = 0.014
```

vice.singlezone.agb_model

[DEPRECATED]

Type : `str` [case-insensitive]

Default : None

Deprecated since version 1.2.0: Users should instead use the `vice.yields.agb.settings` `dataframe` to declare their yields. These allow the same keywords as this attribute as well as user-constructed functions of stellar mass and metallicity.

A keyword denoting which stellar mass-metallicity grid of fractional nucleosynthetic yields from asymptotic giant branch (AGB) stars to adopt.

Recognized Keywords:

- "cristallo11"[1]
- "karakas10"[2]

Note: If the Karakas (2010) set of yields are adopted and any elements tracked by the simulation are heavier than nickel, a LookupError will be raised. The Karakas (2010) study did not report yields for elements heavier than nickel.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example", elements = ["c", "n", "o"])
>>> sz.agb_model = "karakas10"
>>> sz.agb_model = "cristallo11"
```

vice.multizone

An object designed to run simulations of chemical enrichment under the multi-zone approximation for user-specified parameters. At its core, this is an array of `singlezone` objects.

Signature: vice.multizone(name = "multizonemodel", n_zones = 10, n_stars = 1, simple = False, verbose = False)

New in version 1.2.0.

[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Karakas (2010), MNRAS, 403, 1413

Parameters

name [str [default]["multizonemodel"]] The attribute `name`, initialized via keyword argument. See below.

n_zones [int [default][10]] The attribute `n_zones`, initialized via keyword argument. See below.

n_stars [int [default][1]] The attribute `n_stars`, initialized via keyword argument. See below.

simple [bool [default][False]] The attribute `simple`, initialized via keyword argument. See below.

verbose [bool [default][False]] The attribute `verbose`, initialized via keyword argument. See below.

Attributes

name [str [default]["multizonemodel"]] The name of the simulation. Output will be stored in a directory under this name with a ".vice" extension.

zones [zone_array [default][always `singlezone` objects]] An array-like object of `singlezone` objects, detailing the evolutionary parameters of each zone.

migration [`migration.specs` [default][no migration]] The migration specifications for both gas and stars.

n_zones [int [default][10]] The number of zones in the model.

Note: This cannot be changed after creation of the object.

Note: If this is equal to 1, VICE will construct a `singlezone` object rather than a `multizone` object.

Note: See below on your system's maximum number of open file descriptors.

---

n_stars [int [default][1]] The number of star particles forming in each zone at each timestep.

simple [bool [default][False]] If True, each individual zone will be simulated as a one-zone model, ignoring all migration prescriptions.

verbose [bool [default][False]] Whether or not to print to the console as the simulation runs.

Functions

run [[instancemethod]] Run the simulation

from_output [[classmethod]] Obtain a `multizone` object with the parameters of one that produced an output.

Notes

Implementation VICE uses a forward Euler approach to handle its timestepping. Although this isn't the highest numerical resolution timestepping method, the dominant source of error in VICE is not in the numerics but in the approximations built into the model itself. Solutions in which the numerical error is adequately small can be achieved with reasonable timestep sizes. Furthermore, the forward Euler approach allows VICE to treat the discretization of timesteps to correspond directly to a discretization of stellar populations, simplifying its implementation and allowing fast numerical solutions. The exact timestamps at which functions of time describing evolutionary parameters will be evaluated is also a simple calculation as a result, since they will all be integer multiples of the timestep size. For further details, see VICE's science documentation: https://vice-astro.readthedocs.io/en/latest/science_documentation/index.html

Maximum Number of Open File Descriptors All operating systems by default limit the number of open file descriptors per process, though with administrator's privileges this number can be temporarily raised. For each simulation, VICE must open two files per zone, plus one to write the star particle information to. Simulations with a particularly high number of zones will therefore require a relatively high number of files to be opened. The system-enforced maximum number of open file descriptors can be accessed by running `ulimit -n` in a `Unix` terminal. Users should be careful to ensure that this number is higher than what is required for a given simulation at runtime. Users on machines for which they don't have administrator's privileges should speak with their administrator about raising their limit if their models require more zones than their current settings will allow.

Computational Overhead The number of zone indeces that VICE must keep track of scales with the product of the number of timesteps, number of zones, and the attribute `nstars`. The computational overhead can, in theory, be arbitarily large as long as the system has the required space. This is in addition to the abundance information at all timesteps required for calculating metallicity-dependent yields and recycling rates from previous stellar populations. In practice, the `milkyway` object (a built-in subclass of this one) using 200 zones, 1,321 timesteps, and 8 stellar populations per zone per timestep with two elements can require up to ~2 GB of RAM at any given moment, processing nearly 30 GB worth of data in total with otherwise default parameters. These models can take up to ~5 hours to fully integrate on a single CPU. For comparison, the same model with 1 stellar population per zone per timestep and 256 timesteps, but still 200 zones, requires up to ~800 MB of RAM and requires only ~2.5 minutes to fully integrate over ~8.5 GB of data. Using 40 zones instead of 200 then requires ~250 MB of RAM and fully integrates inover ~7 GB of total data in ~11 seconds.

Relationship to ``vice.singlezone`` This object makes use of composition. At its core, it is simply an array of `singlezone` objects, which the user may manipulate like all other `singlezone` objects.

Example Code

```
>>> import vice
>>> mz = vice.multizone(n_zones = 3)
>>> mz
        vice.multizone{
                name ------------> multizonemodel
                n_zones ---------> 3
                n_stars ---------> 1
                verbose ---------> False
                simple ----------> False
                zones -----------> ['zone0', 'zone1', 'zone2']
                migration -------> Stars: <function _DEFAULT_STELLAR_MIGRATION_ at␣
↪0x10e2150e0>
                                                ISM:     MigrationMatrix{
                0 ----------> {0.0, 0.0, 0.0}
                1 ----------> {0.0, 0.0, 0.0}
```

(continues on next page)

(continued from previous page)

```
                2 ---------> {0.0, 0.0, 0.0}
        }
        }
```

vice.multizone.run

Run the simulation.

Signature: x.run(output_times, capture = False, overwrite = False)

Parameters

x [`multizone`] An instance of this class.

output_times [array-like [elements are real numbers]] The times in Gyr at which VICE should record output from the simulation. These need not be sorted from least to greatest.

capture [`bool` [default][False]] If `True`, an output object containing the results of the simulation will be returned.

overwrite [`bool` [default][False]] If `True`, will force overwrite any files with the same name as the simulation output files.

pickle [`bool` [default][True]] If `True`, VICE will save the attributes of this object with the output. See below.

Returns

out [`multioutput` [only returned if `capture == True`]] A `multioutput` object produced from this simulation's output.

Raises

- **RuntimeError**
  - A migration matrix cannot be setup properly according to the current specifications.
  - Any of the zones have duplicate names.
  - The timestep size is not uniform across all zones.
- **ScienceWarning**
  - Any of the attributes `IMF`, `recycling`, `delay`, `RIa`, `schmidt`, `schmidt_index`, `MgSchmidt`, `m_upper`, `m_lower`, or `Z_solar` aren't uniform across all zones.

Other exceptions are raised by `vice.singlezone.run`.

Notes

Note: Calling this function only causes VICE to produce the output files. The `multioutput` class handles the reading and storing of the simulation results.

Note: Saving functional attributes with VICE outputs requires the package dill, an extension to `pickle` in the python standard library. It is recommended that VICE users install dill >= 0.2.0.

Note: When `overwrite == False`, and there are files under the same name as the output produced, this acts as a halting function. VICE will wait for the user's approval to overwrite existing files in this case. If users are running multiple simulations and need their integrations not to stall, they must specify `overwrite = True`.

Note: VICE will always write output at the final timestep of the simulation. This may be one timestep beyond the last element of the specified `output_times` array.

Note: If the keyword argument `pickle == True`, VICE will attempt to save a pickle of each attribute of this class. VICE may not be able to save some attributes, in particular those that are themselves instances of another class, especially if they have data or C-extensions attached to them.

These data make up a significant fraction of the disk usage of output files. Therefore, if many multizone models are to be ran, users are recommended to specify `pickle = False` to lower the storage space required. This will, however, render the vice.multizone.from_output function useless for that output.

Example Code

```
>>> import numpy as np
>>> import vice
>>> mz = vice.multizone(name = "example")
>>> outtimes = np.linspace(0, 10, 1001)
>>> mz.run(outtimes)
```

vice.multizone.from_output

Obtain an instance of the `multizone` class given either the path to an output of a `multioutput` object itself.

Signature: vice.multizone.from_output(arg)

New in version 1.2.0.

Parameters

arg [`str` or `multioutput`] The full or relative path to the output directory; the '.vice' extension is not necessary. Alternatively, an output object.

Returns

mz [`multizone`] A `multizone` object with the same parameters as the one which produced the output.

> **Note:** `multizone` simulations by default save a copy of their attributes with their output, a feature which makes this function possible. If the user calls the `run` function with the keyword argument `pickle = False`, the necessary files to reconstruct the simulation will not be produced. In this case, this function will return a `multizone` object with the default parameters.

Raises

- **TypeError**
  - `arg` is neither a `multioutput` object nor a string.
- **IOError [Only occurs if the output has been altered]**
  - The output is missing files
- **UserWarning**
  - Attributes were not saved with the output at user's request, and the default `multizone` object will be returned.

Notes

Note: If arg is either a `singlezone` output or an `output` object, a `singlezone` object will be returned.

Note: This function serving as the reader, the writer is the vice.core.multizone._multizone.c_multizone.pickle function, implemented in Cython.

Example Code

```
>>> import numpy as np
>>> import vice
>>> vice.multizone(name = "example", n_zones = 3)
>>> mz.run(np.linspace(0, 10, 1001))
>>> mz = vice.multizone.from_output("example")
>>> mz
        vice.multizone{
                name ----------> example
```

(continues on next page)

(continued from previous page)

```
                n_zones --------> 3
                n_stars --------> 1
                verbose --------> False
                simple ---------> False
                zones ----------> ['zone0', 'zone1', 'zone2']
                migration ------> Stars: <function _DEFAULT_STELLAR_MIGRATION_ at␣
↪0x111393f80>
                                                ISM:     MigrationMatrix{
                        0 ---------> {0.0, 0.0, 0.0}
                        1 ---------> {0.0, 0.0, 0.0}
                        2 ---------> {0.0, 0.0, 0.0}
                }
        }
```

vice.multizone.name

Type : `str`

Default : "multizonemodel"

The name of the simulation. The output will be stored in a directory under this name with the extension ".vice". This can also be of the form `./path/to/directory/name` and the output will be stored there.

Tip: Users need not interact with any of the output files. The `multioutput` object is designed to read in all of the results automatically.

Tip: By forcing a ".vice" extension on the output directory, users can run `<command> \*.vice` in a terminal to run commands over all VICE outputs in a given directory.

Note: The outputs of this class contain the output from each individual zone in their respective ".vice" directories as well as the abundances, age information, and initial and final zone numbers of all star particles in an ascii file named "tracers.out". Like the "history.out" and "mdf.out" files associated with the `singlezone` object, this allows this information to be analyzed in languages other than python with ease.

See also:

`vice.singlezone.name`

Example Code

```
>>> import vice
>>> mz = vice.multizone(name = "example")
>>> mz.name = "another_name"
```

vice.multizone.zones

Type : `zone_array`

Default : `n_zones` singlezone objects with default parameters.

A 1-dimensional array-like object which forces all elements to be instances of the `singlezone` class. The attributes of each zone can be manipulated in exactly the same way as other `singlezone` objects.

Note: The output associated with each zone will be stored inside the output directory from this class. For example, for a multizone object whose name is "multizonemodel" with a zone named "onezonemodel", the output will be stored at the path `multizonemodel.vice/onezonemodel.vice`.

See also:

`vice.singlezone`

Example Code

```
>>> import vice
>>> mz = vice.multizone(name = "example")
>>> mz.zones[0]
        vice.singlezone{
                name ----------> zone0
                func ----------> <function _DEFAULT_FUNC_ at 0x10f896290>
                mode ----------> ifr
                verbose --------> False
                elements -------> ('fe', 'sr', 'o')
                IMF ------------> kroupa
                eta ------------> 2.5
                enhancement ----> 1.0
                entrainment ----> <entrainment settings>
                Zin ------------> 0.0
                recycling ------> continuous
                delay ----------> 0.15
                RIa ------------> plaw
                Mg0 ------------> 6000000000.0
                smoothing ------> 0.0
                tau_ia ---------> 1.5
                tau_star -------> 2.0
                schmidt --------> False
                schmidt_index --> 0.5
                MgSchmidt ------> 6000000000.0
                dt -------------> 0.01
                m_upper --------> 100.0
                m_lower --------> 0.08
                postMS ---------> 0.1
                Z_solar --------> 0.014
                bins -----------> [-3, -2.95, -2.9, ... , 0.9, 0.95, 1]
        }
```

vice.multizone.migration

Type : `migration.specs`

Default : No migration of either gas or stars.

An object which stores the migration specifications of the multizone model.

Attributes

gas [`mig_matrix`] A matrix describing how gas moves between zones.

stars [<function>] The migration settings for star particles.

See also:

- `vice.multizone.migration.gas`
- `vice.multizone.migration.stars`

Example Code

```
>>> import vice
>>> mz = vice.multizone(name = "example")
>>> mz.migration.gas[1][0] = 0.05
>>> mz.migration.gas[0][1] = 0.05
>>> def f(zone, tform, time):
        '''
        stars born in zone 0 and 1 swap positions when they're more
        than 1 Gyr old.
        '''
        if zone == 0:
                if time - tform > 1:
                        return 1
                else:
                        return 0
        elif zone == 1:
                if time - tform > 1:
                        return 0
                else:
                        return 1
        else:
                return zone
>>> mz.migration.stars = f
```

vice.multizone.n_zones

Type : `int`

Default : 10

The number of zones in the simulation.

Note: This value may only be set upon initialization of the `multizone` object. In order to change the number of zones in the model, a new `multizone` object must be created.

Example Code

```
>>> import vice
>>> mz1 = vice.multizone(name = "example1", n_zones = 8)
>>> mz2 = vice.multizone(name = "example2", n_zones = 12)
>>> mz2.n_zones
        12
```

vice.multizone.n_stars

Type : `int`

Default : 1

The number of star particles to form per zone per timestep. These are tracer particles which are stand-ins for entire stellar populations which form and migrate between zones according to the attribute `migration.stars`.

Note: If the star formation rate varies in the simulation, this will impact the simulation by forming star particles of different masses as opposed to a different number of star particles.

Example Code

```
>>> import vice
>>> mz = vice.multizone(name = "example")
>>> mz.n_stars
        1
>>> mz.n_stars = 3
>>> mz.n_stars
        3
```

vice.multizone.verbose

Type : `bool`

Default : `False`

If True, the simulation will print to the console as it evolves.

Example Code

```
>>> import vice
>>> mz = vice.multizone(name = "example")
>>> mz.verbose = True
```

vice.multizone.simple

Type : `bool`

Default : `False`

If `True`, the star particles' zone numbers at timestep between formation and the final timestep will be ignored. Each zone will evolve independently, and mixing will be accounted for only at the final timestep. If `False`, this information will be taken into account as the simulation evolves.

Warning: Simulating all zones as a one-zone model will neglect all time-dependent migration prescriptions built into this model, with migration being a purely post-processing prescription in these cases.

Raises

- **ScienceWarning**
  - This attribute is set to `True`.

Example Code

```
>>> import vice
>>> mz = vice.multizone(name = "example")
>>> mz.simple
        False
```

vice.milkyway

An object designed for running chemical evolution models of Milky Way-like spiral galaxies. Inherits from `vice.multizone`.

This object models the Milky Way as a series of concentric annuli of uniform width. A prescription for stellar migration based on the `h277` hydrodynamical simulation (a part of the `g14` simulation suite, Christensen et al. 2012)[1], an observationally motivated star formation law, and a scaling of the outflow mass loading factor η with radius tuned to predict an observationally motivated radial abundance gradient are included by default. For details, see discussion in Johnson et al. (2021)[2].

Signature: vice.milkyway(zone_width = 0.5, name = "milkyway", n_stars = 1, simple = False, verbose = False, N = 1e5, migration_mode = "diffusion")

New in version 1.2.0.

See also:

- `vice.multizone`
- `vice.toolkit.J21_sf_law`
- `vice.toolkit.hydrodisk.hydrodiskstars`
- `vice.singlezone`

Parameters

zone_width [`float` [default][0.5]] The radial width of each annulus in kpc.

name [`str` [default]["milkyway"]] The name of the simulation. Output will be stored in a directory under this name with a ".vice" extension.

n_stars [`int` [default][1]] The number of stellar populations forming in each zone at each timestep.

simple [`bool` [default][False]] If True, VICE will run the model as a series of one-zone models. If False, information at intermediate timesteps will be taken into account.

verbose [`bool` [default][False]] Run the model with verbose output.

N [`int` [default][1e5]] An estimate of the number of total stellar populations that will be simulated. This keyword will be passed to the `hydrodiskstars` object implementing the stellar migration scheme.

migration_mode [`str` [default]["diffusion"]] A string denoting the time-dependence of stellar migration. This keyword will be passed to the `hydrodiskstars` object implementing the stellar migration scheme.

Attributes

annuli [`list`] The radii representing divisions between annuli in the disk model in kpc.

zone_width [`float` [default][0.5]] The radial width of each annulus in kpc.

evolution [<`function`> [default][milkyway.default_evolution]] A function of galactocentric radius in kpc and time in Gyr, respectively. Returns either the surface density of gas in $M_\odot$, the surface density of infall, or the surface density of star formation in $M_\odot yr^{-1} kpc^{-2}$. The interpretation of the return value is set by the attribute `mode`.

