Versatile Integrator for Chemical Evolution

Version 1.1.0

CONTENTS

Version 1.1.0

James W. Johnson

Primary Author

Email: giganano9@gmail.com

The Ohio State University Department of Astronomy

140 W. 18th Ave., Columbus, OH, 43204

Welcome to VICE's documentation! Source code and more resources can found in the git repository. To cite this version of VICE, please reference Johnson & Weinberg (2020). Any modifications to the source code will require rebuilding VICE from source for the change to take effect.

First-time users should familiarize themselves with VICE's API by going through the tutorial, available in the git repository. Any questions users may have can be emailed to the primary author (James W. Johnson).

CHAPTER

ONE

INSTALLING VICE

Binary installers of the latest version of VICE for python versions 3.5-3.8 on Mac OS X and Linux operating systems can be found on PyPI. We recommend that VICE be installed in this manner by running `pip install vice [--user]` from a bash terminal. Users should add the `--user` flag if they do not have administrator privileges; this will install VICE to their `~/.local` directory.

Designed for Unix system architectures, VICE does not function within a windows environment. Windows users should therefore install VICE within the Windows Subsystem for Linux (WSL). An *installation from source* on a windows machine should also be ran from within WSL.

Users who have or would like to modify VICE's source code should conduct a *from source installation*; this also applies to users who would like to install for a development version of python, such as 3.9. Installing from source is also an alternative in the event that the PyPI installation fails for some reason. If you have already installed VICE and would like help getting started, usage guidelines and tutorials can be found here.

Contents

1.1 Dependencies

VICE has no *primary* runtime dependencies; that is, it does not require any external software to run properly. 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 recognized 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 Installing from Source

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

1. Cython >= 0.28.0
2. Python >= 3.5
3. Make >= 3.81
4. gcc >= 4.6 or clang >= 3.6.0

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.

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
```

To install VICE, then run:

```
$ make
$ python setup.py build -j 4 install
```

This will compile VICE on 4 CPUs in parallel and subsequently install. Users installing VICE on a system on which they do not have adminstrator's privileges should perform a local installation. This can be achieved with the `--user` command-line argument:

```
$ python setup.py build -j 4 install --user
```

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. Following the installation, to run the tests and clean the source tree:

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

Please also note that `make tests` runs VICE's tests in the user's default version of python. To force the tests to run in python 3, run `make tests3`. Alternatively, the tests can be ran from within python itself:

```
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 Things to Avoid

1. **conda Environments** VICE should **never** be installed from source within a conda environment. This only applies to from source installations; a binary installation from PyPI should run properly within any conda environment provided the version of python is supported. When installing from source in a conda environment, the installation process will run without errors, but the compiled extensions are not placed in the correct directory, preventing VICE from running properly. This does not apply to the default environment `base` associated with later versions of python and Anaconda.

   VICE will *run* within whatever conda environments users create; it is only the source installation process that this applies to. As noted *here*, VICE is implemented entirely independently of Anaconda, and for this reason, it does not make sense to install VICE from source in a conda environment anyway. This is also true for installing from PyPI in a conda environment, unless a specific version of python is required.

2. **Parallel Installations** Users installing VICE to 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; it's likely this will caues a `segmentation fault`.

1.2.2 Additional Options

By default, VICE will install verbosely, printing to the console. To turn this off, run a quiet installation:

```
$ python setup.py build -j 4 install -q
```

or

```
$ python setup.py build -j 4 install --quiet
```

To change the number of cores used to compile VICE:

```
$ python setup.py build -j <number of cores> install
```

If you have modified VICE's source code and are reinstalling your modified version, there's no need to rebuild the entire package. Any number of extensions can be specified with the `ext` directive. For example, the following will rebuild the singlezone object, whose extension is `vice.core.singlezone._singlezone`:

```
$ python setup.py build install ext=vice.core.singlezone._singlezone
```

1.3 Troubleshooting Your Build

The following are a number of issues that can arise when installing VICE from source. If none of these options solve your problem, you may open an issue here, or email VICE's primary author (James W. Johnson) at giganano9@gmail.com.

1.3.1 ImportError After Installation

Did you install VICE from within a conda environment? If not, please open an issue here.

1.3.2 Running the setup.py File Failed

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

1.3.3 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.

1.3.4 VICE Isn't Running from the Command Line

In this case, it is likely that the required files were copied somewhere that is not on your PATH. If re-installing VICE does not solve the problem, these files can simply be copied to a given directory. For example:

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

Will place both command line entries in the `~/.local/bin/` directory. This can be permanently added to your path by adding

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

to `~/.bash_profile`. This will require `source ~/.bash_profile` to be ran from the terminal before `vice` can be ran from the command line.

Note: If you have installed VICE with the `--user` option, it is likely that VICE has automatically modified your PATH, and that `source ~/.bash_profile` is all that needs ran.

More information on modifying your PATH can be found here.

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

1.3.5 Compiler Failure

This is usually an indication that the build should not be ran on multiple cores. 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 fixed 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 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, rerunning `which vice` should now print nothing.

CHAPTER

TWO

GETTING STARTED

Any questions regarding usage of VICE or its implementation can be directed to the primary author (James W. Johnson: giganano9@gmail.com).

2.1 Tutorial

Under `examples` in VICE's source directory is the quick start tutorial, a notebook intended to provide first-time users with a primar on how to use all of VICE's features. After installation, this jupyter notebook can be viewed in the web browser by running `vice --tutorial` from the command line. Alternatively, if installing from source, it can be launched via `make tutorial` in the root directory. To download this jupyter notebook, simply clone the git repository if you haven't already, and it will be under the `examples` directory.

2.2 Example Code

We provide example scripts in VICE's source tree under examples.

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 git repository.

2.4 From the Command Line

VICE allows simple simulations 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 this feature does not work after installing VICE, 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

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 are in fractional units; that is, they quantify the mass fraction of stellar material's initial mass that is processed into a given element and subsequently ejected to the ISM. Nucleosynthetic products that end up locked in stellar remnants should not be taken into account in these models. These values are denoted with a lower-case y with test subscripts and superscripts denoting the element and the enrichment channel.

The metallicity by mass 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:

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

and logarithmic abundance ratios [X/Y]:

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

Here and hereafter the symbols $\odot$ and τ refer to the sun and a timescale, respectively.

3.1 Background

3.1.1 Galactic Chemical Evolution

Galactic Chemical Evolution (often referred to as galactic archaeology) studies the connection between a galaxy's evolution and the chemical compositions of its stars. 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 archaeology, 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.

3.2 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 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 implemented in its entirety in ANSI/ISO C, standard library Python, and standard library Cython. With this implementation, VICE is entirely cross platform and independent of the user's version of Anaconda (or lackthereof). However, VICE is not wrapped for installation in a Windows environment without modifying the installation source code. We recommend users install and run VICE in a linux environment using the Windows Terminal.

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. 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.

