

---

# Chempy Documentation

*Release 0.1*

**J. Rybizki**

**Jul 13, 2018**



---

## Contents

---

<b>1</b>	<b>Package main content</b>	<b>3</b>
1.1	Contents . . . . .	3
1.1.1	Chempy . . . . .	3
1.2	Installation . . . . .	31
<b>2</b>	<b>Quick Start</b>	<b>33</b>
2.1	References . . . . .	33
<b>3</b>	<b>Indices and tables</b>	<b>35</b>
	<b>Python Module Index</b>	<b>37</b>



This package is a flexible elemental abundance fitting tool for stellar abundances. It takes nucleosynthetic yield tables from literature, integrates them over the IMF for simple stellar (SSP) populations and uses successive SSPs to integrate the chemical evolution of an open-box one-zone inter stellar medium (ISM) over time.



---

## Package main content

---

This package is mostly organized around:

- `Chempy.wrapper.SSP_wrap` that handles the SSP enrichment.
- `Chempy.wrapper.Chempy()` that handles the time-integration.

Additionally, and for convenience

- `Chempy.wrapper.mcmc()` can be used to infer most likely parameters.

To get started with using Chempy it is strongly recommended to browse through the extensive [tutorial](#).

## 1.1 Contents

### 1.1.1 Chempy

#### Chempy package

#### Submodules

#### Chempy.cem\_function module

`Chempy.cem_function.cem(changing_parameter, a)`

This is the function calculating the chemical evolution for a specific parameter set (`changing_parameter`) and for a specific observational constraint specified in `a` (e.g. 'solar\_norm' calculates the likelihood of solar abundances coming out of the model). It returns the posterior and a list of blobs. It can be used by an MCMC. This function actually encapsulates the real cem function in order to capture exceptions and in that case return -inf. This makes the MCMC runs much more stable

INPUT:

`changing_parameter` = parameter values of the free parameters as an array

a = model parameters specified in parameter.py. There are also the names of free parameters specified here

OUTPUT:

log posterior, array of blobs

the blobs contain the prior values, the likelihoods and the actual values of each predicted data point (e.g. elemental abundance value)

`Chempy.cem_function.cem2(a)`

This is the function calculating the chemical evolution for a specific parameter set (`changing_parameter`) and for a specific observational constraint specified in `a` (e.g. 'solar\_norm' calculates the likelihood of solar abundances coming out of the model). It returns the posterior and a list of blobs. It can be used by an MCMC. This function actually encapsulates the real cem function in order to capture exceptions and in that case return -inf. This makes the MCMC runs much more stable

INPUT:

a = model parameters specified in parameter.py and altered by `posterior_function`

OUTPUT:

predictions, name\_of\_prediction

the predicted element abundances for the time of the birth of the star (specified in `a`) are given back, as well as the corona metallicity at that time and the SN-ratio at that time.

`Chempy.cem_function.cem_real(changing_parameter, a)`

real chempy function. description can be found in `cem`

`Chempy.cem_function.cem_real2(a)`

real chempy function. description can be found in `cem2`

`Chempy.cem_function.extract_parameters_and_priors(changing_parameter, a)`

This function extracts the parameters from changing parameters and writes them into the `ModelParameters(a)`, so that Chempy can evaluate the changed parameter settings

`Chempy.cem_function.gaussian(x, x0, xsig)`

function to calculate the gaussian probability (its normed to Pmax and given in log)

INPUT:

x = where is the data point or parameter value

x0 = mu

xsig = sigma

`Chempy.cem_function.gaussian_log(x, x0, xsig)`

function to calculate the gaussian probability (its normed to Pmax and given in log)

INPUT:

x = where is the data point or parameter value

x0 = mu

xsig = sigma

`Chempy.cem_function.get_prior(changing_parameter, a)`

This function calculates the prior probability

INPUT:



changing\_parameter = the values of the parameter vector

a = the model parameters including the names of the parameters (which is needed to identify them with the prescribed priors in parameters.py)