[1] Christensen et al. (2012), MNRAS, 425, 3058
[2] Johnson et al. (2021), MNRAS, 508, 4484

Note: This is the **only** object in the current version of VICE which formulates an evolutionary parameter in terms of surface densities. This is done because many physical quantities are reported as surface densities in the astronomical literature. The `singlezone` and `multizone` objects, however, formulate parameters in terms of mass, out of necessity for the implementation.

mode [`str` [case-insensitive] [default]["ifr"]] The interpretation of the attribute `evolution`. Either "sfr" for star formation rate, "ifr" for infall rate, or "gas" for the ISM gas supply.

elements [`tuple` [elements of type str] [default][("fe", "sr", "o")]] The elements to calculate abundances for in running the model.

IMF [`str` or <`function`> [default]["kroupa"]] The stellar initial mass function to assume. Strings denote built-in IMFs from the literature. Functions will be interpreted as a custom distribution of zero-age main sequence masses in $M_\odot$.

Built-in IMFs:

- "kroupa": Kroupa (2001)[3]
- "salpeter": Salpeter (1955)[4]

mass_loading [<`function`> [default][milkyway.default_mass_loading]] The mass loading factor as a function of galactocentric radius in kpc describing the efficiency of outflows.

dt [`float` [default][0.01]] The timestep size in Gyr to use when running the model.

bins [`list` [default][[-3.0, -2.95, -2.9, ... , 0.9, 0.95, 1.0]]] The bins within which to sort the normalized stellar metallicity distribution function in each [X/H] and [X/Y] abundance ratio measurement.

delay [real number [default][0.15]] The minimum delay time in Gyr before the onset of type Ia supernovae associated with a single stellar population.

RIa [`str` [case-insensitive] or <`function`> [default]["plaw"]] The SN Ia delay-time distribution (DTD) to adopt. Strings denote built-in DTDs and functions must accept time in Gyr as a parameter.

smoothing [`float` [default][0.0]] The outflow smoothing timescale in Gyr. See discussion in Johnson & Weinberg (2020)[5].

tau_ia [`float` [default][1.5]] The e-folding timescale of the SN Ia DTD. Only relevant when the attribute `RIa == "exp"`.

m_upper [`float` [default][100]] The upper mass limit on star formation in $M_\odot$.

m_lower [`float` [default][0.08]] The lower mass limit on star formation in $M_\odot$.

postMS [real number [default][0.1]] The lifetime ratio of the post main sequence to main sequence phases of stellar evolution.

Z_solar [real number [default][0.14]] The adopted metallicity by mass of the sun.

Other attributes are inherited from `vice.multizone`.

Note: The `h277` data is not included in VICE's distribution, but is available in its GitHub repository. When users first create a `milkyway` object, it will download the data automatically and store it internally for future use. With a good internet connection, this process takes about 1 minute to complete, and need not be repeated. If the download fails, it's likely it has to do with not having administrator's privileges over your system. Users in this situation should speak with their administrator, who would then be able to download their data by running the following on their system:

[3] Kroupa (2001), MNRAS, 322, 231
[4] Salpeter (1955), ApJ, 121, 161
[5] Johnson & Weinberg (2020), MNRAS, 498, 1364

```
>>> import vice
>>> vice.toolkit.hydrodisk.data.download()
```

Note: This object, by default, will shut off star formation at $R > 15.5$ kpc by setting the star formation efficiency timescale to a very large number. This can be overridden at any time by resetting the attribute `tau_star` of each zone.

Note: See documentation of `vice.multizone` base class for information on the implementation and required computational overhead of this and other applications of VICE's multizone capabilities. In theory, the data involved can be arbitrarily large provided the system has the space, but coarse versions of finely sampled models often require only minutes to fully integrate, simplifying the debugging process.

Functions

run [[instancemethod]] Run the simulation.

default_evolution [[staticmethod]] The default value of the functional attribute `evolution`.

default_mass_loading [[staticmethod]] The default value of the functional attribute `mass_loading`.

Example Code

```
>>> import vice
>>> import numpy as np
>>> mw = vice.milkyway(name = "example", zone_width = 1)
>>> mw.n_zones
20
>>> mw.n_stars
1
>>> mw.name
"example"
>>> mw.run(np.linspace(0, 13.2, 1321), overwrite = True)
```

vice.milkyway.annuli

Type : `list` [elements of type `float`]

The radii representing divisions between annuli in the disk model in kpc. This property is determined by the `zone_width` attribute, and cannot be modified after initialization of a `milkyway` object.

See also:

vice.milkyway.zone_width

While this attribute stores the radii representing bounds between annuli, the `singlezone` object corresponding to each individual annulus is stored as an array in the `zones` attribute, inherited from the `multizone` class.

By default, they will be named where "zone0" is the zero'th element of the `zones` attribute, corresponding to the innermost zone. The second innermost zone will be the first element of the `zones` attribute, and by default will be named "zone1", and so on.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example", zone_width = 0.2)
>>> mw.annuli
[0.0,
 0.2,
 0.4,
 ...,
 19.6,
 19.8,
 20.0]
```

vice.milkyway.zone_width

Type : `float`

Default : 0.5

The width of each annulus in kpc. This value can only be set at initialization of the `milkyway` object.

See also:

vice.milkyway.annuli

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example", zone_width = 0.2)
>>> mw.zone_width
0.2
```

vice.milkyway.evolution

Type : `<function>`

Default : vice.milkyway.default_evolution

As a function of radius in kpc and time in Gyr, respectively, either the surface density of gas in $M_\odot kpc^{-2}$, the surface density of star formation in $M_\odot kpc^{-2} yr^{-1}$, or the surface density of infall in $M_\odot kpc^{-2} yr^{-1}$. As in the `singlezone` object, the interpretation is set by the attribute `mode`.

See also:

vice.milkyway.default_evolution

Note: This attribute will always be expected to accept radius in kpc and time in Gyr as parameters, in that order. However, surface densities of star formation and infall will always be interpreted as having units of $M_\odot yr^{-1} kpc^{-2}$ according to convention.

Note: This is the **only** object in the current version of VICE which formulates an evolutionary parameter in terms of surface densities. This is done because many physical quantities are reported as surface densities in the astronomical literature. The `singlezone` and `multizone` objects, however, formulate parameters in terms of mass, out of necessity for the implementation.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
```

vice.milkyway.default_evolution

The default evolutionary function of the `milkyway` object.

Signature: vice.milkyway.default_evolution(radius, time)

Parameters

radius [float] Galactocentric radius in kpc.

time [float] Simulation time in Gyr.

Returns

value [float] Always returns the value of 1.0. The interpretation of this is set by the attribute `mode`. With the default value of "ifr", this represents a uniform surface of infall of 1.0 $M_\odot yr^{-1} kpc^{-2}$.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.evolution
<function vice.milkyway.milkyway.milkyway.default_evolution(radius, time)>
>>> vice.milkyway.default_evolution(10, 1)
1.0
>>> vice.milkyway.default_evolution(5, 4)
1.0
```

vice.milkyway.mode

Type : `str` [case-insensitive]

Default : "ifr"

The interpretation of the attribute `evolution`.

- mode = "ifr": The value returned from the attribute `evolution` represents the surface density of gas infall into the interstellar medium in $M_\odot kpc^{-2} yr^{-1}$.
- mode = "sfr": The value returned from the attribute `evolution` represents the surface density of star formation in $M_\odot kpc^{-2} yr^{-1}$.
- mode = "gas": The value returned from the attribute `evolution` represents the surface density of the interstellar medium in $M_\odot kpc^{-2}$.

Note: The attribute `evolution` will always be expected to accept radius in kpc and time in Gyr as the first and second parameters, respectively. However, infall and star formation histories will be interpreted as having units of $M_\odot yr^{-1}$ according to convention.

Note: Updating the value of this attribute also updates the corresponding attribute of the `J21_sf_law` star formation law where it has been assigned the attribute `tau_star`.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.mode
"ifr"
>>> mw.mode = "sfr"
>>> mw.mode
"sfr"
```

vice.milkyway.elements

Type : `tuple` [elements of type str [case-insensitive]]

Default : ("fe", "sr", "o")

The symbols of the elements to track the enrichment for (case-insensitive). The more elements that are tracked, the longer the simulation will take, but the better calibrated is the total metallicity of the ISM in handling metallicity-dependent yields.

Tip: The order in which the elements appear in this tuple will dictate the abundance ratios that are quoted in the final stellar metallicity distribution function. That is, if element X appears before element Y, then VICE will determine the MDF in $dN/d[Y/X]$ as opposed to $dN/d[X/Y]$. The elements that users intend to use as "reference elements" should come earliest in this list.

Note: All versions of VICE support the simulation of all 76 astrophysically produced elements between carbon ("c") and bismuth ("bi"). Versions >= 1.1.0 also support helium ("he").

Note: Some of the heaviest elements that VICE recognizes have statistically significant enrichment from r-process nucleosynthesis[1]. Simulations of these elements with realistic parameters and realistic nucleosynthetic yields will underpredict the absolute abundances of these elements. However, if these nuclei are assumed to be produced promptly following the formation of a single stellar population, the yield can be added to the yield from core collapse supernovae, which in theory can describe the total yield from all prompt sources[2].

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.elements
("fe", "sr", "o")
>>> mw.elements = ["mg", "fe", "n", "c", "o"]
>>> mw.elements
("mg", "fe", "n", "c", "o")
```

vice.milkyway.IMF

Type : `str` [case-insensitive] or <`function`>

Default : "kroupa"

New in version 1.2.0: In version >= 1.2.0, users may construct a function of mass to describe the IMF.

The assumed stellar initial mass function (IMF). If assigned a string, VICE will adopt a built-in IMF. Functions must accept stellar mass as the only parameter and are expected to return the value of the IMF at that mass (it need not be normalized).

Built-in IMFs:

- "kroupa"[1]
- "salpeter"[2]

Note: VICE has analytic solutions to the *cumulative return fraction* and the *main sequence mass fraction* for built-in IMFs. If assigned a function, VICE will calculate these quantities numerically, increasing the required integration time.

[1] Johnson (2019), Science, 363, 474
[2] Johnson & Weinberg (2020), MNRAS, 498, 1364
[1] Kroupa (2001), MNRAS, 322, 231
[2] Salpeter (1955), ApJ, 121, 161

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.IMF = "salpeter"
>>> def f(m):
        if m < 0.5:
                return m**-1.2
        else:
                return m**-2.2
>>> mw.IMF = f
```

vice.milkyway.mass_loading

Type : <function>

Default : vice.milkyway.default_mass_loading

The mass-loading factor as a function of galactocentric radius in kpc describing the efficiency of outflows. For a given star formation rate $\dot{M}_\star$ and an outflow rate $\dot{M}_{\rm out}$, the mass-loading factor is defined as the unitless ratio:

$$\eta \equiv \dot{M}_{\rm out} / \dot{M}_\star$$

This function must return a non-negative real number for all radii defined in the disk model.

Note: This formalism assumes a time-independent mass-loading factor at each radius. To implement a time-dependent alternative, users should modify the attribute `eta` of the `singlezone` objects corresponding to each annulus in this model. See example below.

See also:

vice.singlezone.eta

Example Code

```
>>> import math as m
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> def f(r):
        return 0.5 * m.exp(r / 3)
>>> mw.mass_loading = f
>>> def g(t): # a time-dependent mass-loading factor
        return 3.0 * m.exp(-t / 3)
>>> # assign each individual annulus a time-dependent value
>>> for i in range(mw.n_zones):
>>>     mw.zones[i].eta = g
```

vice.milkyway.default_mass_loading

The default mass loading factor as a function of galactocentric radius in kpc.

Signature: vice.milkyway.default_mass_loading(rgal)

Parameters

rgal [real number] Galactocentric radius in kpc.

Returns

eta [real number] The mass loading factor at that radius, defined by:

$$\eta(r) = (y_{\rm O}^{\rm CC})/Z_{\rm O}^{\odot} 10^{0.08(r - 4\text{ kpc}) - 0.3} - 0.6$$

where $Z_{\rm O}^{\odot}$ is the solar abundance by mass of oxygen and $y_{\rm O}^{\rm CC}$ is the IMF-averaged CCSN yield of oxygen. While these values are customizable through `vice.solar_z` and `vice.yields.ccsne.settings`, this function assumes a value of $Z_{\rm O}^{\odot}$ = 0.00572 (Asplund et al. 2009[1]) and $y_{\rm O}^{\rm CC}$ = 0.015 (Johnson & Weinberg 2020[2], Johnson et al. 2021[3]).

See also:

vice.milkyway.mass_loading

Example Code

```
>>> import vice
>>> vice.milkyway.default_mass_loading(0)
0.029064576665950193
>>> vice.milkyway.default_mass_loading(8)
2.1459664721614495
```

vice.milkyway.dt

Type : `float`

Default: 0.01

The timestep size in Gyr.

[1] Asplund et al. (2009), ARA&A, 47, 481
[2] Johnson & Weinberg (2020), MNRAS, 498, 1364
[3] Johnson et al. (2021), MNRAS, 508, 4484

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example", zone_width = 0.2)
>>> mw.dt
0.01
>>> mw.dt = 0.02
>>> mw.dt
0.02
```

vice.milkyway.bins

Type : `list` [elements must be real numbers]

Default: [-3, -2.95, -2.9, ... , 0.9, 0.95, 1.0]

The bins in each [X/Y] abundance and [X/Y] abundance ratio to sort the normalized stellar metallicity distribution function into. By default, VICE sorts everything into 0.05-dex bins between [X/H] and [X/Y] = -3 and +1.

Note: The metallicity distributions reported by VICE are normalized to probability distribution functions (i.e. the integral over all bins is equal to 1).

See also:

vice.milkyway.elements

Example Code

```
>>> import numpy as np
>>> import vice
>>> mw = vice.milkyway(name = "example", zone_width = 0.2)
>>> mw.bins = np.linspace(-3, 1, 401) # 400 bins between -3 and 1
>>> mw.bins = np.linspace(-2, 2, 801) # 800 bins between -2 and +2
```

vice.milkyway.delay

Type : real number

Default : 0.15

The minimum delay time in Gyr before the onset of type Ia supernovae assocaited with a single stellar population. Default value is adopted from Weinberg, Andrews & Freudenburg (2017)[1].

See also:

vice.milkyway.RIa

[1] Weinberg, Andrews & Freudenburg (2017), ApJ, 837, 183

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.delay
0.15
>>> mw.delay = 0.1
>>> mw.delay
0.1
```

vice.milkyway.RIa

Type : <function> or str [case-insensitive]

Default: "plaw"

The delay-time distribution (DTD) for type Ia supernovae to adopt. If type str, VICE will use a built-in DTD:

- "exp" : $R_{\text{Ia}} \sim e^{-t}$
- "plaw" : $R_{\text{Ia}} \sim t^{-1.1}$

When using the exponential DTD, the e-folding timescale is set by the attribute tau_ia.

Functions must accept time in Gyr as the only parameter and return the rate at that delay-time.

Tip: A custom DTD does not need to be normalized by the user. VICE will take care of this automatically.

Note: Saving functional attribute with VICE outputs requires the package dill, an extension to pickle in the Python standard library. It is recommended that VICE users install dill >= 0.2.0.

Example Code

```
>>> import math as m
>>> import vice
>>> mw = vice.milkway(name = "example")
>>> mw.RIa = "exp"
>>> mw.RIa
"exp"
>>> def f(t):
        if t < 0.2:
                return 1
        else:
                return m.exp(-(t - 0.2) / 1.4)
>>> mw.RIa = f
```

vice.milkyway.smoothing

Type : real number

Default : 0.0

The outflow smoothing in Gyr (Johnson & Weinberg 2020[1]). This is the timescale on which the star formation rate is time-averaged before determining the outflow rate via the mass loading factor (attribute `eta`). For an outflow rate $\dot{M}_{\rm out}$ and a star formation rate $\dot{M}_\star$ with a smoothing time $\tau_{\rm s}$:

$$\dot{M}_{\rm out} = \eta(t)\langle\dot{M}_\star\rangle_{\tau_{\rm s}}$$

The traditional relationship of $\dot{M}_{\rm out} = \eta\dot{M}_\star$ is recovered when the user specifies a smoothing time that is smaller than the timestep size.

Note: While this parameter time-averages the star formation rate, it does NOT time-average the mass-loading factor.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.smoothing = 0.0
>>> mw.smoothing = 0.5
>>> mw.smoothing = 1.0
```

vice.milkyway.tau_ia

Type : real number

Default: 1.5

The e-folding timescale in Gyr of an exponentially decaying delay-time distribution in type Ia supernovae. Default value is adopted from Weinberg, Andrews & Freudenburg (2017)[1].

Note: Because this is an e-folding timescale, it only matters when the attribute `RIa` == "exp".