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 singlezone 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

In VICE we adopt the following functional form for the lifetime of a star on the main sequence:

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

where $\tau_\odot$ is the sun's main sequence lifetime, α is the power-law index of the mass-lifetime relationship. The constant `SOLAR_LIFETIME` declares $\tau_\odot$ = 10 Gyr, and `MASS_LIFETIME_PLAW_INDEX` delcares α = 3.5. Both constants are declared in `vice/src/ssp.h`.

The scaling of $\tau_{\text{MS}} \sim m^{-3.5}$ fails for high mass stars ($\gtrsim 8M_\odot$), but these stars have lifetimes that are very short compared to the relevant timescales of galactic chemical evolution (∼few` Gyr). This approximation fails for low

[1] Johnson & Weinberg (2020), arxiv:1911.02598

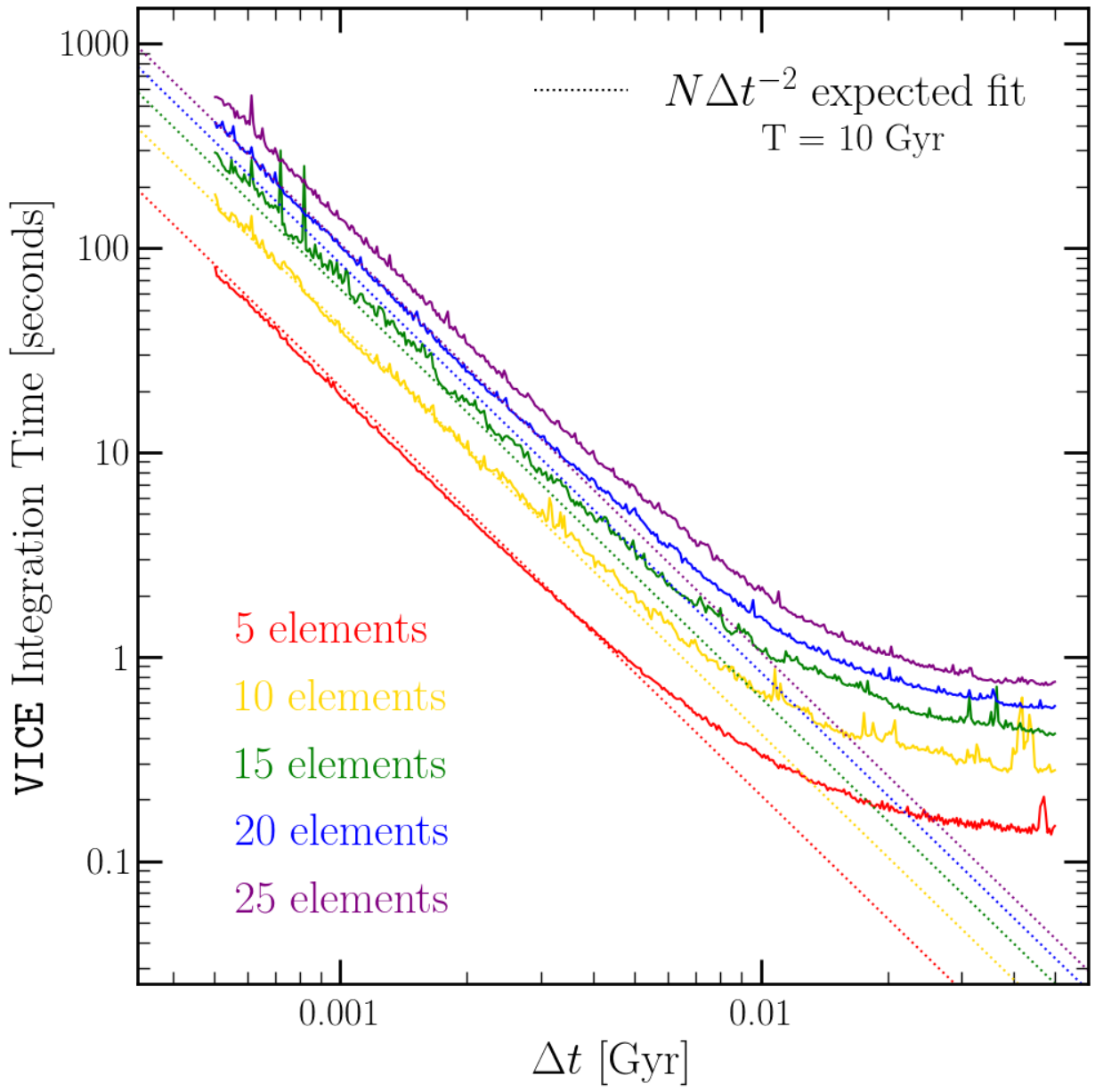

Fig. 1: Timed runs with N = 5, 10, 15, 20, and 25 elements with timesteps ranging from 500 kyr to 10 Myr (solid lines) with an ending time of T_{end} = 10 Gyr. The color-coded dotted lines show the $N\Delta t^{-2}$ expected scaling relation. The fit does well for small Δt, but underpredicts the integration 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.

mass stars as well ($\lesssim 0.5M_\odot$), but these stars have very long lifetimes that are considerably longer than the age of the universe. Because VICE does not support simulations on this long of timescales, this approximation suffices for all timescales of interest.

This is motivated by a conventional power-law relationship between mass and luminosity $L \sim M^{+\beta}$. The lifetime then scales as $\tau \sim M/L \sim M^{1-\beta}$. α = 3.5 corresponds to $L \sim M^{4.5}$ in the mass range of interest.

This equation can be generalized to find the the *total lifetime* of a star of mass m: the time until it produces a remnant by simply amplifying the lifetime by a factor $1 + p_{\rm MS}$:

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

where $p_{\rm MS}$ is an adopted lifetime ratio of the post main sequence to main sequence phases of stellar evolution.

By interpreting $\tau_{\rm total}$ as lookback time, we can solve for the mass of remnant producing stars under this model.

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

This equation allows the solution of both the *main sequence turnoff mass* and the mass of stars at the end of their post main sequence lifetimes by whether or not $p_{\rm MS}$ = 0.

Relevant source code:

- `vice/src/ssp.h`
- `vice/src/ssp/mlr.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:

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

The current version of VICE employs the initial-final remnant mass relation of Kalirai et al. (2008)[1]:

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

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

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

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

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

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

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. Weinberg, Andrews, and Freudenburg (2017)[2] adopted instantaneous recycling, whereby a fraction of the stellar population's mass r_{inst} is returned *instantaneously* in the interest of an analytic approach to singlezone models. They find that $r_{\text{inst}} = 0.4$ and $r_{\text{inst}} = 0.2$ is an adequate approximation for Kroupa[3] and Salpeter[4] 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.