See also:

vice.milkyway.RIa

[1] Johnson & Weinberg (2020), MNRAS, 498, 1364
[1] Weinberg, Andrews & Freudenburg (2017), ApJ, 837, 183

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.tau_ia = 1.0
>>> mw.tau_ia = 1.5
>>> mw.tau_ia = 2.0
```

vice.milkyway.m_upper

Type : real number

Default : 100

The upper mass limit on star formation in $M_\odot$.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.m_upper = 120
```

vice.milkyway.m_lower

Type : real number

Default : 0.08

The lower mass limit on star formation in $M_\odot$.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.m_lower = 0.1
```

vice.milkyway.postMS

Type : real number

Default : 0.1

The ratio of a star's post main sequence lifetime to its main sequence lifetime.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.postMS = 0.12
```

vice.milkyway.Z_solar

Type : real number

Default : 0.014

The metallicity by mass of the sun $M_Z/M_\odot$. This is used in calibrating the total metallicity of the interstellar medium (ISM), which is necessary when there are only a few elements tracked by the simulation with metallicity dependent yields. This scaling is implemented as follows:

$$Z_{\rm ISM} = Z_\odot \left[\sum_i Z_i\right] \left[\sum_i Z_i^\odot\right]^{-1}$$

where the summation is taken over all elements tracked by the simulation.

Note: The default value is the metallicity calculated by Asplund et al. (2009)[1]; VICE by default adopts the Asplund et al. (2009) measurements on their element-by-element basis in calculating [X/H] and [X/Y] in simulations. Users who wish to adopt a different model for the composition of the sun should modify **both** this value **and** the element-by-element entires in `vice.solar_z`.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.Z_solar = 0.014
```

vice.migration

Migration Tools

Provides objects which store prescriptions for the migration of gas and stars in multizone simulations.

Contents

specs [`object`] An object which stores the migration settings of both gas and stars

migration_matrix [`object`] An object which describes the mass fraction of gas migrating between zones as a function of time.

[1] Asplund et al. (2009), ARA&A, 47, 481

vice.migration.specs

Multizone simulation migration prescriptions for gas and stars.

Signature: vice.migration.specs(n)

New in version 1.2.0.

Parameters

n [int] The number of rows and columns in the gas migration matrix. This is also the number of zones in a multizone model.

Attributes

gas [mig_matrix] A matrix containing the mass fraction of gas moving between zones.

stars [<function>] A function of the zone number and time of star formation. Expected to return a function of time in Gyr describing the zone number at all subsequent times of stars forming in that zone at that time.

Example Code

```
>>> import math
>>> import vice
>>> example = vice.migration.specs(3)
>>> def f(zone, tform, time):
        # swap stars between zones 0 and 1 when they're >1 Gyr old.
        if zone == 0:
                if time - tform > 1:
                        return 1
                else:
                        return 0
        elif zone == 1:
                if time - tform > 1:
                        return 0
                else:
                        return 1
        else:
                return zone
>>> example.stars = f
>>> example.gas[1][0] = 0.1
>>> def g(t):
        return 0.2 * math.exp(-t / 2)
>>> example.gas[0][1] = g
>>> example.gas
        MigrationMatrix{
                0 ---------> {0.0, <function g at 0x120588560>, 0.0}
                1 ---------> {0.1, 0.0, 0.0}
                2 ---------> {0.0, 0.0, 0.0}
        }
```

vice.migration.specs.gas

Type : `migration_matrix`

Default: Zeroes (i.e. no gas migration)

The gas migration matrix. For a multizone simulation with N zones, this is an NxN matrix.

This matrix is defined such that G_{ij} represents the mass fraction of gas that moves from the i'th zone to the j'th zone in a 10 Myr time interval.

Allowed Types

- real number The fraction of gas that migrates in a 10 Myr time interval is constant, given by this value.
- <function> Must accept time in Gyr as the only parameter. The fraction of gas that migrates in a 10 Myr time interval varies with time, and is described by this function.

Notes

The mass fraction of interstellar gas that migration from zone i to zone j at a time t is calculated via

$$f_{ij}(t) = G_{ij}(t)\frac{\Delta t}{10 \text{ Myr}}$$

The mass that migrates is then given by $M_{g,i}f_{ij}(t)$, where $M_{g,i}$ is the total mass of the interstellar medium in zone i.

This guarantees that the amount of gas that migrates in a given time interval does not depend on the timestep size.

Example Code

```
>>> import math
>>> import vice
>>> example = vice.migration.specs(3)
>>> example.gas[1][0] = 0.1
>>> def f(t):
        return math.exp(-t / 2)
>>> example.gas[0][1] = f
>>> example.gas
        MigrationMatrix{
                0 ---------> {0.0, <function f at 0x120588560>, 0.0}
                1 ---------> {0.1, 0.0, 0.0}
                2 ---------> {0.0, 0.0, 0.0}
        }
```

vice.migration.specs.stars

Type : <function>

Default : vice._globals._DEFAULT_STELLAR_MIGRATION_

Note: The default migration setting does not move stars between zones at all.

The stellar migration prescription.

This function must accept at `int` followed by two real numbers as parameters, in that order. These are interpreted as the zone number in which a star particle forms in a multizone simulation, the time at which it forms in Gyr, and the time in the simulation in Gyr.

This function must return an `int` describing the zone number of the star particle at times following its formation.

Notes

The third parameter returned by this function is interpreted as the time since the start of the simulation, **NOT** the age of the star particle in question.

This function will never be called with a third parameter that is smaller than the second parameter. These are times at which star particles have not formed yet.

Tip: If users wish to write extra data for star particles to an output file, they should set up this attribute as an instance of a class. If this class has an attribute `write`, VICE will switch it's value to `True` when setting up star particles for simulation. The lines which write to the output file can then be wrapped in an `if self.write` statement.

Tip: If a multizone simulation will form multiple star particles per zone per timestep, they can be assigned different zone occupation histories by allowing the function of initial zone number and formation time to take a keyword argument `n`. VICE will then call this function with `n = 0`, `n = 1`, `n = 2`, and so on up to `n = n_stars - 1`, where `n_stars` is the number of star particles per zone per timestep.

Example Code

```
>>> import vice
>>> example = vice.migration.specs(3)
>>> def f(zone, tform, time):
        # swap stars between zones 0 and 1 when they're >1 Gyr old
        if zone == 0:
                if time - tform > 1:
                        return 1
                else:
                        return 0
        elif zone == 1:
                if time - tform > 1:
                        return 0
                else:
```

(continues on next page)

(continued from previous page)

```
                        return 1
        else:
                return zone
>>> example.stars = f
```

vice.migration.migration_matrix

A square matrix designed to detail the manner in which gas migrates between zones in multizone models. This is a 2-dimensional array-like object denoted by G_{ij}, defined as the mass fraction of the interstellar gas in zone i that migrates to zone j in a 10 Myr time interval.

Signature: vice.migration.migration_matrix(size)

New in version 1.2.0.

Parameters

size [int] The number of rows and columns. This is also the number of zones in the multizone model.

Attributes

size [int] See parameter "size".

Allowed Data Types

- **real number** G_{ij} does not vary with time, and is given by this value.
- **<function>** G_{ij} varies with time, and is described by this function. Time in Gyr is the only parameter the function takes.

Indexing

- **int, int** [the row and column numbers] The value of G_{ij} can be accessed by indexing this object with i as the first index and j as the second. For instance, `example[1][0]` and `example[1, 0]` both look up $G_{1,0}$.

Functions

- tolist
- tonumpyarray

Example Code

```
>>> import math
>>> import vice
>>> example = vice.migration.migration_matrix(3)
>>> example
        MigrationMatrix{
                0 ---------> {0.0, 0.0, 0.0}
                1 ---------> {0.0, 0.0, 0.0}
                2 ---------> {0.0, 0.0, 0.0}
        }
>>> example[1][0] = 0.1
>>> def f(t):
        return 0.1 * math.exp(-t / 5)
>>> example[0, 1] = f
>>> example
        MigrationMatrix{
                0 ---------> {0.0, <function f at 0x120588560>, 0.0}
                1 ---------> {0.1, 0.0, 0.0}
                2 ---------> {0.0, 0.0, 0.0}
        }
>>> example.tonumpyarray()
        array([[0.0, <function f at 0x120588560>, 0.0],
                [0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0]])
```

vice.migration.migration_matrix.size

Type : `int` [positive definite]

The number of rows and columns of this matrix. This is also the number of zones in the multizone model.

Example Code

```
>>> import vice
>>> example = vice.migration.migration_matrix(3)
>>> example.size
        3
```

vice.migration.migration_matrix.tolist

Obtain a copy of this migration matrix as a list.

Signature: x.tolist()

Parameters

x [migration_matrix] An instance of this class.

Returns

copy [list] A list of all values in this matrix.

Example Code

```
>>> import vice
>>> example = vice.migration.migration_matrix(3)
>>> example.tolist()
        [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

vice.migration.migration_matrix.tonumpyarray

Obtain a copy of this migration matrix as a NumPy array.

Signature: x.tonumpyarray()

Parameters

x [migration_matrix] An instance of this class.

Returns

copy [numpy.ndarray] A NumPy array which stores the same values as **x**.

Raises

- **ModuleNotFoundError [ImportError for python < 3.6]**
  - NumPy could not be imported.

Example Code

```
>>> import vice
>>> example = vice.migration.migration_matrix(3)
>>> example.tonumpyarray()
        array([[0, 0, 0],
               [0, 0, 0],
               [0, 0, 0]])
```

vice.history

Obtain a `history` object from a VICE output containing the time-evolution of the interstellar medium and its relevant abundance information.

Signature: vice.history(name)

Parameters

name [`str`] The full or relative path to the output directory. The '.vice' extension is not required.

Returns

hist [`history` [VICE `dataframe` derived class]] A subclass of the VICE dataframe designed to store the output and to calculate relevant quantities automatically upon indexing.

Raises

- **IOError [Only occurs if the output has been altered]**
  - Output directory not found.
  - Output files not formatted correctly.
  - Other VICE output files are missing from the output.

See also:

vice.core.dataframe.history

Example Code

```
>>> import numpy as np
>>> import vice
>>> vice.singlezone(name = "example").run(np.linspace(0, 10, 1001))
>>> example = vice.history("example")
>>> example["time"][:10]
        [0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]
>>> example["[o/fe]"][:10]
        [-0.30581989611140603,
         -0.3059028126227887,
         -0.3059856206579771,
         -0.3060683202832149,
         -0.30615091156463625,
         -0.30623330628476564,
         -0.30631559283107557,
         -0.3063978595147838,
         -0.30647984166504416,
         -0.3065618040838354]
>>> example[100]
        vice.dataframe{
                time ----------> 1.0
```

(continues on next page)

(continued from previous page)

```
                mgas -----------> 5795119000.0
                mstar ----------> 2001106000.0
                sfr ------------> 2.897559
                ifr ------------> 9.1
                ofr ------------> 7.243899
                eta_0 ----------> 2.5
                r_eff ----------> 0.3534769
                z_in(fe) -------> 0.0
                z_in(sr) -------> 0.0
                z_in(o) --------> 0.0
                z_out(fe) ------> 0.0002769056
                z_out(sr) ------> 3.700754e-09
                z_out(o) -------> 0.001404602
                mass(fe) -------> 1604701.0
                mass(sr) -------> 21.44631
                mass(o) --------> 8139837.0
                z(fe) ----------> 0.00027690561660597 48
                z(sr) ----------> 3.700754031107903e-09
                z(o) -----------> 0.0014046022178319376
                [fe/h] ---------> -0.6682579454664828
                [sr/h] ---------> -1.1074881208001155
                [o/h] ----------> -0.6098426789720387
                [sr/fe] --------> -0.43923017533363273
                [o/fe] ---------> 0.05841526649444406
                [o/sr] ---------> 0.4976454418280768
                z --------------> 0.0033582028978416337
                [m/h] ----------> -0.6200211036287412
                lookback -------> 9.0
        }
```

vice.mdf

Obtain a `fromfile` object from a VICE output containing the metallicity distribution function of stars.

Signature: vice.mdf(name)

Parameters

name [`str`] The full or relative path to the output directory. The '.vice' extension is not required.

Returns

mdf [`fromfile` [VICE `dataframe` derived class]] A subclass of the VICE dataframe designed to handle simulation output.

Raises

- **IOError [Only occurs if the output has been altered]**
  - The output file is not found.
  - The output file is not formatted correctly.
  - Other VICE output files are missing from the output.

Notes

VICE normalizes metallicity distribution functions to a probability density, meaning that the area under the distribution is always equal to one. The value of the distribution in some bin times that bin's width denotes the fraction of stars with metallicities in that bin.

Note: For abundances [X/H] and abundance ratios [X/Y] that in the simulation never achieve a value in the user-specified binspace, the distribution will be `NaN` in all bins.

Note: For an output under a given name, the metallicity distribution function is stored in an ascii text file under name.vice/mdf.out. This allows users to open these files without VICE if necessary.

See also:

vice.core.dataframe.fromfile

Example Code

```
>>> import vice
>>> example = vice.mdf("example")
>>> example.keys()
        ["dn/d[sr/h],",
        "dn/d[sr/fe],"
        "bin_edge_left,"
        "dn/d[o/h],"
        "dn/d[o/fe],"
        "dn/d[fe/h],"
        "bin_edge_right,"
        "dn/d[o/sr]"]
>>> example["bin_edge_left"][:10]
        [-3.0, -2.95, -2.9, -2.85, -2.8, -2.75, -2.7, -2.65, -2.6, -2.55]
>>> example[60]
        vice.dataframe{
                bin_edge_left --> 0.0
                bin_edge_right -> 0.05
                dn/d[fe/h] -----> 0.0
                dn/d[sr/h] -----> 0.0
                dn/d[o/h] ------> 0.0
                dn/d[sr/fe] ----> 0.06001488
                dn/d[o/fe] -----> 0.4337209
```

(continues on next page)

(continued from previous page)

```
                dn/d[o/sr] -----> 0.0
        }
```

vice.output

Reads in the output from singlezone simulations and allows the user to access it easily via dataframes.

Signature: vice.output(name)

Parameters

name [str] The full or relative path to the output directory. The '.vice' extension is not required.

Note: If `name` corresponds to output from the `multizone` class, a `multioutput` object is created instead.

Attributes

name [str] The name of the .vice directory containing the simulation output.

elements [tuple] The symbols of the elements whose enrichment was tracked by the simulation, as they appear on the periodic table.

history [dataframe] The dataframe read in via vice.history.

mdf [dataframe] The dataframe read in via vice.mdf.

agb_yields [dataframe] The asymptotic giant branch star yields employed in the simulation.

ccsne_yields [dataframe] The core-collapse supernova yields employed in the simulation.

sneia_yields [dataframe] The type Ia supernova yields employed in the simulation.

Note: Reinstancing functional yields and simulation parameters requires dill, an extension to `pickle` in the python standard library. It is recommand that VICE users install dill >= 0.2.0.

Tip: VICE outputs are stored in directories with a '.vice' extension following the name of the simulation. This allows users to run <command> *.vice in a terminal to run commands on all VICE outputs in a given directory.

Functions

- show (requires matplotlib >= 2.0.0)

See also:

- vice.history
- vice.mdf
- vice.multioutput

Example Code

```
>>> import vice
>>> out = vice.output("example")
>>> out.history[100]
        vice.dataframe{
                time -----------> 1.0
                mgas -----------> 5795119000.0
                mstar ----------> 2001106000.0
                sfr ------------> 2.897559
                ifr ------------> 9.1
                ofr ------------> 7.243899
                eta_0 ----------> 2.5
                r_eff ----------> 0.3534769
                z_in(fe) -------> 0.0
                z_in(sr) -------> 0.0
                z_in(o) --------> 0.0
                z_out(fe) ------> 0.0002769056
                z_out(sr) ------> 3.700754e-09
                z_out(o) -------> 0.001404602
                mass(fe) -------> 1604701.0
                mass(sr) -------> 21.44631
                mass(o) --------> 8139837.0
                z(fe) ----------> 0.0002769056166059748
                z(sr) ----------> 3.700754031107903e-09
                z(o) -----------> 0.0014046022178319376
                [fe/h] ---------> -0.6682579454664828
                [sr/h] ---------> -1.1074881208001155
                [o/h] ----------> -0.6098426789720387
                [sr/fe] --------> -0.43923017533363273
                [o/fe] ---------> 0.05841526649444406
                [o/sr] ---------> 0.4976454418280768
                z --------------> 0.0033582028978416337
                [m/h] ----------> -0.6200211036287412
                lookback -------> 9.0
        }
>>> out.mdf[60]
        vice.dataframe{
                bin_edge_left --> 0.0
                bin_edge_right -> 0.05
                dn/d[fe/h] -----> 0.0
                dn/d[sr/h] -----> 0.0
                dn/d[o/h] ------> 0.0
                dn/d[sr/fe] ----> 0.06001488
                dn/d[o/fe] -----> 0.4337209
                dn/d[o/sr] -----> 0.0
        }
```

vice.output.name

Type : `str`

The name of the simulation; this corresponds to the name of the '.vice' directory containing the output.

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.name
        'example'
```

vice.output.elements

Type : `tuple` of strings

The symbols of the elements whose enrichment was modeled to produce the output file, as they appear on the periodic table.

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.elements
        ('fe', 'sr', 'o')
```

vice.output.history

Type : `dataframe`

The dataframe read in via vice.history with the same name as this output.

See also:

vice.history

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.history["time"][100]
        1.0
>>> example.history
        vice.dataframe{
                time -----------> 1.0
                mgas -----------> 5795119000.0
                mstar ----------> 2001106000.0
```

(continues on next page)

(continued from previous page)

```
                sfr ------------> 2.897559
                ifr ------------> 9.1
                ofr ------------> 7.243899
                eta_0 ----------> 2.5
                r_eff ----------> 0.3534769
                z_in(fe) -------> 0.0
                z_in(sr) -------> 0.0
                z_in(o) --------> 0.0
                z_out(fe) ------> 0.0002769056
                z_out(sr) ------> 3.700754e-09
                z_out(o) -------> 0.001404602
                mass(fe) -------> 1604701.0
                mass(sr) -------> 21.44631
                mass(o) --------> 8139837.0
                z(fe) ----------> 0.0002769056166059748
                z(sr) ----------> 3.700754031107903e-09
                z(o) -----------> 0.0014046022178319376
                [fe/h] ---------> -0.6682579454664828
                [sr/h] ---------> -1.1074881208001155
                [o/h] ----------> -0.6098426789720387
                [sr/fe] --------> -0.43923017533363273
                [o/fe] ---------> 0.05841526649444406
                [o/sr] ---------> 0.4976454418280768
                z --------------> 0.0033582028978416337
                [m/h] ----------> -0.6200211036287412
                lookback -------> 9.0
        }
```

vice.output.mdf

Type : `dataframe`

The dataframe read in via vice.mdf with the same name as this output.

See also:

vice.mdf

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.mdf["bin_edge_left"][:10]
        [-3.0, -2.95, -2.9, -2.85, -2.8, -2.75, -2.7, -2.65, -2.6, -2.55]
>>> example.mdf[60]
        vice.dataframe{
                bin_edge_left --> 0.0
                bin_edge_right -> 0.05
                dn/d[fe/h] -----> 0.0
                dn/d[sr/h] -----> 0.0
                dn/d[o/h] ------> 0.0
```

(continues on next page)

(continued from previous page)

```
                dn/d[sr/fe] ----> 0.06001488
                dn/d[o/fe] -----> 0.4337209
                dn/d[o/sr] -----> 0.0
        }
```

vice.output.agb_yields

Type : `dataframe`

The asymptotic giant branch star yields employed in the simulation.

New in version 1.2.0: Prior to version 1.2.0, the `singlezone` object required an attribute `agb_model`, denoting which table of yields to adopt.

Note: This dataframe is not customizable.

See also:

vice.yields.agb.settings

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.agb_yields
        vice.dataframe{
                fe -------------> cristallo11
                o --------------> cristallo11
                sr -------------> cristallo11
        }
```

vice.output.ccsne_yields

Type : `dataframe`

The core-collapse supernova yields employed in the simulation.

Note: This dataframe is not customizable

See also:

vice.yields.ccsne.settings

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.ccsne_yields
        vice.dataframe{
                fe --------------> 0.000246
                o ---------------> 0.00564
                sr --------------> 1.34e-08
        }
```

vice.output.sneia_yields

Type : `dataframe`

The type Ia supernova yields employed in the simulation.

Note: This dataframe is not customizable.

See also:

vice.yields.sneia.settings

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.sneia_yields
        vice.dataframe{
                fe --------------> 0.00258
                o ---------------> 5.79e-05
                sr --------------> 0
        }
```

vice.output.show

Show a plot of the given quantity referenced by a keyword argument.

Signature: x.show(key, xlim = None, ylim = None)

Parameters

x [output] An instance of this class.

key [str [case-insensitive]] The keyword argument. If this is a quantity stored in the history attribute, it will be plotted against time by defult. Conversely, if it is stored in the mdf attribute, the corresponding stellar metallicity distribution function will be plotted.

Users can also specify an argument of the format "key1-key2" where key1 and key2 are elements of the history output. This will then plot key1 against key2.

xlim [array-like (contains real numbers) [default][None]] The x-limits to impose on the shown plot, if any.

ylim [array-like (contains real numbers) [default][None]] The y-limits to impose on the shown plot, if any.

Raises

- **KeyError**
  - Key is not found in either history or mdf attributes
- **ModuleNotFoundError**
  - Matplotlib version >= 2.0.x is not found in the user's system.

Note: In python 3.5.x, this will be an ImportError.

Other errors may be raised by matplotlib.pyplot.show.

Notes

This function is **NOT** intended to generate publication quality plots for users. It is included purely as a convenience function to allow "quick and dirty" data visualization and inspection of simulation outputs immediately with only one line of code.

Example Code

```
>>> import vice
>>> out = vice.output("example")
>>> out.show("dn/d[o/fe]")
>>> out.show("sfr")
>>> out.show("[o/fe]-[fe/h]")
```

vice.output.zip

Compress a VICE output into a zipfile.

Signature: vice.output.zip(name)

New in version 1.1.0.

Parameters

name [str or output] The full or relative path to an output, or the output object itself. The '.vice' extension is not required.

Raises

- **IOError**
  - Output is not found
  - Directory could not be interpreted as a VICE output.

Example Code

```
>>> import numpy as np
>>> import vice
>>> vice.singlezone(name = "example").run(np.linspace(0, 10, 1001))
>>> vice.output.zip("example")
```

vice.output.unzip

Decompress a VICE output from a zipfile.

Signature: vice.output.unzip(name)

New in version 1.1.0.

Parameters

name [str] The full or relative path to a compressed VICE output file. The '.vice.zip' extension is not required.

Raises

- **IOError**
  - Zipped file is not found.

Example Code

```
>>> import vice
>>> vice.output.unzip("example.vice.zip")
>>> out = vice.output("example")
```

vice.multioutput

Reads in the output from multizone simulations and allows the user to access it easily via dataframes.

Signature: vice.multioutput(name)

New in version 1.2.0.

Parameters

name [str] The full or relative path to the output directory. The '.vice' extension is not required.

Note: If `name` corresponds to output from the `singlezone` class, an `output` object is created instead.

Attributes

name [str] The full or relative path to the output directory.

zones [dataframe] A dataframe containing each zone's corresponding `output` object. The keys to this dataframe are the names of the `singlezone` objects contained in the `multizone` object which produced the output.

stars [dataframe] A dataframe containing all star particle data.

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.name
        "example"
>>> example.zones
        vice.dataframe{
                zone0 ----------> <VICE output from singlezone: example.vice/zone0>
                zone1 ----------> <VICE output from singlezone: example.vice/zone1>
                zone2 ----------> <VICE output from singlezone: example.vice/zone2>
                zone3 ----------> <VICE output from singlezone: example.vice/zone3>
                zone4 ----------> <VICE output from singlezone: example.vice/zone4>
                zone5 ----------> <VICE output from singlezone: example.vice/zone5>
                zone6 ----------> <VICE output from singlezone: example.vice/zone6>
                zone7 ----------> <VICE output from singlezone: example.vice/zone7>
                zone8 ----------> <VICE output from singlezone: example.vice/zone8>
                zone9 ----------> <VICE output from singlezone: example.vice/zone9>
        }
```

(continues on next page)

(continued from previous page)

```
>>> example.stars
        vice.dataframe{
                formation_time -> [0, 0, 0, ... , 10, 10, 10]
                zone_origin ----> [0, 1, 2, ... , 7, 8, 9]
                zone_final -----> [0, 1, 2, ... , 7, 8, 9]
                mass -----------> [3e+07, 3e+07, 3e+07, ... , 2.96329e+07, 2.96329e+07,␣
↪2.96329e+07]
                z(fe) ----------> [0, 0, 0, ... , 0.000846062, 0.000846062, 0.000846062]
                z(sr) ----------> [0, 0, 0, ... , 1.03519e-08, 1.03519e-08, 1.03519e-08]
                z(o) -----------> [0, 0, 0, ... , 0.0018231, 0.0018231, 0.0018231]
                [fe/h] ---------> [-inf, -inf, -inf, ... , -0.183187, -0.183187, -0.
↪183187]
                [sr/h] ---------> [-inf, -inf, -inf, ... , -0.660756, -0.660756, -0.
↪660756]
                [o/h] ----------> [-inf, -inf, -inf, ... , -0.496585, -0.496585, -0.
↪496585]
                [sr/fe] --------> [nan, nan, nan, ... , -0.477569, -0.477569, -0.477569]
                [o/fe] ---------> [nan, nan, nan, ... , -0.313397, -0.313397, -0.313397]
                [o/sr] ---------> [nan, nan, nan, ... , 0.164171, 0.164171, 0.164171]
                z --------------> [0, 0, 0, ... , 0.0053307, 0.0053307, 0.0053307]
                [m/h] ----------> [-inf, -inf, -inf, ... , -0.419344, -0.419344, -0.
↪419344]
                age ------------> [10, 10, 10, ... , 0, 0, 0]
        }
```

vice.multioutput.name

Type : `str`

The name of the ".vice" directory containing the output of a `multizone` object. The attributes of this object, its corresponding singlezone outputs, and their attributes, are all contained in this directory.

Example Code

```
>>> import vice
>>> example = vice.multioutput("example")
>>> example.name
        "example"
```

vice.multioutput.zones

Type : `dataframe`

The data for each simulated zone. The keys to this dataframe are the names of the `singlezone` objects contained in the `multizone` object which produced the output.

Example Code

```
>>> import vice
>>> example = vice.multioutput("example")
>>> example.zones
        vice.dataframe{
                zone0 ----------> <VICE output from singlezone: example.vice/zone0>
                zone1 ----------> <VICE output from singlezone: example.vice/zone1>
                zone2 ----------> <VICE output from singlezone: example.vice/zone2>
                zone3 ----------> <VICE output from singlezone: example.vice/zone3>
                zone4 ----------> <VICE output from singlezone: example.vice/zone4>
                zone5 ----------> <VICE output from singlezone: example.vice/zone5>
                zone6 ----------> <VICE output from singlezone: example.vice/zone6>
                zone7 ----------> <VICE output from singlezone: example.vice/zone7>
                zone8 ----------> <VICE output from singlezone: example.vice/zone8>
                zone9 ----------> <VICE output from singlezone: example.vice/zone9>
        }
>>> example.zones["zone0"].name
        "example.vice/zone0"
```

vice.multioutput.stars

Type : `dataframe`

The data for the star particles of this simulation. This stores the formation time in Gyr of each particle, its mass, its formation and final zone numbers, and the metallicity by mass of each element in the simulation.

Example Code

```
>>> import vice
>>> example = vice.multioutput("example")
>>> example.stars
        vice.dataframe{
                formation_time -> [0, 0, 0, ... , 10, 10, 10]
                zone_origin ----> [0, 1, 2, ... , 7, 8, 9]
                zone_final -----> [0, 1, 2, ... , 7, 8, 9]
                mass -----------> [3e+07, 3e+07, 3e+07, ... , 2.96329e+07, 2.96329e+07,␣
↪2.96329e+07]
                z(fe) ----------> [0, 0, 0, ... , 0.000846062, 0.000846062, 0.000846062]
                z(sr) ----------> [0, 0, 0, ... , 1.03519e-08, 1.03519e-08, 1.03519e-08]
                z(o) -----------> [0, 0, 0, ... , 0.0018231, 0.0018231, 0.0018231]
                [fe/h] ---------> [-inf, -inf, -inf, ... , -0.183187, -0.183187, -0.
↪183187]
                [sr/h] ---------> [-inf, -inf, -inf, ... , -0.660756, -0.660756, -0.
↪660756]
                [o/h] ----------> [-inf, -inf, -inf, ... , -0.496585, -0.496585, -0.
↪496585]
                [sr/fe] --------> [nan, nan, nan, ... , -0.477569, -0.477569, -0.477569]
                [o/fe] ---------> [nan, nan, nan, ... , -0.313397, -0.313397, -0.313397]
                [o/sr] ---------> [nan, nan, nan, ... , 0.164171, 0.164171, 0.164171]
```

(continues on next page)

(continued from previous page)

```
                z --------------> [0, 0, 0, ... , 0.0053307, 0.0053307, 0.0053307]
                [m/h] ----------> [-inf, -inf, -inf, ... , -0.419344, -0.419344, -0.
↪419344]
                age ------------> [10, 10, 10, ... , 0, 0, 0]
        }
```

vice.stars

Read in the star particles from a multizone simulation output.

Signature: vice.stars(name)

New in version 1.2.0.

Parameters

name [str] The full or relative path to the output directory. The '.vice' extension is not required.

Returns

stars [tracers [VICE dataframe derived class]] A subclass of the VICE dataframe designed to store the star particles and to calculate relevant quantities automatically upon indexing.

Raises

- **IOError [Only occurs if the output has been altered]**
  - Output directory not found.
  - Output files not formatted correctly.
  - Other VICE output files are missing from the output.

See also:

vice.core.dataframe.tracers

Example Code

```
>>> import vice
>>> stars = vice.stars(example")
>>> stars[100]
        vice.dataframe{
                formation_time -> 0.1
                zone_origin ----> 0.0
                zone_final -----> 0.0
                mass -----------> 29695920.0
                z(fe) ----------> 1.128362e-05
                z(sr) ----------> 6.203682e-10
                z(o) -----------> 0.0002587532
```

(continues on next page)

(continued from previous page)

```
            [fe/h] ---------> -2.058141258363775
            [sr/h] ---------> -1.8831288138453521
            [o/h] ----------> -1.3445102993763647
            [sr/fe] --------> 0.17501244451842268
            [o/fe] ---------> 0.7136309589874101
            [o/sr] ---------> 0.5386185144689875
            z ---------------> 0.0005393007991864363
            [m/h] ----------> -1.4142969718113587
            age ------------> 9.9
    }
```

vice.mirror

[DEPRECATED]

Obtain an instance of either vice.singlezone or vice.multizone class given only an instance of the vice.output class or the path to the output. The returned object will have the same parameters as that which produced the output, allowing re-simulation with whatever modifications the user desires.

Signature: vice.mirror(arg)

Deprecated since version 1.1.0: Users should instead call vice.singlezone.from_output or vice.multizone.from_output to achieve this functionality.

Parameters

arg [`str` or `output`] Either the path to the output (type `str`) or the output object itself.

Returns

obj [`singlezone` or `multizone`] If arg is of type `output`, then the `singlezone` object which produced the output is returned. If arg is of type `str`, then obj is either of type vice.singlezone or vice.multizone, depending on which type of simulation produced the output. If arg is of type `multioutput`, then the corresponding `multizone` object is returned.

Raises

- **ImportError**
  - The output has encoded functional attributes and the user does not have dill installed.
- **UserWarning**
  - The output was produced with functional attributes, but was ran on a system without dill, and they have thus been lost.

Note: Saving and reinstancing functional simulation parameters from VICE outputs requires dill, an extenstion to `pickle` in the python standard library. It is recommended that VICE users install dill >= 0.2.0.

Example Code

```
>>> out = vice.output("example")
>>> new = vice.mirror(out)
>>> new
vice.singlezone{
        name ------------> onezonemodel
        func ------------> <function _DEFAULT_FUNC_ at 0x1085a6ae8>
        mode ------------> ifr
        verbose ---------> False
        elements --------> ('fe', 'sr', 'o')
        IMF -------------> kroupa
        eta -------------> 2.5
        enhancement -----> 1.0
        Zin -------------> 0.0
        recycling -------> continuous
        delay -----------> 0.15
        RIa -------------> plaw
        Mg0 -------------> 6000000000.0
        smoothing -------> 0.0
        tau_ia ----------> 1.5
        tau_star --------> 2.0
        schmidt ---------> False
        schmidt_index ---> 0.5
        MgSchmidt -------> 6000000000.0
        dt --------------> 0.01
        m_upper ---------> 100.0
        m_lower ---------> 0.08
        Z_solar ---------> 0.014
        bins ------------> [-3, -2.95, -2.9, ... , 0.9, 0.95, 1]
}
>>> import numpy as np
>>> new.run(np.linspace(0, 10, 1001))
```

vice.toolkit

VICE Toolkit : General utilities to maximize VICE's computational power and user-friendliness.

New in version 1.2.0.

Contents

hydrodisk [<module>] Utilities for simulating migration in disk galaxies.

interpolation [<module>] Interpolation schema.

J21_sf_law [<module>] The observationally motivated star formation law from Johnson et al. (2021)[1], intended for use as the attribute `tau_star` of the `singlezone` class.

[1] Johnson et al. (2021), MNRAS, 508, 4484

vice.toolkit.hydrodisk

Built-in stellar radial migration schema for disk galaxies inspired by hydrodynamical simulations.

New in version 1.2.0.

Contents

hydrodiskstars [object] A stellar migration scheme informed by the h277 simulation, a zoom-in hydrodynamic simulation of a Milky Way like galaxy ran from cosmological initial conditions (a part of the g14 simulation suite, Christensen et al 2012[1]). Stellar migrations can migrate according to a handful of assumptions about the time dependence of their orbital radius between birth and the end of the simulation. For discussion, see section 2 of Johnson et al. (2021)[2].

data [module] Manages VICE's supplementary data containing the h277 star particle subsamples. Executes a download upon first creation of a hydrodiskstars object.

vice.toolkit.hydrodisk.hydrodiskstars

A stellar migration scheme informed by the h277 simulation, a zoom-in hydrodynamic simulation of a Milky Way like galaxy ran from cosmological initial conditions (a part of the g14 simulation suite, Christensen et al 2012[1]).

Signature: vice.toolkit.hydrodisk.hydrodiskstars(radial_bins, N = 1e5, mode = "diffusion")

New in version 1.2.0.

Warning: Simulations which adopt this model that run for longer than 13.2 Gyr are not supported. Stellar populations in the built-in hydrodynamical simulation data span 13.2 Gyr of ages; simulations on longer timescales are highly likely to produce a segmentation fault.

Parameters

radial_bins [array-like [elements must be positive real numbers]] The bins in galactocentric radius in kpc describing the disk model. This must extend from 0 to at least 20. Need not be sorted in any way. Will be stored as an attribute.

N [int [default][1e5]] An approximate number of star particles from the hydrodynamical simulation to include in the sample of candidate analogs. Their data are not stored in a single file, but split across random subsamples to decrease computational overhead when the full sample is not required. In practice, this number should be slightly larger than the number of (relevant) stellar populations simulated by a multizone model.