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:

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

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

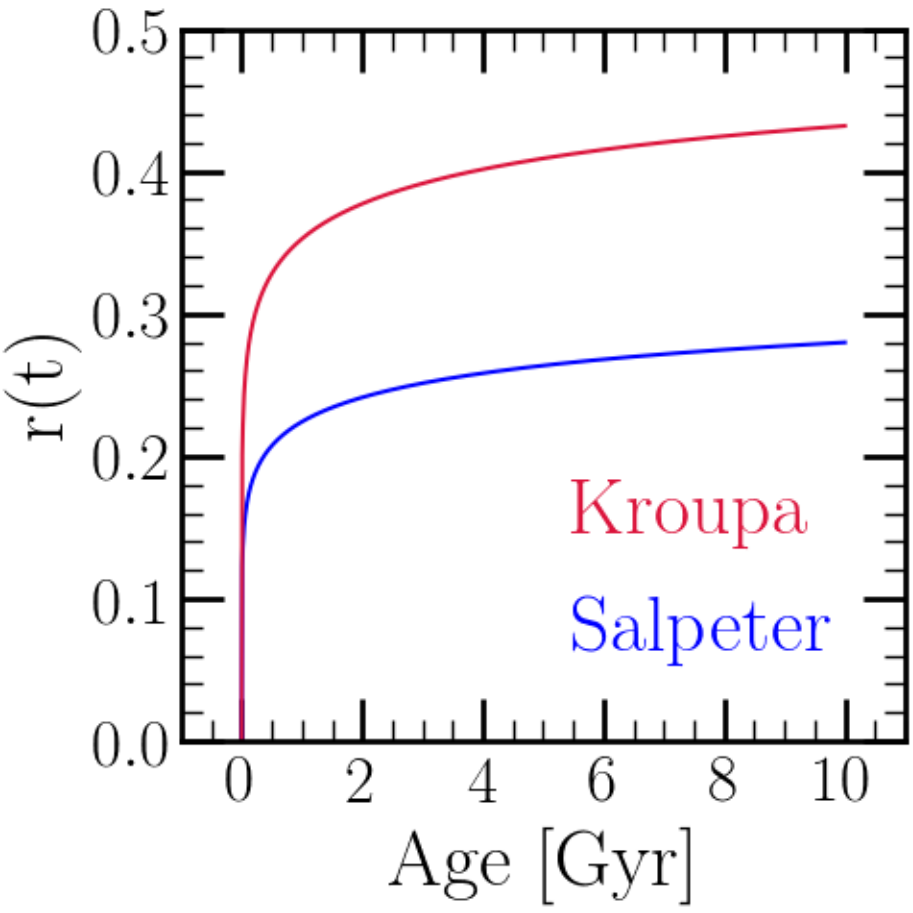

Fig. 2: The cumulative return fraction as a function of age for Kroupa[5] (red) and Salpeter[6] (blue) IMFs. The Kroupa IMF is higher at all nonzero ages because it has fewer low mass stars than Salpeter. In both cases the post main sequence lifetime is assumed to be 10% of the main sequence lifetime (i.e. $p_{\rm MS} = 0.1$).

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[7]:

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

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. By 10 Gyr, $h(t)$ is as low as ~ 0.45 for the Kroupa IMF and ~ 0.65 for the Salpeter[8] 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.

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[11], the entire CCSN net yield is injected within the first timestep at $t = \Delta t$:

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

[7] Kroupa (2001), MNRAS, 322, 231
[8] Salpeter (1955), ApJ, 121, 161
[9] Kroupa (2001), MNRAS, 322, 231
[10] Salpeter (1955), ApJ, 121, 161
[11] See the discussion of *enrichment from CCSNe* for justification of this assumption.

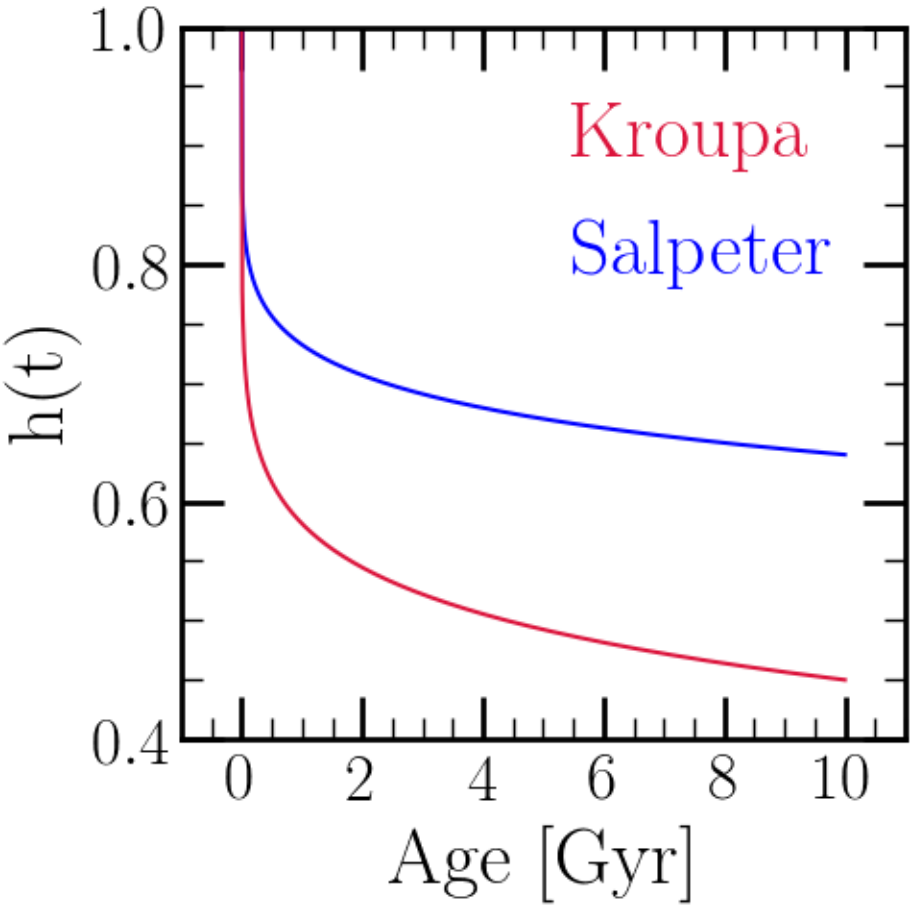

Fig. 3: The main sequence mass fraction as a function of age for Kroupa[9] and Salpeter[10] IMFs. The Kroupa IMF is lower at all nonzero ages because it has fewer low mass stars than Salpeter.

where $y_x^{\rm CC}(Z)$ 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[12]:

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

and from type Ia supernovae (SN Ia) according to[13]:

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

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`

3.4 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_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 ($\dot{M}_{\rm in}$ as a function of time), a star formation history ($\dot{M}_\star$ as a function of time), or the gas history ($\dot{M}_{\rm gas}$ 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 $\dot{M}_g$ specified by the user, $\tau_\star$, and the implementation of $\dot{M}_{\rm out}$ and $\dot{M}_{\rm r}$ discussed in this section, the solution to M_g as a function of time is 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:

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

[12] Justification of this can be found *here*.
[13] Justification of this can be found *here*.
[1] In the astronomical literature, this quantity is often referred to as the "depletion time" rather than star formation efficiency. In the chemical evolution literature, it quantifies the fractioanl rate at which gas is converted into stars, and is thus referred to as star formation efficiency. This is the

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

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: $\eta \equiv \dot{M}_\text{out}/\dot{M}_\star$. 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:

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

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

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

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 $\tau_\text{s} < \Delta t$, VICE automatically recovers the traditional relation of $\dot{M}_\text{out} = \eta(t)\dot{M}_\star(t)$.