Note: There are 3,102,519 star particles available for this object. Any more stellar populations than this would oversample these data.

---

mode [str [case-insensitive] or None [default]["diffusion"]] The attribute 'mode', initialized via keyword argument.

[1] Christensen et al. (2012), MNRAS, 425, 3058
[2] Johnson et al. (2021), MNRAS, 508, 4484
[1] Christensen et al. (2012), MNRAS, 425, 3058

Attributes

radial_bins [list] The bins in galactocentric radius in kpc describing the disk model.

analog_data [dataframe] The raw star particle data from the hydrodynamical simulation.

analog_index [int] The index of the star particle acting as the current analog. -1 if the analog has not yet been set (see note below under *Calling*).

mode [str or `None`] The mode of stellar migration, describing the approximation of how stars move from birth to final radii. Either "diffusion", "sudden", or "linear". See property docstring for more details.

Note: Only subclasses may set this attribute `None`, in which case it is assumed that a custom migration approximation is employed by an overridden `__call__` function. In this case, if this attribute is not set to `None`, multizone simulations will *still* use the approximation denoted by this property.

Calling

As all stellar migration prescriptions must, this object can be called with three parameters, in the following order:

zone [int] The zone index of formation of the stellar population. Must be non-negative.

tform [float] The time of formation of the stellar population in Gyr.

time [float] The simulation time in Gyr (i.e. not the age of the star particle).

Note: The search for analog star particles is ran when the formation time and simulation time are equal. Therefore, calling this object with the second and third parameters equal resets the star particle acting as the analog, and the data for the corresponding star particle can then be accessed via the attribute `analog_index`.

Functions

decomp_filter [[instancemethod]] Filter the star particles based on their kinematic decomposition.

Raises

- **ValueError**
  - Minimum radius does not equal zero
  - Maximum radius < 20
- **ScienceWarning**
  - This object is called with a time larger than 13.2 Gyr
  - The number of analog star particles requested is larger than the number available from the hydrodynamical simulation (3,102,519)

Notes

This object requires VICE's supplementary data sample, available in its GitHub repository at ./vice/toolkit/hydrodisk/data. The first time a `hydrodiskstars` object is constructed, VICE will download the additional data automatically. If this process fails, it may be due to not having administrator's privileges on your system; in this event, users should speak with their administrator, who would then be able to download their data by running the following on their system:

```
>>> import vice
>>> vice.toolkit.hydrodisk.data.download()
```

This migration scheme works by assigning each stellar population in the simulation an analog star particle from the hydrodynamical simulation. The analog is randomly drawn from a sample of star particles which formed at a similar radius and time, and the stellar population then assumes the change in orbital radius of its analog.

VICE first searches for analogs in the `h277` data for star particles which formed at a radius of $R\pm$ 250 pc and at a time of $T\pm$ 250 Myr. If no analogs are found that satisfy this requirement, the search is widened to $R\pm$ 500 pc and $T\pm$ 500 Myr. If still no analog is found, then the time restriction of $T\pm$ 500 Myr is maintained, and VICE finds the star particle with the smallest difference in birth radius, assigning it as the analog. These values parameterizing this search algorithm are declared in `vice/src/toolkit/hydrodiskstars.h` in the VICE source tree. For further details, see "Milky Way-Like Galaxies" under VICE's science documentation.

This object can be subclassed to implement a customized migration approximation by overriding the `__call__` function. However, in this case, users must also set the attribute `mode` to `None`. If this requirement is not satisfied, multizone simulations will **still** use the approximation denoted by the `mode` attribute, **not** their overridden `__call__` function.

The `h277` galaxy had a weak and transient bar, but does not have one at the present day. This is one notable difference between it and the Milky Way.

Example Code

```
>>> from vice.toolkit.hydrodisk import hydrodiskstars
>>> import numpy as np
>>> example = hydrodiskstars(np.linspace(0, 20, 81), N = 5e5)
>>> example.radial_bins
[0.0,
 0.25,
 0.5,
 ...
 19.5,
 19.75,
 20.0]
>>> example.analog_data.keys()
['id', 'tform', 'rform', 'rfinal', 'zfinal', 'vrad', 'vphi', 'vz']
>>> example.analog_index
-1
>>> example(5, 7.2, 7.2)
5
>>> example.analog_index
200672
>>> example.analog_data["vrad"][example.analog_index]
5.6577
>>> example.mode
"diffusion"
```

vice.toolkit.hydrodisk.hydrodiskstars.radial_bins

Type : list [elements are positive real numbers]

The bins in galactocentric radius in kpc describing the disk model. Must extend from 0 to at least 20 kpc. Need not be sorted in any way when assigned.

Example Code

```
>>> from vice.toolkit.hydrodisk import hydrodiskstars
>>> import numpy as np
>>> example = hydrodiskstars([0, 5, 10, 15, 20])
>>> example.radial_bins
[0, 5, 10, 15, 20]
>>> example.radial_bins = list(range(31))
>>> example.radial_bins
[0,
 1,
 2,
 ...
 17,
 18,
 19,
 20]
```

vice.toolkit.hydrodisk.hydrodiskstars.analog_data

Type : dataframe

The star particle data from the hydrodynamical simulation. The following keys map to the following data:

- id: The IDs of each star particle
- tform: The time the star particle formed in Gyr
- rform: The radius the star particle formed at in kpc
- rfinal: The radius the star particle ended up at in kpc
- zform: The disk midplane distance in kpc at the time of formation.
- zfinal: The disk midplane distance in kpc at the end of the simulation
- vrad: The radial velocity of the star particle at the end of the simulation in km/sec
- vphi: The azimuthal velocity of the star particle at the end of the simulation in km/sec
- vz: The velocity perpendicular to the disk midplane at the end of the simulation in km/sec
- decomp: An integer denoting which kinematic subclass the star particle belongs to (1: thin disk, 2: thick disk, 3: bulge, 4: pseudobulge, 5: halo).

Example Code

```
>>> from vice.toolkit.hydrodisk import hydrodiskstars
>>> import numpy as np
>>> example = hydrodiskstars(np.linspace(0, 20, 81))
>>> example.analog_data.keys()
['id', 'tform', 'rform', 'rfinal', 'zfinal', 'vrad', 'vphi', 'vz']
>>> example.analog_data["rfinal"][:10]
[2.0804,
 14.9953,
 2.2718,
 15.1236,
 2.3763,
 0.9242,
 9.0908,
 0.1749,
 8.415,
 20.1452]
```

vice.toolkit.hydrodisk.hydrodiskstars.analog_index

Type : int

The index of the analog in the hydrodynamical simulation star particle data. -1 if it has not yet been assigned.

Note: Calling this object at a given zone with the formation time and the simulation time equal resets the star particle acting as the analog.

Example Code

```
>>> from vice.toolkit.hydrodisk import hydrodiskstars
>>> import numpy as np
>>> example = hydrodiskstars(np.linspace(0, 20, 81))
>>> example.analog_index
-1 # no analog yet
>>> example(2, 1, 1) # final two arguments equal resets analog
15745
>>> example(10, 4, 4)
10
>>> example.analog_index
101206
>>> example.analog_data["rfinal"][example.analog_index]
2.6411
>>> example.analog_data["vrad"][example.analog_data]
92.2085
```

vice.toolkit.hydrodisk.hydrodiskstars.mode

Type : str [case-insensitive] or None

Default : "diffusion"

Recognized Values

The following is a breakdown of how stellar populations migrate in multizone simulations under each approximation.

- **"diffusion"** The orbital radius at times between birth and 13.2 Gyr are assigned via a sqrt(time) dependence, the mean displacement if stars migrated according to a random walk. Stellar populations spiral inward or outward with a smooth time dependence in this model.
- **"linear"** Orbital radii at times between birth and 13.2 Gyr are assigned via linear interpolation. Stellar populations therefore spiral uniformly inward or outward from birth to final radii, with orbital radius changing more slowly for young stars than in the diffusion model, but more quickly for old stars.
- **"sudden"** The time of migration is randomly drawn from a uniform distribution between when a stellar population is born and 13.2 Gyr. At times prior to this, it is at its radius of birth; at subsequent times, it is at its final radius. Stellar populations therefore spend no time at intermediate radii.
- **None** Only supported for subclasses of the hydrodiskstars object, this should be used when the user intends to override the built-in migration assumptions and implement a different one. In these cases, the `__call__` method should be overridden in addition to this attribute being set to None.

Note: The "post-processing" model from Johnson et al. (2021)[1] corresponds to setting the attribute `simple = True` in the `milkyway` object, inherited from the base class `multizone`.

Example Code

```
>>> from vice.toolkit.hydrodisk import hydrodiskstars
>>> import numpy as np
>>> example = hydrodiskstars(np.linspace(0, 20, 81))
>>> example.mode
'diffusion'
>>> example.mode = "sudden"
>>> example.mode = "linear"
```

vice.toolkit.hydrodisk.hydrodiskstars.decomp_filter

Filter the star particles from the hydrodynamic simulation based on the kinematic decomposition.

[1] Johnson et al. (2021), 2103.09838

Parameters

values [`list` [elements of type `int`]] The integer values of the "decomp" column in the `analog_data` attribute to base the filter on. Those with a decomposition tag equal to one of the values in this list will pass the filter and remain in the sample.

Note: The integer values mean that an individual star particle has kinematics associated with the following sub-populations:

- 1: thin disk
- 2: thick disk
- 3: bulge
- 4: pseudobulge
- 5: halo

Example Code

```
>>> from vice.toolkit.hydrodisk import hydrodiskstars
>>> import numpy as np
>>> example = hydrodiskstars(np.linspace(0, 20, 81))
>>> len(example.analog_data['id'])
102857
>>> example.decomp_filter([1, 2]) # disk stars only
>>> len(example.analog_data['id'])
57915
>>> all([i in [1, 2] for i in example.analog_data['decomp']])
True
>>> example = hydrodiskstars(np.linspace(0, 20, 81))
>>> len(example.analog_data['id'])
102857
>>> example.decomp_filter([3, 4]) # bulge stars only
>>> len(example.analog_data['id'])
44942
>>> all([i in [3, 4] for i in example.analog_data['decomp']])
True
```

vice.toolkit.interpolation

VICE Interpolation Schema : Internal utilities for interpolation.

New in version 1.2.0.

Contents

interp_scheme_1d [object] A 1-D linear interpolation scheme given a list of (x, y) points.

interp_scheme_2d [object] A 2-D linear interpolation scheme given lists of x- and y-coordinates and a 2-D list of z-coordinates.

vice.toolkit.interpolation.interp_scheme_1d

A 1-dimensional interpolation scheme. This object takes in x-coordinates and y-coordinates of the same length and constructs a continuous function by connecting the points with straight lines.

Signature: vice.toolkit.interpolation.interp_scheme_1d(xcoords, ycoords)

New in version 1.2.0.

Parameters

xcoords [array-like] The attribute `xcoords`. See below.

ycoords [array-like] The attribute `ycoords`. See below.

Attributes

xcoords [`list` [elements are real numbers]] The x-coordinates of the points to construct the interpolation scheme out of, in arbitrary units.

Note: These values will be automatically sorted from least to greatest upon construction of an `interp_scheme_1d` object. While this could potentially alter the ordering of this attribute, it will not affect the `ycoords` attribute, which is assumed to correspond component-wise to the x-coordinates in their least to greatest ordering.

---

ycoords [`list` [elements are real numbers]] The y-coordinates of the points to construct the interpolation scheme out of, in arbitrary units.

Calling

Call this object with a given x-coordinate, and it will automatically determine the correct pair of (x, y) coordinates to interpolate from, and return the appropriate value.

Parameters:

- **x** [real number] The x-coordinate to evaluate the interpolation scheme at, in the same units as the attribute `xcoords`.

Returns:

- **y** [real number] The value of the y-coordinate, approximated via the line connecting the two points (x_1, y_1) and (x_2, y_2) such that $x_1 \leq x \leq x_2$. If `x` is less than the smallest x-coordinate or larger than the largest one, the result will be determined via linear extrapolation using either the two smallest or two largest elements of the `xcoords` attribute.

Indexing

Index this object as you would an array-like object, and it will return the (x, y) coordinates of the sampled points from the attributes `xcoords` and `ycoords`.

Example Code

```
>>> from vice.toolkit.interpolation import interp_scheme_1d
>>> example = interp_scheme_1d([1, 2, 3], [2, 4, 6])
>>> example(0)
0.0
>>> example(5)
10.0
>>> example(4)
8.0
>>> example[:]
[[1.0, 2.0], [2.0, 4.0], [3.0, 6.0]]
>>> example.xcoords
[1.0, 2.0, 3.0]
>>> example.ycoords
[2.0, 4.0, 6.0]
>>> example.n_points
3
```

vice.toolkit.interpolation.interp_scheme_1d.xcoords

Type : `list` [elements are real numbers]

The x-coordinates on which the function is sampled.

vice.toolkit.interpolation.interp_scheme_1d.ycoords

Type : `list` [elements are real numbers]

The y-coordinates on which the function is sampled.

vice.toolkit.interpolation.interp_scheme_1d.n_points

Type : `int`

The number of (x, y) points on which the function is sampled.

vice.toolkit.interpolation.interp_scheme_2d

A 2-dimensional interpolation scheme. This object takes in x-, y-, and z-coordinates of the appropriate lengths and constructs a continuous function by connecting the points with bilinear interpolation.

Signature: vice.toolkit.interpolation.interp_scheme_2d(xcoords, ycoords, zcoords)

New in version 1.2.0.

Parameters

xccords [array-like] The attribute `xcoords`. See below.

ycoords [array-like] The attribute `ycoords`. See below.

zcoords [array-like] The attribute `zcoords`. See below.

Attributes

xcoords [`list` [elements are real numbers]] The x-coordinates of the points to construct the interpolation scheme out of, in arbitrary units.

Note: These values will be automatically sorted from least to greatest upon construction of an `interp_scheme_2d` object. While this could potentially alter the ordering of this attribute, it will not affect the `zcoords` attribute, which is assumed to correspond component-wise to the x-coordinates in their least to greatest sorting. The burden is on the user to ensure that their coordinates are in the proper ordering.

---

ycoords [`list` [elements are real numbers]] The y-coordinates fo the points to construct the interpolation scheme out of, in arbitrary units.

Note: The same note which applies to the x-coordinates above also applies to the y-coordinates.

---

zcoords [`list` [elements are of type `list` containing real numbers]] The z-coordinates of the points to construct the interpolation scheme out of, in arbitrary units. This must be of the same length as the `xcoords` array, containing elements which are of the same length as the `ycoords` array. The values stored should correspond component-wise to those arrays such that `self(xcoords[i], ycoords[j]) = zcoords[i][j]`.

Calling

Call this object with any given x- and y-coordinates, and it will automatically determine the correct set of (x, y) coordinates to interpolate between, and return the appropriate value.

Parameters:

- **x** [real number] The x-coordinate to evaluate the interpolation scheem at, in the same units as the attribute `xcoords`.
- **y** [real number] The y-coordinate to evaluate the interpolation scheme at, in the same units as the attribute `ycoords`.

Returns:

- **z** [real number] The value of the z-coordinate, approximated via bilinear interpolation connecting the points (x_1, y_1), (x_1, y_2), (x_2, y_1), and (x_2, y_2): the points defining the four corners of the box in x-y space bouding the point (**x**, **y**).

  The interpolation is such that the values of $f(x_1, y)$ and $f(x_2, y)$ are determined via linear interpolation in one-dimension at constant x, then the value of $f(x, y)$ is calculated similarly at constant y.

Example Code

```
>>> from vice.toolkit.interpolation import interp_scheme_2d
>>> example = interp_scheme_2d([1, 2, 3], [2, 4, 6],
        [[3, 6, 9], [4, 8, 12], [5, 10, 15]])
>>> example(0, 0)
0.0
>>> example(10, 10)
60.0
>>> example(3.1, 2.8)
7.140000000000001
>>> example.xcoords
[1, 2, 3]
>>> example.ycoords
[2, 4, 6]
>>> example.zcoords
[[3, 6, 9], [4, 8, 12], [5, 10, 15]]
```

vice.toolkit.interpolation.interp_scheme_2d.xcoords

Type : `list` [elements are real numbers]

The x-coordinates on which the function is sampled.

vice.toolkit.interpolation.interp_scheme_2d.ycoords

Type : `list` [elements are real numbers]

The y-coordinates on which the function is sampled.

vice.toolkit.interpolation.interp_scheme_2d.zcoords

Type : `list` [elements are of type `list`, storing real numbers]

The z-coordinates on which the function is sampled. A 2-D list, this attribute stores a value at each of the (x, y) coordinates on which a z-coordinate is known. The first axis corresponds to the x-coordinate, and the second the y-coordinate.

vice.toolkit.interpolation.interp_scheme_2d.n_x_values

Type : `int`

The number of x-coordinates on which the function is sampled. For each x-coordinate, there are `n_y_values` y-coordinates at which the z-coordinate is known.

vice.toolkit.interpolation.interp_scheme_2d.n_y_values

Type : `int`

The number of y-coordinates on which the function is sampled. For each y-coordinate, there are `n_x_values` x-coordinates at which the z-coordinate is known.

vice.toolkit.J21_sf_law

The default star formation law of the `milkyway` object. This is a callable object, accepting simulation time and either star formation rate or gas supply as second arguments. By default, it implements the star formation law adopted in Johnson et al. (2021)[1].

Signature: vice.toolkit.J21_sf_law(area, **kwargs)

New in version 1.2.0.