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:

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

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

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

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

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

Weinberg, Andrews & Freudenburg (2017)[2] demonstrate that r_{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`

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.

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

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.

---

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

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`

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_{Ia} and an age τ, the rate of production of some element x from a single stellar population is given by

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

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^{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. ϵ_x^{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.

---

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

In practice, y_x^{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^{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:

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

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:

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

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

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

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

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

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

VICE implements this normalization of R_{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": $R_{\text{Ia}} \sim t^{-1.1}$
- "exp": $R_{\text{Ia}} \sim e^{-t/\tau_{\text{Ia}}}$

Users may also construct their own functional forms of R_{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`

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

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:

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

This expression is approximated numerically as:

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

[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`

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

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

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:

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

[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

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`

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:

$$y_x^{\text{CC}} \equiv \frac{\int_{l_{\text{CC}}}^{u} E(m) m_x \frac{dN}{dm} dm}{\int_l^u m \frac{dN}{dm} dm}$$

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). $E(m)$ denotes the *explodability*: the fraction of stars of mass m which explode as a CCSN. The

[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.

constant `CC_MIN_STELLAR_MASS` declares $l_{\rm CC} = 8M_\odot$ in `vice/src/ccsne.h`. This equation is nothing more than the mathematical statement of "production divided by total initial mass."

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 values from a user-specified nucleosynthesis study, interpolating linearly between values of m on the grid. We clarify that the interpolation is linaer 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
- Chieffi & Limongi (2013), ApJ, 764, 21
- Chieffi & Limongi (2004), ApJ, 608, 405
- Woosley & Weaver (1995), ApJS, 101, 181

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 $l_{\rm CC}$, in order to minimize numerical artifacts introduced when extrapolating off of the grid in m to lower stellar masses.

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}.

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

- Cristallo et al. (2011), ApJS, 197, 17
- Karakas (2010), MNRAS, 403, 1413

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 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.

3.8 Stellar Metallicity Distribution Functions

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

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

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

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:

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

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

CHAPTER

FOUR

COMPREHENSIVE API REFERENCE

4.1 From the Command Line

Included with VICE is a command line entry which runs simple simulations from a terminal. This feature allows the parameters of a onezone model to be specified as command-line arguments; run `vice --help` from a terminal after installing VICE (from any directory except the source tree). 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.

VICE also includes a command-line entry for automatically accessing the documentation. Simply run `vice --docs` from any directory except the source tree, and the documentation will be opened by the default web browser.

4.2 Package Contents

VICE: Versatile Integrator for Chemical Evolution

4.2.1 Provides

- A dataframe object meant for case-insensitive lookup
- Simulations of galactic chemical evolution models
- Simulations of nucleosynthesis from single stellar populations
- Built-in yield tables from nucleosynthesis studies

4.2.2 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`.

Example scripts can be found under examples/ in the git repository at http://github.com/giganano/VICE.

4.2.3 Contents

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

output [type] Read and store output from single- and multi-zone 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.

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 [type] An extension to the Python type `dict` to allow case-insensitivity.

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

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

4.2.4 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.

4.2.5 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's version_info

- major : The major version number
- minor : The minor version number
- micro : The micro version number (also known as patch number)
- build : The build number
- __version__ : The version string <major>.<minor>.<micro>
- released : If True, this version of VICE has been released

Note: This object can be type-cast to a tuple of the form: (major, minor, micro, build).

---

vice.atomic_number

The VICE dataframe: derived class (inherits from noncustomizable)

Stores persistent data for each element.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - Any (cannot be modified)

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

Built-In Instances

- **vice.atomic_number** The atomic number (protons only) of each element.
- **vice.primordial** The primordial abundance by mass Z of each element following big bang nucleosynthesis. This is zero for all elements with the exception of helium, which is assigned the standard model value of $Y_{\rm p} = 0.24672 \pm 0.00017$[1][2][3].

  New in version 1.1.0.
- **vice.solar_z** The abundance by mass of each element in the sun. This is adopted from Asplund et al. (2009)[4].
- **vice.sources** The dominant astrophysical enrichment channels of each element. This is adopted from Johnson (2019)[5].
- **vice.stable_isotopes** The mass number (protons + neutrons) of the stable isotopes of each element.

  New in version 1.1.0.

Example Code

```
>>> import vice
>>> vice.atomic_number['c']
        6
>>> vice.primordial['c']
        0
>>> vice.solar_z['c']
        0.00236
>>> vice.sources['c']
        ["CCSNE", "AGB"]
>>> vice.stable_isotopes['c']
        [12, 13]
```

Signature: vice.core.dataframe.builtin_elemental_data(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.

vice.primordial

The VICE dataframe: derived class (inherits from noncustomizable)

Stores persistent data for each element.

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

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - Any (cannot be modified)

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

Built-In Instances

- **vice.atomic_number** The atomic number (protons only) of each element.
- **vice.primordial** The primordial abundance by mass Z of each element following big bang nucleosynthesis. This is zero for all elements with the exception of helium, which is assigned the standard model value of $Y_{\rm p} = 0.24672 \pm 0.00017$[1][2][3].

  New in version 1.1.0.
- **vice.solar_z** The abundance by mass of each element in the sun. This is adopted from Asplund et al. (2009)[4].
- **vice.sources** The dominant astrophysical enrichment channels of each element. This is adopted from Johnson (2019)[5].
- **vice.stable_isotopes** The mass number (protons + neutrons) of the stable isotopes of each element.

  New in version 1.1.0.

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

Example Code

```
>>> import vice
>>> vice.atomic_number['c']
        6
>>> vice.primordial['c']
        0
>>> vice.solar_z['c']
        0.00236
>>> vice.sources['c']
        ["CCSNE", "AGB"]
>>> vice.stable_isotopes['c']
        [12, 13]
```

Signature: vice.core.dataframe.builtin_elemental_data(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.

vice.solar_z

The VICE dataframe: derived class (inherits from noncustomizable)

Stores persistent data for each element.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - Any (cannot be modified)

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

Built-In Instances

- **vice.atomic_number** The atomic number (protons only) of each element.
- **vice.primordial** The primordial abundance by mass Z of each element following big bang nucleosynthesis. This is zero for all elements with the exception of helium, which is assigned the standard model value of $Y_{\rm p} = 0.24672 \pm 0.00017$[1][2][3].

  New in version 1.1.0.
- **vice.solar_z** The abundance by mass of each element in the sun. This is adopted from Asplund et al. (2009)[4].
- **vice.sources** The dominant astrophysical enrichment channels of each element. This is adopted from Johnson (2019)[5].
- **vice.stable_isotopes** The mass number (protons + neutrons) of the stable isotopes of each element.

  New in version 1.1.0.

Example Code

```
>>> import vice
>>> vice.atomic_number['c']
        6
>>> vice.primordial['c']
        0
>>> vice.solar_z['c']
        0.00236
>>> vice.sources['c']
        ["CCSNE", "AGB"]
>>> vice.stable_isotopes['c']
        [12, 13]
```

Signature: vice.core.dataframe.builtin_elemental_data(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.

vice.sources

The VICE dataframe: derived class (inherits from noncustomizable)

Stores persistent data for each element.

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

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - Any (cannot be modified)

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

Built-In Instances

- **vice.atomic_number** The atomic number (protons only) of each element.
- **vice.primordial** The primordial abundance by mass Z of each element following big bang nucleosynthesis. This is zero for all elements with the exception of helium, which is assigned the standard model value of $Y_{\rm p} = 0.24672 \pm 0.00017$[1][2][3].

  New in version 1.1.0.
- **vice.solar_z** The abundance by mass of each element in the sun. This is adopted from Asplund et al. (2009)[4].
- **vice.sources** The dominant astrophysical enrichment channels of each element. This is adopted from Johnson (2019)[5].
- **vice.stable_isotopes** The mass number (protons + neutrons) of the stable isotopes of each element.

  New in version 1.1.0.

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

Example Code

```
>>> import vice
>>> vice.atomic_number['c']
        6
>>> vice.primordial['c']
        0
>>> vice.solar_z['c']
        0.00236
>>> vice.sources['c']
        ["CCSNE", "AGB"]
>>> vice.stable_isotopes['c']
        [12, 13]
```

Signature: vice.core.dataframe.builtin_elemental_data(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.

vice.stable_isotopes

The VICE dataframe: derived class (inherits from noncustomizable)

Stores persistent data for each element.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - Any (cannot be modified)

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

Built-In Instances

- **vice.atomic_number** The atomic number (protons only) of each element.
- **vice.primordial** The primordial abundance by mass Z of each element following big bang nucleosynthesis. This is zero for all elements with the exception of helium, which is assigned the standard model value of $Y_{\rm p} = 0.24672 \pm 0.00017$[1][2][3].

  New in version 1.1.0.
- **vice.solar_z** The abundance by mass of each element in the sun. This is adopted from Asplund et al. (2009)[4].
- **vice.sources** The dominant astrophysical enrichment channels of each element. This is adopted from Johnson (2019)[5].
- **vice.stable_isotopes** The mass number (protons + neutrons) of the stable isotopes of each element.

  New in version 1.1.0.

Example Code

```
>>> import vice
>>> vice.atomic_number['c']
        6
>>> vice.primordial['c']
        0
>>> vice.solar_z['c']
        0.00236
>>> vice.sources['c']
        ["CCSNE", "AGB"]
>>> vice.stable_isotopes['c']
        [12, 13]
```

Signature: vice.core.dataframe.builtin_elemental_data(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.

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_lower = 0.08, postMS = 0.01)

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

Parameters

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

IMF [`str` [case-insensitive] [default][“kroupa”]] The assumed stellar initial mass function (IMF). Strings denote built-in IMFs.

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.

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 $0.394M_\odot + 0.109M$.

---

[1] Kroupa (2001), MNRAS, 231, 322
[2] Salpeter (1955), ApJ, 121, 161
[3] Kalirai et al. (2008), ApJ, 676, 594

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

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] [default]["kroupa"]] The assumed stellar initial mass function (IMF). Strings denote built-in IMFs.

Recognized built-in IMFs:

- Kroupa[1]
- Salpeter[2]

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

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.

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 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.

IMF [`str` [case-insensitive] [default]["kroupa"]] The stellar initial mass function (IMF) to assume. Strings denote built-in IMFs.

Recognized built-in IMFs:

- Kroupa[1]
- Salpeter[2]

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

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

Note: Functions do not need to return 0 at times smaller than the SN Ia minimum delay time.

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

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] [default][“cristallo11”]] A keyword denoting which table of nucleosynthetic yields from AGB stars to adopt.

Recognized Keywords:

- “cristallo11”[3]
- “karakas10”[4]

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**

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

  - 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]**
  - 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.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")
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.builtin_elemental_data

The VICE dataframe: derived class (inherits from noncustomizable)

Stores persistent data for each element.

Allowed Data Types

- **Keys**
  - **`str` [case-insensitive]** [elemental symbols] The symbols of the elements as they appear on the periodic table.
- **Values**
  - Any (cannot be modified)

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

Built-In Instances

- **vice.atomic_number** The atomic number (protons only) of each element.
- **vice.primordial** The primordial abundance by mass Z of each element following big bang nucleosynthesis. This is zero for all elements with the exception of helium, which is assigned the standard model value of $Y_{\rm p} = 0.24672 \pm 0.00017$[1][2][3].

  New in version 1.1.0.
- **vice.solar_z** The abundance by mass of each element in the sun. This is adopted from Asplund et al. (2009)[4].
- **vice.sources** The dominant astrophysical enrichment channels of each element. This is adopted from Johnson (2019)[5].
- **vice.stable_isotopes** The mass number (protons + neutrons) of the stable isotopes of each element.

  New in version 1.1.0.

Example Code

```
>>> import vice
>>> vice.atomic_number['c']
        6
>>> vice.primordial['c']
        0
>>> vice.solar_z['c']
        0.00236
>>> vice.sources['c']
        ["CCSNE", "AGB"]
>>> vice.stable_isotopes['c']
        [12, 13]
```

Signature: vice.core.dataframe.builtin_elemental_data(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.

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

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.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.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)',
         '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{
```

(continues on next page)

(continued from previous page)

```
                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.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 one variable describing the yield as a function of metallicity Z. In this version of VICE, this is only applicable to CCSN 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
```

(continues on next page)

(continued from previous page)

```
                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.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

Contains

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

ccsne [<package>] Yields from core collapse supernovae

sneia [<package>] Yields from type Ia supernovae

presets [<package>] Yield settings presets

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

Notes

The yield tables built into VICE do not include any treatment of radioactive isotopes. Equations are evaluated and tables are returned counting only the total mass yield of stable isotopes. In the case of elements with a significant nucleosynthetic contribution from radioactive decay products, the values returned from the functions in this module should be interpreted as lower bounds rather than estimates of the true yield.

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]

Contents

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

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)[1]
- "karakas10": Karakas (2010)[2]

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.

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

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.
- **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

Note: The nucleosynthetic yield tables built into VICE do not include any treatment of radioactive isotopes. The yield tables returned by this function will not include what the specified study reported for radioactive isotopes. In the case of elements with a significant nucleosynthetic contribution from radioactive decay products, the values returned from this function should be interpreted as lower bounds rather than estimates of the true yield.