Warning: In a `milkyway` object, every zone has an instance of this class as its `tau_star` attribute. Any modifications to the attributes of this class should be made for **every zone**; if this is not ensured, the star formation law will not be consistent across all zones.

Parameters

area [real number] The value of the attribute `area`. See below.

****kwargs** [varying types] Other attributes can have their values set via keyword. See below.

Attributes

area [real number] The surface area in kpc^2 of the corresponding annulus in a `milkyway` model.

present_day_molecular [real number [default][2.0]] The depletion time of molecular gas at the present day in Gyr. Positive definite.

molecular_index [real number [default][0.5]] The power-law index on the time-dependence.

Sigma_g1 [real number [default][5.0e+06]] The smaller of the two surface densities of gas at which there is a break in the Kennicutt-Schmidt relation. Assumes units of $M_\odot kpc^{-2}$.

Sigma_g2 [real number [default][2.0e+07]] The larger of the two surface densities of gas at which there is a break in the Kennicutt-Schmidt relation. Assumes units of $M_\odot kpc^{-2}$.

index1 [real number [default][1.7]] The index of the power-law at gas surface densities below `Sigma_g1`.

[1] Johnson et al. (2021), MNRAS, 508, 4484

index2 [real number [default][3.6]] The index of the power-law at gas surface densities between `Sigma_g1` and `Sigma_g2`, above which it is assumed to be linear.

mode [str ["sfr", "ifr", or "gas"] [default]["ifr"]] The mode of the `milkyway` object.

Calling

Calling this object will calculate the star formation efficiency timescale $\tau_\star$ according to the parameters of the star formation law entered as attributes of this object. As in the `singlezone` object, this timescale is the gas supply per unit star formation in Gyr.

Parameters:

- **time** [real number] Simulation time in Gyr. Postive definite.
- **arg2** [real number] Either the gas supply in $M_\odot$ or the star formation rate in $M_\odot Gyr^{-1}$. Will be called by VICE directly. With the attribute `area`, the surface density of the corresponding quantity is known. Positive definite.

Returns:

- **tau_star** [real number] The star formation efficiency timescale given that simulation time and star formation rate/gas supply, in Gyr (as necessary).

See also:

vice.milkyway

Notes

This object encodes the parameters of the desired Kennicutt-Schmidt relation into the `milkyway` object. This is the relationship relating the surface densities of star formation $\dot{\Sigma}_\star$ and gas Σ_{gas}.

This object implements a star formation law in the `milkyway` object defined according to:

$$\dot{\Sigma}_\star \sim \Sigma_{\text{gas}}^N$$

The value of the power-law index N has two breaks, at $\Sigma_{\text{gas},1}$ and $\Sigma_{\text{gas},2}$.

$$\begin{aligned} N &= 1.0 \ (\Sigma_{\text{gas}} \geq \Sigma_{\text{gas},2}) \\ N &= \gamma_2 \ (\Sigma_{\text{gas},1} \leq \Sigma_{\text{gas}} \leq \Sigma_{\text{gas},2}) \\ N &= \gamma_1 \ (\Sigma_{\text{gas}} \leq \Sigma_{\text{gas},1}) \end{aligned}$$

The values γ_1 and γ_2 correspond to the attributes `index1` and `index2`, respectively. As their names would suggest, $\Sigma_{\text{gas},1}$ and $\Sigma_{\text{gas},2}$ correspond to `Sigma_g1` and `Sigma_g2`.

The depletion time of molecular gas due to star formation τ_{mol} is defined according to the following scaling:

$$\tau_{\text{mol}} = \tau_{\text{mol},0} \ (t/t_0)^\beta$$

where t_0 is the age of the universe today, t is the age of the universe at some simulation time, and $\tau_{\text{mol},0}$ is τ_{mol} at the present day. Because the `milkyway` model only supports lookback times up to 13.2 Gyr, the relation between age of the universe and simulation time is a simple linear translation:

$$t = t_{\text{sim}} + 0.5 \text{ Gyr}$$

with the assumption that t_0 = 13.7 Gyr is the age of the universe at the present day.

vice.toolkit.J21_sf_law.area

Type : float

The surface area of the star forming region in kpc^2. In the `milkway` object, this is an annulus for which calculation of the area is trivial.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.area
0.7853981633974483
>>> mw.zones[10].tau_star.area
16.493361431346415
```

vice.toolkit.J21_sf_law.molecular

Calculate the star formation efficiency timescale of molecular gas at a given simulation time.

Signature: x.molecular(time)

Parameters

x [`J21_sf_law`] An instance of this class.

time [`float`] The simulation time in Gyr.

Returns

tau_star_mol [float] The star formation efficiency timescale $\tau_\star$ for molecular gas only ($\tau_{\rm mol}$), defined accordingly:

$$\tau_{\rm mol} = \tau_{\rm mol,0}\left(\frac{0.5 + t}{t_0}\right)^\gamma$$

where $\tau_{\rm mol,0}$ is the value of $\tau_{\rm mol}$ at the present day, t_0 is the present-day age of the universe (assumed to be 13.7 Gyr), γ is the power-law index, and t is the parameter `time` passed to this function. A value of 0.5 is added, beacuse the onset of star formation is assumed to occur 0.5 Gyr following the big bang in the Johnson et al. (2021) models[1].

The values of $\tau_{\rm mol,0}$ and γ are controlled by the attributes `present_day_molecular` and `molecular_index`, respectively.

[1] Johnson et al. (2021), MNRAS, 508, 4484

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.molecular(0)
0.6617825960083583
>>> mw.zones[0].tau_star.molecular(5)
1.3776103291490314
>>> mw.zones[0].tau_star.molecular(13.2)
2.0
```

vice.toolkit.J21_sf_law.present_day_molecular

Type : float

Default : 2

The star formation efficiency timescale of molecular gas at the present day, in Gyr. Scales with time according to a power-law whose index is given by teh value of the attribute `molecular_index`. Must be positive.

Note: In the interstellar medium and star formation literature, this quantity is more often referred to as the depletion time due to star formation. Default value is chosen based on the results of Leroy et al. (2008)[1].

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.present_day_molecular
2.0
>>> for i in range(mw.n_zones):
>>>     mw.zones[i].tau_star.present_day_molecular = 1.0
>>> mw.zones[0].tau_star.present_day_molecular
1.0
```

vice.toolkit.J21_sf_law.molecular_index

Type : float

Default : 0.5

The power-law index on the time-dependence of the molecular gas star formation efficiency timescale. The normalization will be set such that the value at t = 12 Gyr is equal to the value of the attribute `present_day_molecular`.

Note: The default value is chosen based on the redshift dependence of $\tau_{\rm mol}$ reported by Tacconi et al. (2018)[1] and a redshift-time relation accurate for $z \lesssim 3$. See discussion in section 2 of Johnson et al. (2021)[2].

---

[1] Leroy et al. (2008), AJ, 136, 2782
[1] Tacconi et al. (2018), ApJ, 853, 179
[2] Johnson et al. (2021), MNRAS, 508, 4484

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.molecular_index
0.5
>>> for i in range(mw.n_zones):
>>>     mw.zones[i].tau_star.molecular_index = 0.6
>>> mw.zones[0].tau_star.molecular_index
0.6
```

vice.toolkit.J21_sf_law.Sigma_g1

Type : float

Default : 5.0e+06

The lower of the two surface densities of gas at which there is a break in the $\dot{\Sigma}_\star - \Sigma_{\text{gas}}$ relation. Below this value, the relation scales as a power-law with index set by the attribute `index1`. Assumes units of $M_\odot kpc^{-2}$.

Note: The value of this attribute should be smaller than that of the attribute `Sigma_g2`, the larger of the two surface densities of gas. Default values are chosen based on the aggregate data from Bigiel et al. (2010)[1] and Leroy et al. (2013)[2] presented in Krumholz et al. (2018)[3].

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.Sigma_g1
5.0e+06
>>> for i in range(mw.n_zones):
>>>     mw.zones[i].tau_star.Sigma_g1 = 6.0e+06
>>> mw.zones[0].tau_star.Sigma_g1
6.0e+06
```

vice.toolkit.J21_sf_law.Sigma_g2

Type : float

Default : 2.0e+07

The larger of the two surface densities of gas at which there is a break in the $\dot{\Sigma}_\star - \Sigma_{\text{gas}}$ relation. Below this value, the relation scales as a power-law with index set by the attribute `index2`, and above it, it is assumed to be linear. Assumes units of $M_\odot kpc^{-2}$.

[1] Bigiel et al. (2010), AJ, 140, 1194
[2] Leroy et al. (2013), AJ, 146, 19
[3] Krumholz et al. (2018), MNRAS, 477, 2716

Note: The value of this attribute should be larger than that of the attribute `Sigma_g1`, the smaller of the two surface densities of gas. Default values are chosen based on the aggregate data from Bigiel et al. (2010)[1] and Leroy et al. (2013)[2] presented in Krumholz et al (2018)[3].

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.Sigma_g2
2.0e+07
>>> for i in range(mw.n_zones):
>>>     mw.zones[i].tau_star.Sigma_g2 = 1.5e+07
>>> mw.zones[0].tau_star.Sigma_g2
1.5e+07
```

vice.toolkit.J21_sf_law.index1

Type : float

Default : 1.7

The power-law index of the $\dot{\Sigma}_\star - \Sigma_{\text{gas}}$ relation at gas surface densities $\Sigma_{\text{gas}} \leq \Sigma_{\text{gas},1}$, where $\Sigma_{\text{gas},1}$ is the value of the attribute `Sigma_g1`.

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.index1
1.7
>>> for i in range(mw.n_zones):
>>>     mw.zones[i].tau_star.index1 = 1.5
>>> mw.zones[0].tau_star.index1
1.5
```

vice.toolkit.J21_sf_law.index2

Type : float

Default : 3.6

The power-law index of the $\dot{\Sigma}_\star - \Sigma_{\text{gas}}$ relation at gas surface densities Σ_{gas} between $\Sigma_{\text{gas},1}$ and $\Sigma_{\text{gas},2}$, the values of the attributes `Sigma_g1` and `Sigma_g2`, respectively. At $\Sigma_{\text{gas}} \geq \Sigma_{\text{gas},2}$, the relation is assumed to be linear.

[1] Bigiel et al. (2010), AJ, 140, 1194
[2] Leroy et al. (2013), AJ, 146, 19
[3] Krumholz et al. (2018), MNRAS, 477, 2716

Example Code

```
>>> import vice
>>> mw = vice.milkyway(name = "example")
>>> mw.zones[0].tau_star.index2
3.6
>>> for i in range(mw.n_zones):
>>>     mw.zones[i].tau_star.index2 = 3.4
>>> mw.zones[0].tau_star.index2
3.4
```

vice.dataframe

The VICE Dataframe: base class

Provides a means of storing and accessing data with both case-insensitive strings and integers, allowing both indexing and calling.

Signature: vice.dataframe(frame)

Parameters

frame [`dict`] A python dictionary to construct the dataframe from. Keys must all be of type `str`.

Raises

- **TypeError**
  - frame has a key that is not of type `str`

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [column label] A label given to the stored quantity (or list/array thereof).
- **Values**
  - All

Indexing

- **`str` [case-insensitive]** [column label] A label given to the quantities stored.
- **`int`** [index for values which are array-like.] If all values stored by the dataframe are array-like, the i'th value of all of them can be obtained by indexing the dataframe with `i`.

Calling

The VICE dataframe and all subclasses can be called rather than indexed to achieve the same result.

Functions

- keys
- todict
- remove
- filter

Example Code

```
>>> import vice
>>> example = vice.dataframe({
        "a": [1, 2, 3],
        "b": [4, 5, 6],
        "c": [7, 8, 9]})
>>> example["A"]
[1, 2, 3]
>>> example("a")
[1, 2, 3]
>>> example[0]
vice.dataframe{
        a ---------------> 1
        b ---------------> 4
        c ---------------> 7
}
>>> example.keys()
['a', 'b', 'c']
>>> example.todict()
{'a': [1, 2, 3], 'b': [4, 5, 6], 'c': [7, 8, 9]}
>>> example.filter("c", "<", 9)
vice.dataframe{
        a ---------------> [1, 2]
        b ---------------> [4, 5]
        c ---------------> [7, 8]
}
```

vice.dataframe.keys

Returns the keys to the dataframe in their lower-case format

Signature: x.keys()

Parameters

x [dataframe] An instance of this class

Returns

keys [list] A list of lower-case strings which can be used to access the values stored in this dataframe.

Example Code

```
>>> import vice
>>> example = vice.dataframe({
        "a": [1, 2, 3],
        "b": [4, 5, 6],
        "c": [7, 8, 9]})
>>> example
vice.dataframe{
        a --------------> [1, 2, 3]
        b --------------> [4, 5, 6]
        c --------------> [7, 8, 9]
}
>>> example.keys()
['a', 'b', 'c']
```

vice.dataframe.todict

Returns the dataframe as a standard python dictionary

Signature: x.todict()

Parameters

x [dataframe] An instance of this class

Returns

copy [dict] A dictionary copy of the dataframe.

Note: Python dictionaries are case-sensitive, and are thus less flexible than this class.

Example Code

```
>>> import vice
>>> example = vice.dataframe({
        "a": [1, 2, 3],
        "b": [4, 5, 6],
        "c": [7, 8, 9]})
>>> example
vice.dataframe{
        a --------------> [1, 2, 3]
        b --------------> [4, 5, 6]
        c --------------> [7, 8, 9]
}
>>> example.todict()
{'a': [1, 2, 3], 'b': [4, 5, 6], 'c': [7, 8, 9]}
```

vice.dataframe.remove

Remove an element of the dataframe

Signature: x.remove(key)

New in version 1.1.0.

Parameters

x [`dataframe`] An instance of this class

key [`str` [case-insensitive]] The key to remove from the dataframe

Raises

- **KeyError**
  - Key is not in the dataframe

Example Code

```
>>> import vice
>>> example = vice.dataframe({
        "a": [1, 2, 3],
        "b": [4, 5, 6],
        "c": [7, 8, 9]})
>>> example
vice.dataframe{
        a --------------> [1, 2, 3]
        b --------------> [4, 5, 6]
        c --------------> [7, 8, 9]
}
>>> example.remove("a")
```

(continues on next page)

(continued from previous page)

```
vice.dataframe{
        b --------------> [4, 5, 6]
        c --------------> [7, 8, 9]
}
>>> example.remove("b")
vice.dataframe{
        c --------------> [7, 8, 9]
}
```

vice.dataframe.filter

Obtain a copy of the dataframe whose elements satisfy a filter. Only applies to dataframes whose values are all array-like.

Signature: x.filter(key, relation, value)

New in version 1.1.0.

Parameters

x [`dataframe`] An instance of this class

key [`str` [case-insensitive]] The dataframe key to filter based on

relation [`str`] Either '<', '<=', '=', '==', '!=', '>=', or '>', denoting the relation to filter based on.

value [real number] The value to filter based on.

Returns

filtered [`dataframe`] A dataframe whose elements are only those which satisfy the specified filter. This will always be an instance of the base class, even if the function called with an instance of a derived class.

Raises

- **KeyError**
  - Key is not in the dataframe
- **ValueError**
  - Invalid relation

Example Code

```
>>> import vice
>>> example = vice.dataframe({
        "a": [1, 2, 3],
        "b": [4, 5, 6],
        "c": [7, 8, 9]})
>>> example
vice.dataframe{
        a ---------------> [1, 2, 3]
        b ---------------> [4, 5, 6]
        c ---------------> [7, 8, 9]
}
>>> example.filter("a", "=", 2)
vice.dataframe{
        a ---------------> [2]
        b ---------------> [5]
        c ---------------> [8]
}
>>> example.filter("c", "<=", 8)
vice.dataframe{
        a ---------------> [1, 2]
        b ---------------> [4, 5]
        c ---------------> [7, 8]
}
```

vice.core.dataframe.agb_yield_settings

The VICE dataframe: derived class (inherits from yield_settings)

Stores the current nucleosynthetic yield settings for asymptotic giant branch (AGB) stars.

New in version 1.2.0.

Note: Modifying yield settings through these dataframes is equivalent to going through the vice.elements module.

Allowed Data Types

- **Keys**
  - **str [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - **str [case-insensitive]** [keywords] Denote a built-in table of net yields published in a nucleosynthesis study.

    Recognized Keywords:

    * "cristallo11" : Cristallo et al. (2011)[1]

[1] Cristallo et al. (2011), ApJS, 197, 17

* "karakas10" : Karakas (2010)[2]
* "ventura13" : Ventura et al. (2013)[3]
* "karakas16": Karakas & Lugaro (2016)[4]; Karakas et al. (2018)[5]

– **<function>** [Mathematical function describing the yield.] Must accept the stellar mass in $M_\odot$ and the metallicity by mass Z as parameters, in that order.

Note: Functions of mass and metallicity to describe these yields can significantly increase the required integration time in simulations, especially for fine timestepping.

Indexing

- **str [case-insensitive]** [elemental symbols] Must be indexed by the symbol of an element recognized by VICE as it appears on the periodic table.

Functions

- keys
- todict
- restore_defaults
- factory_settings
- save_defaults

Built-In Instances

- **vice.yields.agb.settings** The user's current nucleosynthetic yield settings for asymptotic giant branch stars.

Example Code