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.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]
- Chieffi & Limongi (2013)[2]
- Chieffi & Limongi (2004)[3]
- Woosley & Weaver (1995)[4]

[1] Limongi & Chieffi (2018), ApJS, 237, 13
[2] Chieffi & Limongi (2013), ApJ, 764, 21
[3] Chieffi & Limongi (2004), ApJ, 608, 405
[4] Woosley & Weaver (1995), ApJ, 101, 181

Contents

fractional [<function>] Calculate an IMF-averaged yield for a given element.

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

LC18 [yield preset] Sets yields according to the Limongi & Chieffi (2018) study.

CL13 [yield preset] Sets yields according to the Chieffi & Limongi (2013) study.

CL04 [yield preset] Sets yields according to the Chieffi & Limongi (2004) study.

WW95 [yield preset] Sets yields according to the Woosley & Weaver (1995) study.

vice.yields.ccsne.fractional

Calculate an IMF-integrated fractional net nucleosynthetic yield of a given element from core-collapse supernovae.

Signature: vice.yields.ccsne.fractional(element, study = "LC18", MoverH = 0, rotation = 0, IMF = "kroupa", method = "simpson", m_lower = 0.08, m_upper = 100, tolerance = 1.0e-03, 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]
- "CL13": Chieffi & Limongi (2013)[2]
- "CL04": Chieffi & Limongi (2004)[3]
- "WW95": Woosley & Weaver (1995)[4]

MoverH [real number [default][0]] The total metallicity [M/H] 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
- "CL13": [M/H] = 0
- "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
- "CL13": v = 0, 300

[1] Limongi & Chieffi (2018), ApJS, 237, 13
[2] Chieffi & Limongi (2013), ApJ, 764, 21
[3] Chieffi & Limongi (2004), ApJ, 608, 405
[4] Woosley & Weaver (1995), ApJ, 101, 181

- "CL04": v = 0
- "WW95": v = 0

IMF [`str` [case-insensitive] [default]["kroupa"]] The stellar initial mass function (IMF) to assume. Strings denote built-in IMFs, which must be either "Kroupa"[5] or "Salpeter"[6].

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)[7].

---

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 din't converge numerically.

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

[5] Kroupa (2001), MNRAS, 231, 322
[6] Salpeter (1955), ApJ, 121, 161
[7] Press, Teukolsky, Vetterling & Flannery (2007), Numerical Recipes, Cambridge University Press

  - 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.
  - 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.

Notes

This function evaluates the solution to the following equation.

$$y_x^{\text{CC}} \frac{\int_8^u m_x \frac{dN}{dm} dm}{\int_l^u m_x \frac{dN}{dm} dm}$$

Note: The nucleosynthetic yield tables built into VICE do not include any treatment of radioactive isotopes. The above equation is evaluated directly from the total mass yield of stable isotopes only. In this regard, if any element has a significant contribution to its nucleosynthesis from radioactive decay products, then the values returned from this function should be interpreted as lower bounds rather than estimates of the true nucleosynthetic yield.

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.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. In this version of VICE, this is only allowed for CCSN yields.

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.yields.ccsne.settings.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.yields.ccsne.settings.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.yields.ccsne.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.yields.ccsne.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.yields.ccsne.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.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 $M_\odot$.

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.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)

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.

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import WW95
>>> WW95.set_params(m_lower = 0.3, m_upper = 45, IMF = "salpeter")
```

See also:

vice.yields.ccsne.fractional

[1] Woosley & Weaver (1995), ApJ, 101, 181

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 $M_\odot$.

Tip: By importing this module, the user does not sacrifice the ability to specify their yield settings directly.

Note: [M/H] = 0.15 corresponds to Z = 0.02 if the solar abundance is Z = 0.014 (Asplund et al. 2009)[1].

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.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)

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.

[1] Asplund et al. (2009), ARA&A, 47, 481
[1] Chieffi & Limongi (2004), ApJ, 608, 405

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import CL04
>>> CL04.set_params(m_lower = 0.3, m_upper = 40, IMF = "salpeter")
```

See also:

vice.yields.ccsne.fractional

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 $M_\odot$.

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.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)

Parameters

kwargs [varying types] Keyword arguments to pass to vice.yields.ccsne.fractional.

[1] Chieffi & Limongi (2013), ApJ, 764, 21

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.

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import CL13
>>> CL13.set_params(m_lower = 0.3, m_upper = 40, IMF = "salpeter")
```

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 $M_\odot$.

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.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)

[1] Limongi & Chieffi (2018), ApJS, 237, 17

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.

Example Code

```
>>> import vice
>>> from vice.yields.ccsne import LC18
>>> LC18.set_params(m_lower = 0.3, m_upper = 40, IMF = "salpeter")
```

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]

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.

seitenzahl13 [yield preset] Sets the yields according to the Seitenzahl et al. (2013) study.

iwamoto99 [yield preset] Sets the yields according to the Iwamoto et al. (1999) study.

[1] Seitenzahl et al. (2013), MNRAS, 429, 1156
[2] Iwamoto et al. (1999), ApJ, 124, 439

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]

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

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.

[1] Seitenzahl et al. (2013), MNRAS, 429, 1156
[2] Iwamoto et al. (1999), ApJ, 124, 439

Notes

Note: The nucleosynthetic yield tables built into VICE do not include any treatment of radioactive isotopes. The mass yield of the given element will be reported as the sum of stable isotopes only. In the case of elements with a significant nucleosynthetic contribution from radioactive decay products, the values returned from this function should be interpreted as lower bounds rather than estimates of the true yield.

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("ni", model = "n100l")
        0.0391409000000526
```

See also:

vice.yields.sneia.fractional

vice.yields.sneia.fractional

Calculate an IMF-averaged 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]

model [str [case-insensitive] [default]["N1"]] The 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

n [real number [default][2.2e-03]] The average number of type Ia supernovae produced per unit stellar mass formed $N_{\rm Ia}/M_\star$ in $M_\odot^{-1}$.

[1] Seitenzahl et al. (2013), MNRAS, 429, 1156
[2] Iwamoto et al. (1999), ApJ, 124, 439

Note: The default value for this parameter is adopted from Maoz & Mannucci (2012)[3].

Returns

y [real number] The IMF-averaged yield.

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.

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`.

Note: The nucleosynthetic yield tables built into VICE do not include any treatment of radioactive isotopes. This function evaluates the solution to the above equation given the total mass yield of stable isotopes only. In the case of elements with a significant nucleosynthetic contribution from radioactive decay products, the values returned from this function should be interpreted as lower bounds rather than estimates of the true yield.

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

Example Code

```
>>> import vice
>>> vice.fractional_ia_yield("fe")
        0.0025825957080000002
>>> vice.fractional_ia_yield("ca")
        8.935489894764334e-06
>>> vice.fractional_ia_yield("ni")
        0.00016576890932800003
```

vice.yields.sneia.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. In this version of VICE, this is only allowed for CCSN yields.

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.yields.sneia.settings.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.yields.sneia.settings.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.yields.sneia.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.yields.sneia.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.yields.sneia.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.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 IMF-averaged yields calculated with the Iwamoto et al. (1999) yield table under the W70 explosion model. This study is for Chandrasekhar Mass progenitors (1.4 $M_\odot$).

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.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.

Example Code

```
>>> from vice.yields.sneia import iwamoto99
>>> iwamoto99.set_params(n = 1.5e-03)
```

See also:

- vice.yields.sneia.fractional
- vice.yields.sneia.single

vice.yields.sneia.seitenzahl13

Seitenzahl et al. (2013), MNRAS, 429, 1156 typa 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 IMF-averaged yields calculated with the Seitenzahl et al. (2013) yield table under the N1 explosion model. This study is for delayed detonation explosion models of Chandrasekhar mass progenitors (1.4 $M_\odot$).

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.

[1] Iwamoto et al. (1999), ApJ, 124, 439

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(**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.

See also:

vice.yields.sneia.fractional

Example Code

```
>>> from vice.yields.sneia import seitenzahl13
>>> seitenzahl13.set_params(n = 1.5e-03)
```

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.

---

[1] Seitenzahl et al. (2013), ApJ, 124, 439

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

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:

[1] Johnson & Weinberg (2020), arxiv:1911.02598

```
>>> 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), arxiv:1911.02598 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

- $y_{\rm O}^{\rm CC} = 0.015$
- $y_{\rm Fe}^{\rm CC} = 0.0012$
- $y_{\rm Sr}^{\rm CC} = 3.5 \times 10^{-8}$

SNe Ia

- $y_{\rm O}^{\rm Ia} = 0$
- $y_{\rm Fe}^{\rm Ia} = 0.0017$
- $y_{\rm Sr}^{\rm Ia} = 0$

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.

[1] Cristallo et al. (2011), ApJS, 197, 17
[1] Johnson & Weinberg (2020), arxiv:1911.02598

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:

$$y_{\rm Sr}^{\rm CC} = 3.5 \times 10^{-8} \left(\frac{Z}{Z_\odot}\right)$$

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].

[2] Asplund et al. (2009), ARA&A, 47, 481
[1] Johnson & Weinberg (2020), arxiv:1911.02598
[2] Asplund et al. (2009), ARA&A, 47, 481

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
```

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
        }
>>> example = vice.elements.element("sr")
```

(continues on next page)

(continued from previous page)

```
>>> 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
        }
>>> 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
        }
```

See also:

- vice.yields
- vice.atomic_number
- vice.primordial
- vice.solar_z
- vice.sources
- vice.stable_isotopes

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.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
        }
>>> example = vice.elements.element("sr")
```

(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)

```
>>> 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
        }
>>> 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
        }
```

vice.elements.element.symbol

Type : `str`

The one- or two-letter symbol of this element as it appears on the periodic table.

New in version 1.1.0.

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.

New in version 1.1.0.

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.

New in version 1.1.0.

Attributes

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.sneia
        0.00258
>>> 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.

New in version 1.1.0.

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.

New in version 1.1.0.

[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

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 1.1.0.

Example Code

```
>>> import vice
>>> vice.elements.Fe.solar_z
        0.00129
>>> vice.elements.O.solar_z
        0.00572
```

vice.elements.element.sources

Type : `list` of strings

Strings denoting the dominant sources of enrichment for this element as reported by Johnson (2019)[1].

New in version 1.1.0.

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.

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

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.

Parameters

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

Attributes

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.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.

New in version 1.1.0.

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
```

vice.elements.yields.sneia

Type : real number

The current yield setting for type Ia supernovae (SNe Ia). Interpreted as a constant, metallicity-independent yield.

New in version 1.1.0.

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
```

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.

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:

$$\frac{dN}{dm} \sim m^{-\alpha}$$

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
```

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

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.

Returns

dndm [real number] The unnormalized value of the Salpeter IMF at that stellar mass, defined by:

$$\frac{dN}{dm} \sim m^{-\alpha}$$

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: Signature: vice.singlezone(**kwargs)

[1] Salpeter (1955), ApJ, 121, 161

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]

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”.

[1] Kroupa (2001), MNRAS, 231, 322
[2] Salpeter (1955), ApJ, 121, 161
[3] Johnson & Weinberg (2020), arxiv:1911.02598

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. This changes when the attribute `schmidt` == True.

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.

schmidt_index [real number [default][0.5]] The power-law index of gas-dependent star formation efficiency.

MgSchmidt [real umber [default][6.0e+09]] The normalization of the gas-supply when the attribute `schmidt` == True.

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 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][“cristallo11”]] A keyword denoting which table of nucleosynthetic yields from AGB stars to adopt.

Recognized Keywords:

- “cristallo11”[4]
- “karakas10”[5]

Functions

run [[instancemethod]] Run the simulation

from_output [[classmethod]] Obtain a `singlezone` object with the parameters of the one that produced an output.

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
```

(continues on next page)

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

(continued from previous page)

```
                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.

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: 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
                Zin ------------> 0.0
                recycling ------> continuous
                delay ----------> 0.15
                RIa ------------> plaw
                Mg0 ------------> 6000000000.0
                smoothing ------> 0.0
```

(continues on next page)

(continued from previous page)

```
                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 >= 1.2.0 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[2].

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.elements
("fe", "sr", "o")
>>> sz.elements = ["mg", "fe", "c", "n", "o"]
>>> sz.elements
("mg", "fe", "c", "n", "o")
```

vice.singlezone.IMF

Type : `str` [case-insensitive]

Default : "kroupa"

The assumed stellar initial mass function (IMF). Strings denote built-in IMFs.

Built-in IMFs:

- "kroupa"[1]
- "salpeter"[2]

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.IMF = "kroupa"
>>> sz.IMF = "salpeter"
```

[1] Johnson (2019), Science, 363, 474
[2] Johnson & Weinberg (2020), arxiv:1911.02598
[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.

$$\eta \equiv \frac{\dot{M}_{\text{out}}}{\dot{M}_*}$$

Note: If the attribute smoothing is nonzero, this relationship generalizes to

$$\dot{M}_{\text{out}} = \eta(t)\langle\dot{M}_*\rangle_{\tau_s} = \begin{cases} \frac{\eta(t)}{t}\int_0^t \dot{M}_*(t')dt' & (t < \tau_s) \\ \frac{\eta(t)}{\tau_s}\int_{t-\tau_s}^t \dot{M}_*(t')dt' & (t \geq \tau_s) \end{cases}$$

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.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: Inflow masses due to primordial abundances and metal-rich infall are treated independently of one another. For this reason, if a helium-rich infall is required, the difference between the desired helium abundance and the primordial abundance ΔY should be specified as opposed to the total abundance.

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
```

(continues on next page)

(continued from previous page)

```
        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"
```

[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

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).