```
>>> import math
>>> from vice.yields.agb import settings as example
>>> example["c"] = "karakas10"
>>> def f(m, z):
        return 1e-3 * m * math.exp(-m / 2) * (z / 0.014)
>>> example["C"] = f
```

Signature: vice.core.dataframe.agb_yield_settings(frame, name, allow_funcs, config_field)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe and instead use the base class. Instances of this class are created automatically.

[2] Karakas (2010), MNRAS, 403, 1413
[3] Ventura et al. (2013), MNRAS, 431, 3642
[4] Karakas & Lugaro (2016), ApJ, 825, 26
[5] Karakas et al. (2018), MNRAS, 477, 421

Parameters

frame [dict] A dictionary from which to construct the dataframe.

name [str] String denoting a description of the values stored in this dataframe.

allow_funcs [bool] If True, functional attributes will be allowed.

config_field [str] The name of the ".config" file that is stored in VICE's install directory whenever the user saves new default yield settings.

vice.core.dataframe.ccsn_yield_table

The VICE dataframe: derived class (inherits from base)

Stores the data from a core collapse supernova (CCSN) mass yield table published in a nucleosynthesis study.

Note: This dataframe is not customizable. Its values cannot be altered.

Attributes

masses [tuple] The initial CCSN progenitor masses in $M_\odot$.

isotopes [tuple] The stable isotopes of the element with reported yields in $M_\odot$. None if created with a call to vice.yields.ccsne.table with the keyword argument `isotopic = False`.

Indexing

- **real number** [progenitor stellar mass] The initial mass of the CCSN progenitor in $M_\odot$.
- **str** [isotope] The isotope of the element. Only allowed when the dataframe is initialized with the keyword argument `isotopic = True`.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> example = vice.yields.ccsne.table('o')
>>> example
        vice.dataframe{
                13.0 -----------> 0.247071034
                15.0 -----------> 0.585730308
                20.0 -----------> 1.256452301
                25.0 -----------> 2.4764558329999997
                30.0 -----------> 0.073968147
```

(continues on next page)

(continued from previous page)

```
                40.0 ----------> 0.087475695
                60.0 ----------> 0.149385561
                80.0 ----------> 0.24224373600000002
                120.0 ----------> 0.368598602
        }
>>> example = vice.yields.ccsne.table('c', isotopic = True)
>>> example
        vice.dataframe{
                13 -------------> {'c12': 0.11404, 'c13': 0.0006918}
                15 -------------> {'c12': 0.22096, 'c13': 0.00077869}
                20 -------------> {'c12': 0.40941, 'c13': 0.0010411}
                25 -------------> {'c12': 0.61944, 'c13': 0.0012839}
                30 -------------> {'c12': 0.025054, 'c13': 0.0014423}
                40 -------------> {'c12': 0.031933, 'c13': 0.0018031}
                60 -------------> {'c12': 0.39826, 'c13': 0.0017254}
                80 -------------> {'c12': 1.1226, 'c13': 0.0017021}
                120 ------------> {'c12': 2.0178, 'c13': 0.0023899}
        }
```

Signature: vice.core.dataframe.ccsn_yield_table(masses, yields, isotopes = None)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe. To obtain instances of this class, use the lookup function vice.yields.ccsne.table.

Parameters

masses [tuple] The progenitor masses on which the grid is sampled in $M_\odot$.

yields [tuple] The yields at each mass. A 2-D tuple is interpreted as an isotopic breakdown.

isotopes [tuple [default][None]] The isotopes of the element. None if created with a call to vice.yields.ccsne.table with the keyword argument isotopic = False.

vice.core.dataframe.ccsn_yield_table.masses

Type : tuple

The initial masses of CCSN progenitors in $M_\odot$ reported by the nucleosynthesis study.

Example Code

```
>>> import vice
>>> example = vice.yields.ccsne.table("o")
>>> example.masses
        (13.0, 15.0, 20.0, 25.0, 30.0, 40.0, 60.0, 80.0, 120.0)
```

vice.core.dataframe.ccsn_yield_table.isotopes

Type : `tuple`

The stable isotopes whose yields are reported by the nucleosynthesis study. If this table was generated with a call to vice.yields.ccsne.table with the keyword argument `isotopic = False`, this attribute will be `None`.

Example Code

```
>>> import vice
>>> example = vice.yields.ccsne.table("c", isotopic = True)
>>> example.isotopes
        ('c12', 'c13')
```

vice.core.dataframe.channel_entrainment

The VICE dataframe: derived class (inherits from elemental_settings)

Stores entrainment fractions for each element from a given enrichment channel. These numbers denote the mass fraction of that element produced by some enrichment channel which is retained by the interstellar medium, the remainder of which is added directly to an outflow.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - **real number** [mass fraction] The mass fraction of the element that is entrained. Must be between 0 and 1.

Indexing

- **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> example = vice.singlezone(name = "example")
>>> example.entrainment.ccsne['o'] = 0.9
>>> example.entrainment.ccsne['fe'] = 0.95
>>> example.entrainment.sneia['fe'] = 0.95
```

Signature: vice.core.dataframe.entrainment(frame)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe and instead use the base class. Instances of this class are created automatically by the `singlezone` object.

Parameters

frame [`dict`] A dictionary from which to construct the dataframe.

vice.core.dataframe.elemental_settings

The VICE dataframe: derived class (inherits from base)

Stores data on an element-by-element basis.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbol] The symbol of a chemical element as it appears on the periodic table.
- **Values**
  - All

Indexing

- **`str` [case-insensitive]** [elemental symbol] The symbol of a chemical element as it appears on the periodic table.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> vice.yields.ccsne.settings['o'] = 0.01
>>> vice.yields.ccsne.settings['fe'] = 0.0012
>>> vice.yields.sneia.settings['o'] = 0.0
>>> vice.yields.sneia.settings['fe'] = 0.0017
```

Signature: vice.core.dataframe.elemental_settings(frame)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe and instead use the base class. Instances of this class are created automatically.

Parameters

frame [dict] A dictionary from which to construct the dataframe.

name [str] String denoting a description of the values stored in this dataframe.

vice.core.dataframe.evolutionary_settings

The VICE dataframe: derived class (inherits from elemental_settings)

Stores simulation parameters on an element-by-element basis which may or may not vary with time.

Allowed Data Types

- **Keys**
  - **str [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - **real number** A constant which does not vary with time.
  - **<function>** Must accept time in Gyr as the only parameter, and return the value of this parameter at that time for a given element.

Indexing

- **str [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> example = vice.singlezone(name = "example", Zin = {})
>>> example.Zin['o'] = 0.002
>>> example.Zin['fe'] = lambda t: 0.001 * (t / 3)
```

Signature: vice.core.dataframe.evolutionary_settings(frame, name)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe and instead use the base class. Instances of this class are created automatically.

Parameters

frame [dict] A dictionary from which to construct the dataframe.

name [str] String denoting a description of the values stored in this dataframe.

vice.core.dataframe.fromfile

The VICE dataframe: derived class (inherits from base)

Provides a means of storing and accessing generic simulation output. Fromfile objects are created by various functions which read in simulation output (e.g. vice.mdf).

Attributes

name [str] The name of the file that the data was pulled from.

size [tuple] Contains two integers: the (length, width) of the data.

Allowed Data Types

- **Keys**
  - **str [case-insensitive]** [the physical quantity] A name given to the physical quantity to take from or store with the output.

    Note: VICE automatically assigns keys to quantities in the output which cannot be overridden.

- **Values**
  - **array-like** Must have the same length as the values of the dataframe obtained from the output file.

Indexing

- `int` : A given line-number of the output. Returns a dataframe with the same keys, but whose values are taken only from the specified line of output.
- `str` [case-insensitive] : labels of the lists of quantities stored.

  For MDF objects, the following are assigned automatically by VICE when reading in the output and will not be re-assigned:

  - 'bin_edge_left' : Lower bin edges of the distribution
  - 'bin_edge_right' : Upper bin edges of the distribution
  - 'dn/d[x/h]' : The value of the probability distribution function of stars in their [X/H] logarithmic abundance.
  - 'dn/d[x/y]' : The value of the probability distribution function of stars in their [X/Y] logarithmic abundance ratio.

Functions

- keys
- todict
- filter

Example Code

```
>>> import vice
>>> example = vice.mdf("example")
>>> example.keys()
        ['bin_edge_left',
         'bin_edge_right',
         'dn/d[fe/h]',
         'dn/d[sr/h]',
         'dn/d[o/h]',
         'dn/d[sr/fe]',
         'dn/d[o/fe]',
         'dn/d[o/sr]']
>>> example["bin_edge_left"][:10]
        [-3.0, -2.95, -2.9, -2.85, -2.8, -2.75, -2.7, -2.65, -2.6, -2.55]
>>> example[60]
        vice.dataframe{
                bin_edge_left --> 0.0
                bin_edge_right -> 0.05
                dn/d[fe/h] -----> 0.0
                dn/d[sr/h] -----> 0.0
                dn/d[o/h] ------> 0.0
                dn/d[sr/fe] ----> 0.06001488
                dn/d[o/fe] -----> 0.4337209
                dn/d[o/sr] -----> 0.0
        }
```

Signature: vice.core.dataframe.fromfile(filename = None, labels = None, adopted_solar_z = None)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe. Fromfile objects are created by various functions which read in simulation output (e.g. vice.mdf).

Parameters

filename [`str` [default][None]] The name of the ascii file containing the output.

list [`list` of strings [default][None]] The strings to assign the column labels.

adopted_solar_z [real number [default][None]] The metallicity by mass of the sun $Z_{\odot}$ adopted in the simulation.

vice.core.dataframe.fromfile.name

Type : `str`

The name of the file that this data was read from.

Example Code

```
>>> import vice
>>> example = vice.mdf("example")
>>> example.name
        'example.vice/mdf.out'
```

vice.core.dataframe.fromfile.size

Type : `tuple`

Contains two integers: the (length, width) of the dataframe.

Example Code

```
>>> import vice
>>> example = vice.mdf("example")
>>> example.size
        (80, 8)
```

vice.core.dataframe.history

The VICE dataframe: derived class (inherits from fromfile)

Provides a means of storing and accessing the time-evolution of the interstellar medium from the output of a singlezone object. History objects can be created from VICE outputs by calling vice.history.

Attributes

name [`str`] The name of the file that the data was pulled from.

size [`tuple`] Contains two integers: the (length, width) of the data.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [the physical quantity.] A name given to the physical quantity to take from or store with the output.

    Note: VICE automatically assigns keys to quantities in the output which cannot be overridden. A list of them can be found here under *Indexing*.

- **Values**
  - **array-like** Must have the same length as the values of the dataframe obtained from the output file.

Indexing

- **`int`** [A given line-number of output.] Returns a dataframe with the same keys, but whose values are taken only from the specified line of output.
- **`str` [case-insensitive]** [labels of the lists of quantities stored.] The following are assigned automatically by VICE when reading in an output file and will not be re-assigned:
  - 'time' : Time in Gyr from the start of the simulation.
  - 'lookback' : Lookback time in Gyr from the end of the simulation.
  - 'mgas' : The mass of the interstellar medium in $M_\odot$
  - 'sfr' : Star formation rate in $M_\odot \text{yr}^{-1}$
  - 'ifr' : Infall rate in $M_\odot \text{yr}^{-1}$
  - 'ofr' : Outflow rate in $M_\odot \text{yr}^{-1}$
  - 'eta_0' : The user-specified value of the mass-loading parameter η, independent of the smoothing timescale $\tau_\star$ employed in the simulation.
  - 'r_eff' : The effective recycling parameter $\dot{M}_\text{r}/\dot{M}_\star$.
  - 'z_in(x)' : The inflow metallicity by mass Z of the element x.
  - 'z_out(x)' : The outflow metallicity by mass Z of the element x.
  - 'mass(x)' : The mass of the element x in the interstellar medium.

- 'z(x)' : The metallicity by mass Z of the element x in the interstellar medium.
- '[x/h]' : The logarithmic abundance relative to the sun of the element x, given by $\log_{10}(Z_x/Z_{x,\odot})$.
- '[y/x]' : The logarithmic abundance ratio relative to the sun between the elements y and x, given by $\log_{10}(Z_y/Z_{y,\odot}) - \log_{10}(Z_x/Z_{x,\odot})$.
- 'z' : The scaled total metallicity by mass Z.
- '[m/h]' : The scalled logarithmic metallicity relative to the sun, given by $\log_{10}(Z/Z_\odot)$.

Note: The scaled total metallicity by mass is defined by:

$$Z = Z_\odot \frac{\sum_i Z_i}{\sum_i Z_{i,\odot}}$$

where $Z_\odot$ is the metallicity of the sun adopted in the simulation, and Z_i is the abundance by mass of the i'th element. This scaling is employed so that an accurate estimation of the total metallicity can be obtained without every element's abundance information.

Note: The scaled logarithmic metallicity is defined from the scaled total metallcity by mass according to:

$$[M/H] = \log_{10}\left(\frac{Z}{Z_\odot}\right)$$

Functions

- keys
- todict
- filter

Example Code

```
>>> example = vice.history("example")
>>> example.keys()
        ['time',
         'mgas',
         'mstar',
         'sfr',
         'ifr',
         'ofr',
         'eta_0',
         'r_eff',
         'z_in(fe)',
         'z_in(sr)',
         'z_in(o)',
         'z_out(fe)',
         'z_out(sr)',
         'z_out(o)',
```

(continues on next page)

(continued from previous page)

```
         'mass(fe)',
         'mass(sr)',
         'mass(o)',
         'z(fe)',
         'z(sr)',
         'z(o)',
         '[fe/h]',
         '[sr/h]',
         '[o/h]',
         '[sr/fe]',
         '[o/fe]',
         '[o/sr]',
         'z',
         '[m/h]',
         'lookback']
>>> example[100]
        vice.dataframe{
                time -----------> 1.0
                mgas -----------> 5795119000.0
                mstar ----------> 2001106000.0
                sfr ------------> 2.897559
                ifr ------------> 9.1
                ofr ------------> 7.243899
                eta_0 ----------> 2.5
                r_eff ----------> 0.3534769
                z_in(fe) -------> 0.0
                z_in(sr) -------> 0.0
                z_in(o) --------> 0.0
                z_out(fe) ------> 0.0002769056
                z_out(sr) ------> 3.700754e-09
                z_out(o) -------> 0.001404602
                mass(fe) -------> 1604701.0
                mass(sr) -------> 21.44631
                mass(o) --------> 8139837.0
                z(fe) ----------> 0.0002769056166059748
                z(sr) ----------> 3.700754031107903e-09
                z(o) -----------> 0.0014046022178319376
                [fe/h] ---------> -0.6682579454664828
                [sr/h] ---------> -1.1074881208001155
                [o/h] ----------> -0.6098426789720387
                [sr/fe] --------> -0.43923017533363273
                [o/fe] ---------> 0.05841526649444406
                [o/sr] ---------> 0.4976454418280768
                z --------------> 0.0033582028978416337
                [m/h] ----------> -0.6200211036287412
                lookback -------> 9.0
        }
```

Signature: vice.core.dataframe.history(filename = None, adopted_solar_z = None, labels = None)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe. To obtain a history object from a VICE output, simply call vice.history.

Parameters

filename [`str` [default][None]] The name of the ascii file containing the history output.

adopted_solar_z [real number [default][None]] The metallicity by mass of the sun $Z_\odot$ adopted in the simulation.

labels [`list` of strings [default][None]] The strings to assign the column labels.

vice.core.dataframe.noncustomizable

The VICE dataframe: derived class (inherits from elemental_settings)

Stores data on an element-by-element basis that is not modifiable by the user.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbol] The symbol of a chemical element as it appears on the periodic table.
- **Values**
  - All

Indexing

- **`str` [case-insensitive]** [elemental symbol] The symbol of a chemical element as it appears on the periodic table.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> vice.atomic_number['c']
        6
>>> vice.solar_z['c']
        0.00236
```

Signature: vice.core.dataframe.noncustomizable(frame)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe and instead use the base class. Instances of this class are created automatically.

Parameters

frame [dict] A dictionary from which to construct the dataframe.

name [str] String denoting a description of the values stored in this dataframe.

vice.core.dataframe.saved_yields

The VICE dataframe: derived class (inherits from noncustomizable)

Stores nucleosynthetic yield settings that were used in simulation. This is only a saved copy and is not modifiable by the user.

See also:

- vice.core.dataframe.yield_settings
- vice.yields.ccsne.settings
- vice.yields.sneia.settings
- vice.yields.agb.settings

Allowed Data Types

- **Keys**
  - **str [case-insensitive]** [elemental symbol] The symbol of a chemical element as it appears on the periodic table.
- **Values**
  - **real number** A constant yield which does not vary with stellar mass or metallicity.
  - **<function>** A function of either one or two variables, depending on the enrichment channel. Core collapse and type Ia supernova yields will be function of metallicity, while asymptotic giant branch star yields will be functions of stellar mass and metallicity.
  - **str** Keywords denoting a built-in table of yields sampled on a grid of stellar masses and metallicities. Only allowed for asymptotic giant branch star yields.

Indexing

- **str [case-insensitive]** [elemental symbol] The symbol of a chemical element as it appears on the periodic table.

Functions

- keys
- todict

Example Code

```
>>> import vice
>>> example = vice.output("example")
>>> example.agb_yields
        vice.dataframe{
                fe --------------> cristallo11
                o ---------------> cristallo11
                sr --------------> cristallo11
        }
>>> example.ccsne_yields
        vice.dataframe{
                fe --------------> 0.000246
                o ---------------> 0.00564
                sr --------------> 1.34e-08
        }
```

Signature: vice.core.dataframe.saved_yields(frame, name)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe and instead use the base class. Instances of this class are created automatically by the `output` object.