Example Code

```
>>> import numpy as np
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> # 400 bins between 0 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

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 user's 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 initla 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 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 $\dot{M}_*$ with a smoothing time $\tau_{\rm s}$:

$$\dot{M}_{\rm out} = \eta(t)\langle\dot{M}_*\rangle_{\tau_{\rm s}}$$

[1] Johnson & Weinberg (2020), arxiv:1911.02598

The traditional relationship of $\dot{M}_{\rm out} = \eta\dot{M}_*$ 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

$$\tau_* \equiv M_{\rm g}/\dot{M}_*$$

where $M_{\rm g}$ is the ISM gas mass and $\dot{M}_*$ is the star formation rate. Numbers will be interpreted as a constant value. Functions must accept time in Gyr as the only parameter.

Tip: In infall and gas modes, this parameter can be set to infinity to forcibly shut off star formation.

Note: When the attribute `schmidt == True`, this is interpreted as the prefactor on gas-dependent star formation efficiency:

$$\tau_*^{-1} = \tau_{*,\text{specified}}^{-1} \left(\frac{M_\text{g}}{M_{\text{g,Schmidt}}} \right)^\alpha$$

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 usually referred to as the "star formation efficiency timescale."

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 τ_*. At each timestep, the star formation efficiency timescale is determined via:

$$\tau_*(t) = \tau_{*,\text{specified}}(t) \left(\frac{M_g}{M_{g,\text{Schmidt}}} \right)^{-\alpha}$$

where $\tau_{*,\text{specified}}(t)$ is the value of the attribute `tau_star`, M_g is the mass of the interstellar medium, $M_{g,\text{Schmidt}}$ the normalization thereof (attribute `MgSchmidt`), and α 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.

See also:

- vice.singlezone.tau_star
- vice.singlezone.schmidt_index
- vice.singlezone.MgSchmidt

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:

$$\tau_*^{-1} \sim M_g^\alpha$$

[1] Kennicutt (1998), ApJ, 498, 541
[2] Schmidt (1959), ApJ, 129, 243
[3] Schmidt (1963), ApJ, 137, 758

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:

$$\tau_* \sim \left(\frac{M_g}{M_{g,\text{Schmidt}}}\right)^{-\alpha}$$

where α is specified by the attribute `schmidt_index`.

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 solar masses.

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.

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:

$$Z_{\text{ISM}} = Z_\odot \left[\sum_i Z_i\right] \left[\sum_i Z_i^\odot\right]^{-1}$$

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 adopts the Asplund et al. (2009) measurements on their element-by-element basis in calculating [X/H] and [X/Y] in simulations; it is thus recommended that users adopt these measurements as well so that the adopted solar composition is self-consistent. This however has no qualitative impact on the behavior of the simulation.

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example")
>>> sz.Z_solar = 0.014
```

vice.singlezone.agb_model

Type : `str` [case-insensitive]

Default : "cristallo11"

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.

---

[1] Asplund et al. (2009), ARA&A, 47, 481
[1] Cristallo et al. (2011), ApJS, 197, 17
[2] Karakas (2010), MNRAS, 403, 1413

Example Code

```
>>> import vice
>>> sz = vice.singlezone(name = "example", elements = ["c", "n", "o"])
>>> sz.agb_model = "karakas10"
>>> sz.agb_model = "cristallo11"
```

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,
```

(continues on next page)

(continued from previous page)

```
         -0.30623330628476564,
         -0.30631559283107557,
         -0.3063978595147838,
         -0.30647984166504416,
         -0.3065618040838354]
>>> 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
        }
```

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]
```

(continues on next page)

(continued from previous page)

```
        [-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
        }
```

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.

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.

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

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
                sfr -----------> 2.897559
                ifr -----------> 9.1
```

(continues on next page)

(continued from previous page)

```
                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
                dn/d[sr/fe] ----> 0.06001488
                dn/d[o/fe] -----> 0.4337209
                dn/d[o/sr] -----> 0.0
        }
```

vice.output.ccsne_yields

Type : `dataframe`

The core-collapse supernova yields employed in the simulation.

Note: This dataframe will not be 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 will not be 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 visualization and inspection of simulation output 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.mirror

[DEPRECATED]

Obtain an instance of the vice.singlezone 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 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`] A new `singlezone` object, with the same parameters as that which produced the output.

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
```

(continues on next page)

(continued from previous page)

```
        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.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 this version of VICE in research should cite Johnson & Weinberg (2020). If you're using `BibTeX`, add the following to your `.bib` file:

```
@ARTICLE{2019arXiv191102598J,
       author = {{Johnson}, James W. and {Weinberg}, David H.},
        title = "{The Impact of Starbursts on Element Abundance Ratios}",
      journal = {arXiv e-prints},
     keywords = {Astrophysics - Astrophysics of Galaxies},
         year = 2019,
        month = nov,
          eid = {arXiv:1911.02598},
        pages = {arXiv:1911.02598},
archivePrefix = {arXiv},
       eprint = {1911.02598},
 primaryClass = {astro-ph.GA},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2019arXiv191102598J},
```

(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

Primary Author
Email: giganano9@gmail.com
Webiste: https://sites.google.com/view/jameswjohnson/
The Ohio State University Department of Astronomy
140 W. 18th Ave., Columbus, OH, 43204

5.3.2 David H. Weinberg

Advising Author
Website: http://www.astronomy.ohio-state.edu/~dhw/
The Ohio State University Department of Astronomy
140 W. 18th Ave., Columbus, OH, 43204

5.4 Acknowledgements

J.W.J. is grateful to D.H.W. and Jennifer A. Johnson at The Ohio State University for continual guidance in galactic chemical evolution modeling. J.W.J. also acknowledges the valuable discussion on the implementation of the cumulative return fraction contributed by Jenna Freudenburg at The Ohio State University. Construction of this software was supported in part by an Ohio State University Graduate Fellowship.

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 `singlezone` object 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` object, the `dataframe` and all derived classes, the `output` object, and single stellar population routines in the `vice/core/ssp/` subdirectory. The hierarchical file structure of these directories is designed to mirror one another. Separate from the VICE `core` is

the `yields` module, in which all nucleosynthetic yield calculations are implemented, independent of the simulation features.

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 add any necessary routines in the fork. These changes should reflect the overall design of the package: with all C extensions in `vice/src/`; deviating from this pattern will cause a broken import following installation of the modified version of the code. Unless the modification is to the `vice/yields/` module, the python wrapping of these functions should be in `vice/core/`.

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 functions via `decorators`. Place `@unittest` before a function returning a string describing the dot-notation path to the function and the unit test function itself to obtain a `unittest` object. Similarly, place `@moduletest` before a function return a string describing the dot-notation path to the module and a list of `unittest` and `moduletest` objects to obtain a `moduletest` object. Finally, link the tests to that of the parent directory's `moduletest` object.

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/users_guide/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*.