Parameters

frame [`dict`] A dictionary from which to construct the dataframe.

name [`str`] String denoting a description of the values stored in this dataframe.

vice.core.dataframe.tracers

The VICE dataframe: derived class (inherits from history)

Provides a means of storing and accessing the star particles formed in a multizone simulation. Tracers objects can be created from VICE outputs by calling vice.stars.

Attributes

name [str] The name of the file that the data was pulled from.

size [tuple] Contains two integers: the (length, width) of the data.

Allowed Data Types

- **Keys**
  - **str [case-insensitive]** [the physical quantity] A name given to the physical quantity to take from or store with the output.

    Note: VICE automatically assigns keys to quantities in the output which cannot be overridden. A list of them can be found here under *Indexing*.

- **Values**
  - **array-like** Must have the same length as the values of the dataframe obtained from the output file.

Indexing

- **int** [A given line-number of output.] Returns a dataframe with the same keys, but whose values are taken only from the specified line of output.
- **str [case-insensitive]** [labels of the lists of quantities stored.] The following are assigned automatically by VICE when reading in an output file and will not be re-assigned:
  - 'formation_time' : Time in Gyr from the start of the simulation when a star particle formed.
  - 'zone_origin' : The zone number the star formed in.
  - 'zone_final' : The zone number of the star at the end of the simulation.
  - 'mass' : The mass of the star particle in $M_\odot$.
  - 'z(x)' : The metallicity by mass Z of the element x in that star particle.
  - '[x/h]' : The logarithmic abundance relative to the sun of the element x, given by $\log_{10}(Z_x/Z_{x,\odot})$.
  - '[y/x]' : The logarithmic abundance ratio relative to the sun between the elements y and x, given by $\log_{10}(Z_y/Z_{y,\odot}) - \log_{10}(Z_x/Z_{x,\odot})$.
  - 'z' : The scaled toatl metallicity by mass Z.
  - '[m/h]' : The scaled logarithmic metallicity relative to the sun, given by $\log_{10}(Z/Z_\odot)$.

  Note: The scaled total metallicity by mass is defined by:

  $$Z = Z_\odot \frac{\sum_i Z_i}{\sum_i Z_{i,\odot}}$$

  where $Z_\odot$ is the metallicity of the sun adopted in the simulation, and Z_i is the abundance by mass of the i'th element. This scaling is employed so that an accurate estimation of the total metallicity can be obtained without every element's abundance information.

Note: The scaled logarithmic metallicity is defined from the scaled total metallicity by mass according to:

$$[M/H] = \log_{10}\left(\frac{Z}{Z_\odot}\right)$$

Functions

- keys
- todict
- filter

Example Code

```
>>> example = vice.stars("example")
>>> example.keys()
        ['formation_time',
         'zone_origin',
         'zone_final',
         'mass',
         'z(fe)',
         'z(sr)',
         'z(o)',
         '[fe/h]',
         '[sr/h]',
         '[o/h]',
         '[sr/fe]',
         '[o/fe]',
         '[o/sr]',
         'z',
         '[m/h]',
         'age']
>>> example[100]
        vice.dataframe{
                formation_time -> 0.1
                zone_origin ----> 0.0
                zone_final -----> 0.0
                mass -----------> 29695920.0
                z(fe) ----------> 1.128362e-05
                z(sr) ----------> 6.203682e-10
                z(o) -----------> 0.0002587532
                [fe/h] ---------> -2.058141258363775
                [sr/h] ---------> -1.8831288138453521
                [o/h] ----------> -1.3445102993763647
                [sr/fe] --------> 0.17501244451842268
                [o/fe] ---------> 0.7136309589874101
                [o/sr] ---------> 0.5386185144689875
                z --------------> 0.00053930007991864363
                [m/h] ----------> -1.4142969718113587
```

(continues on next page)

(continued from previous page)

```
                age ------------> 9.9
        }
```

Signature: vice.core.dataframe.tracers(filename = None, adopted_solar_z = None, labels = None)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe. To obtain a tracers object from a VICE output, simply call vice.stars.

Parameters

filename [`str` [default][None]] The name of the ascii file containing the tracers output.

adopted_solar_z [real number [default][None]] The metallicity by mass of the sun $Z_\odot$ adopted in the simulation.

labels [`list` of strings [default][None]] The strings to assign the column labels.

vice.core.dataframe.yield_settings

The VICE dataframe: derived class (inherits from elemental_settings)

Stores the current nucleosynthetic yield settings for different enrichment channels.

Note: Modifying yield settings through these dataframes is equivalent to going through the vice.elements module.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - real number : denote a constant, metallicity-independent yield.
  - **<function>** [Mathematical function describing the yield.] Must accept the metallicity by mass Z as the only parameter.

    Note: Functions of metallicity for yields of delayed enrichment channels (e.g. type Ia supernovae) can significantly increase the required integration time in simulations, especially for fine timestepping.

Indexing

- **str [case-insensitive]** [elemental symbols] Must be indexed by the symbol of an element recognized by VICE as it appears on the periodic table.

Functions

- keys
- todict
- restore_defaults
- factory_settings
- save_defaults

Built-In Instances

- **vice.yields.ccsne.settings** The user's current nucleosynthetic yield settings for core collapse supernovae.
- **vice.yields.sneia.settings** The user's current nucleosynthetic yield settings for type Ia supernovae.

Example Code

```
>>> from vice.yields.ccsne import settings as example
>>> example["fe"] = 0.001
>>> example["FE"] = 0.0012
>>> def f(z):
        return 0.005 + 0.002 * (z / 0.014)
>>> example["Fe"] = f
```

Signature: vice.core.dataframe.yield_settings(frame, name, allow_funcs, config_field)

Warning: Users should avoid creating new instances of derived classes of the VICE dataframe.

Parameters

frame [dict] A dictionary from which to construct the dataframe.

name [str] String denoting a description of the values stored in this dataframe.

allow_funcs [bool] If True, functional attributes will be allowed.

config_field [str] The name of the ".config" file that is stored in VICE's install directory whenever the user saved new default yield settings.

vice.core.dataframe.yield_settings.restore_defaults

Restores the dataframe to its default parameters.

Signature: x.restore_defaults()

Parameters

x [yield_settings] An instance of this class.

Example Code

```
>>> from vice.yields.ccsne import settings as example
>>> example["fe"]
0.000246
>>> example["fe"] = 0.001
>>> example.restore_defaults()
>>> example["fe"]
0.000246
```

vice.core.dataframe.yield_settings.factory_settings

Restores the dataframe to its factory defaults.

Signature: x.factory_settings()

Tip: To revert your nucleosynthetic yield settings back to the production defaults *permanently*, simply call `x.save_defaults()` immediately following this function.

Parameters

x [yield_settings] An instance of this class

Example Code

```
>>> from vice.yields.ccsne import settings as example
>>> example["fe"]
0.001 # <--- previously saved preset
>>> example.factory_settings()
0.000246
```

vice.core.dataframe.yield_settings.save_defaults

Saves the current dataframe settings as the default values.

Signature: x.save_defaults()

Parameters

x [yield_settings] An instance of this class.

Note: Saving functional yields requires the package dill, an extension to pickle in the python standard library. It is recommended that VICE users install dill >= 0.2.0.

Example Code

```
>>> from vice.yields.ccsne import settings as example
>>> example["fe"] = 0.001
>>> example.save_defaults()
```

After re-launching the python interpreter:

```
>>> from vice.yields.ccsne import settings as example
>>> example["fe"]
0.001
```

vice.ScienceWarning

A Warning class designed to treat as a distinct set of warnings those related to the scientific accuracy or precision of values returned from a given function.

Signature: vice.ScienceWarning

Although it is not recommended, this class of warnings can be silenced via:

```
>>> warnings.filterwarnings("ignore", category = vice.ScienceWarning)
```

Alternatively, to silence all errors within VICE:

```
>>> vice.warnings.filterwarnings("ignore")
```

To silence all warnings globally:

```
>>> warnings.filterwarnings("ignore")
```

vice.VisibleRuntimeWarning

A `RuntimeWarning` which - contrary to the python default `RuntimeWarning` - is visible by default. Features which raise this warning may take considerably longer to finish than otherwise.

Signature: vice.VisibleRuntimeWarning

New in version 1.1.0.

Although it is not recommended, this class of warnings can be silenced via:

```
>>> warnings.filterwarnings("ignore",
        category = vice.VisibleRuntimeWarning)
```

Alternatively, to silence all errors within VICE:

```
>>> vice.warnings.filterwarnings("ignore")
```

To silence all warnings globally:

```
>>> warnings.filterwarnings("ignore")
```

vice.VisibleDeprecationWarning

A `DeprecationWarning` which - contrary to the python default `DeprecationWarning` - is visible by default. Features which raise this warning are deprecated and will be removed in a future release of VICE.

Signature: vice.VisibleDeprecationWarning

New in version 1.1.0.

Although it is not recommended, this class of warnings can be silenced via:

```
>>> warnings.filterwarnings("ignore",
        category = vice.VisibleDeprecationWarning)
```

Alternatively, to silence all errors within VICE:

```
>>> vice.warnings.filterwarnings("ignore")
```

To silence all warnings globally:

```
>>> warnings.filterwarnings("ignore")
```

CHAPTER

FIVE

DEVELOPER'S DOCUMENTATION

5.1 License

VICE is protected under an MIT License, found in the git repository:

MIT License

Copyright (c) 2019 James W. Johnson

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

5.2 Citing VICE

Usage of VICE in research publications should cite Johnson & Weinberg (2020). If your work makes use of any of the following capabilities, please also cite the indicated paper:

- Multi-zone chemical evolution models: Johnson et al. (2021)
- Core collapse supernova yield calculations: Griffith et al. (2021)

If you're compiling your references with `BibTeX`, the following entries can be added to your `.bib` file:

Johnson & Weinberg (2020):

```
@ARTICLE{2020MNRAS.498.1364J,
       author = {{Johnson}, James W. and {Weinberg}, David H.},
        title = "{The impact of starbursts on element abundance ratios}",
      journal = {\mnras},
     keywords = {methods: numerical, galaxies: abundances, galaxies: evolution,␣
↪galaxies: star formation, galaxies: stellar content, Astrophysics - Astrophysics of␣
↪Galaxies},
```

(continues on next page)

(continued from previous page)

```
         year = 2020,
        month = aug,
       volume = {498},
       number = {1},
        pages = {1364-1381},
          doi = {10.1093/mnras/staa2431},
archivePrefix = {arXiv},
       eprint = {1911.02598},
 primaryClass = {astro-ph.GA},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2020MNRAS.498.1364J},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

Johnson et al. (2021):

```
@ARTICLE{2021arXiv210309838J,
       author = {{Johnson}, James W. and {Weinberg}, David H. and {Vincenzo}, Fiorenzo␣
↪and {Bird}, Jonathan C. and {Loebman}, Sarah R. and {Brooks}, Alyson M. and {Quinn},␣
↪Thomas R. and {Christensen}, Charlotte R. and {Griffith}, Emily J.},
        title = "{Stellar Migration and Chemical Enrichment in the Milky Way Disc: A␣
↪Hybrid Model}",
      journal = {arXiv e-prints},
     keywords = {Astrophysics - Astrophysics of Galaxies},
         year = 2021,
        month = mar,
          eid = {arXiv:2103.09838},
        pages = {arXiv:2103.09838},
archivePrefix = {arXiv},
       eprint = {2103.09838},
 primaryClass = {astro-ph.GA},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2021arXiv210309838J},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

Griffith et al. (2021):

```
@ARTICLE{2021arXiv210309837G,
       author = {{Griffith}, Emily J. and {Sukhbold}, Tuguldur and {Weinberg}, David H.␣
↪and {Johnson}, Jennifer A. and {Johnson}, James W. and {Vincenzo}, Fiorenzo},
        title = "{The Impact of Black Hole Formation on Population Averaged Supernova␣
↪Yields}",
      journal = {arXiv e-prints},
     keywords = {Astrophysics - Solar and Stellar Astrophysics, Astrophysics -␣
↪Astrophysics of Galaxies},
         year = 2021,
        month = mar,
          eid = {arXiv:2103.09837},
        pages = {arXiv:2103.09837},
archivePrefix = {arXiv},
       eprint = {2103.09837},
 primaryClass = {astro-ph.SR},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2021arXiv210309837G},
```

(continues on next page)

(continued from previous page)

```
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

5.3 Contributors

5.3.1 James W. Johnson

Lead Developer and **License Owner** (Spring 2018 - Present)
Email: giganano9@gmail.com johnson.7419@buckeyemail.osu.edu
Webiste: https://sites.google.com/view/jameswjohnson/
The Ohio State University Department of Astronomy
140 W. 18th Ave., Columbus, OH, 43204

5.3.2 Emily J. Griffith

Contributing Developer (Summer 2020 - Present)
Email: griffith.802@buckeyemail.osu.edu
Website: https://www.emilyjgriffith.com
The Ohio State University Department of Astronomy
140 W. 18th Ave., Columbus, OH, 43204

5.3.3 John W. Bredall

Contributing Developer (Fall 2021 - Present)
Email: bredall.1@buckeyemail.osu.edu
Website: https://u.osu.edu/jbredall/
The Ohio State University Department of Astronomy
140 W. 18th Ave., Columbus, OH, 43204

5.3.4 David H. Weinberg

Advising Developer (Spring 2018 - Present)
Website: http://www.astronomy.ohio-state.edu/~dhw/
The Ohio State University Department of Astronomy
140 W. 18th Ave., Columbus, OH, 43204

5.3.5 Fiorenzo Vincenzo

Advising Developer (Spring 2019 - Present)
Website: https://ccapp.osu.edu/people/vincenzo.3
The Ohio State University Center for Cosmology and Astroparticle Physics
191 West Woodruff Ave., Columbus, OH, 43210

5.3.6 Jonathan C. Bird

Advising Developer (Spring 2019 - Present)
Website: http://vpac00.phy.vanderbilt.edu/~birdjc1/
Vanderbilt University Department of Physics & Astronomy
6301 Stevenson Center, 2301 Vanderbilt Place, Nasvhille, TN 37235

5.3.7 Jennifer A. Johnson

Advising Developer (Summer 2020 - Present)
Website: http://www.astronomy.ohio-state.edu/~jaj/
The Ohio State University Department of Astronomy
140 W. 18th Ave., Columbus, OH, 43204

5.4 Acknowledgements

J.W.J. and E.J.G. acknowledge financial support from Ohio State University Graduate Fellowships. The developers acknowledge further financial support from National Science Foundation grant AST-1909841. We are grateful for valuable discussion with the Ohio State Astronomy Gas, Galaxies, and Feedback group (2017 - present). J.W.J. acknowledges input from Jenna Freudenburg on the implementation of the cumulative return fraction. The developers thank Paolo Ventura for providing unpublished, theoretically predicted yields from asymptotic giant branch stars at progenitor metallicities of $Z = 0.001$ and 0.002. The developers also thank Ryan J. Cooke for discussion of the primordial helium abundance measured by Pitrou et al. (2021), MNRAS, 502, 2474.

5.5 Submitting a Bug Report

If you suspect buggy behavior in VICE, please open an issue at the issues page on GitHub. Create a new issue with a description of the problem, a copy of relevant pieces of code, and a full traceback if possible. Please also attach the label *bug* to the issue.

5.6 Contributing to VICE

VICE is written in a cohesive manner around a core set of objects. That is, VICE's implementation shares one library, with considerable overlap between relevant calculations (e.g. the `multizone` object makes use of the `singlezone` object, which makes use of the `dataframe` objects, and so on). The `dataframe` being the exception which is implemented in `Cython`, the majority of these objects are implemented in C, declared via `typedef struct` statements in the file `vice/src/objects/objects.h`. VICE's entire C library can be found in the directory `vice/src/`, and the major components of its python implementation in `vice/core/`. This includes the `singlezone` and `multizone` objects, the `dataframe` and all derived classes, the `output` and `multioutput` objects, and single stellar population routines in the `vice/core/ssp/` subdirectory. The hierarchical file structure of these directories are designed to mirror one another. The `yields` module, however, is separate from the VICE `core`, and it implements all of the nucleosynthetic yield calculations, inependent of the chemical evolution model/simulation features. Those wishing to contribute to VICE are strongly encouraged to reach out to our *contributors* or to join us on Slack; we are happy to collaborate with those interested in extending VICE's capabilities!

Note: The primary author (James W. Johnson) reserves the right to make revisions to all contributed code and associated documentation.

5.6.1 Building a New Extension

To contribute to VICE, first fork the repository and develop the necessary objects and functions in the fork. The changes should reflect the overall design of the package and should, to the best of one's ability, comply with the stylistic conventions adopted throughout the code base.

All extensions should be given unit tests, making use of the `moduletest` and `unittest` objects scripted in the files `vice/testing/moduletest.py` and `vice/testing/unittest.py`. These objects can be created from the `@unittest` and `@moduletest` decorators implemented in `vice/testing/decorators.py`. The associated documentation should provide adequate instruction on how to make use of these objects.

5.6.2 Documenting Changes

All docstrings visible to the user after installation should be in the `numpydocs` format. This is not required (though recommended) for docstrings not accessible to the user. Any C routines added to the source code should be given comment headers with descriptions of their purpose, any parameters they accept, what they return, and the header files they're declared in. These comment headers should reflect the style of those already present in the C library. Finally, add the new features to the API reference config file at `docs/src/api/pkgcontents/gen/config.py` and generate the documentation by running `make` in the `docs/` directory.

5.6.3 Submitting a Contribution

To submit your contribution, first conduct the steps outlined above, then please open a pull request, and label it as an *enhancement*.