OUTPUT:

the log prior is returned

`Chempy.cem_function.global_optimization(changing_parameter, result)`

This function is a buffer function if `global_optimization_real` fails and it only returns the negative posterior

`Chempy.cem_function.global_optimization_error_returned(changing_parameter, result)`

this is a buffer function preventing failures from `global_optimization_real` and returning all its output including the best model error

`Chempy.cem_function.global_optimization_real(changing_parameter, result)`

This function calculates the predictions from several Chempy zones in parallel. It also calculates the likelihood for common model errors BEWARE: Model parameters are called as saved in parameters.py!!!

INPUT:

changing\_parameter = the global SSP parameters (parameters that all stars share)

result = the complete parameter set is handed over as an array of shape(len(stars),len(all parameters)).  
From those the local ISM parameters are taken

OUTPUT:

-posterior = negative log posterior for all stellar zones

error\_list = the optimal standard deviation of the model error

elements = the corresponding element symbols

`Chempy.cem_function.lognorm(x, mu, factor)`

this function provides Prior probability distribution where the factor away from the mean behaves like the sigma deviation in normal\_log BEWARE: this function is not a properly normalized probability distribution. It only provides relative values.

INPUT:

x = where to evaluate the function, can be an array

mu = peak of the distribution

factor = the factor at which the probability decreases to 1 sigma

Can be used to specify the prior on the yield factors

`Chempy.cem_function.lognorm_log(x, mu, factor)`

this function provides Prior probability distribution where the factor away from the mean behaves like the sigma deviation in normal\_log

for example if mu = 1 and factor = 2

for 1 it returns 0

for 0.5 and 2 it returns -0.5

for 0.25 and 4 it returns -2.0

and so forth

Can be used to specify the prior on the yield factors

`Chempy.cem_function.posterior_function` (*changing\_parameter, a*)

The posterior function is the interface between the optimizing function and Chempy. Usually the likelihood will be calculated with respect to a so called 'stellar wildcard'. Wildcards can be created according to the tutorial 6. A few wildcards are already stored in the input folder. Chempy will try the current folder first. If no wildcard npy file with the name `a.stellar_identifier` is found it will look into the `Chempy/input/stars` folder.

INPUT:

`changing_parameter` = parameter values of the free parameters as an array

`a` = model parameters specified in `parameter.py`. There are also the names of free parameters specified here

OUTPUT:

log posterior, array of blobs

the blobs contain the likelihoods and the actual values of each predicted data point (e.g. elemental abundance value)

`Chempy.cem_function.posterior_function_for_minimization` (*changing\_parameter, a*)

calls the posterior function but just returns the negative log posterior instead of posterior and blobs

`Chempy.cem_function.posterior_function_local` (*changing\_parameter, stellar\_identifier, global\_parameters, errors, elements*)

The posterior function is the interface between the optimizing function and Chempy. Usually the likelihood will be calculated with respect to a so called 'stellar wildcard'. Wildcards can be created according to the tutorial 6 from the github page. A few wildcards are already stored in the input folder. Chempy will try the current folder first. If no wildcard npy file with the name `a.stellar_identifier` is found it will look into the `Chempy/input/stars` folder.

INPUT:

`changing_parameter` = parameter values of the free parameters as an array

`a` = model parameters specified in `parameter.py`. There are also the names of free parameters specified here

`global_parameters` = the SSP Parameters which are fixed for this optimization but need to be handed over to Chempy anyway

`errors` = the model error for each element

`elements` = the corresponding names of the elements

OUTPUT:

log posterior, array of blobs

the blobs contain the actual values of each predicted data point (e.g. elemental abundance value)

`Chempy.cem_function.posterior_function_local_for_minimization` (*changing\_parameter, stellar\_identifier, global\_parameters, errors, elements*)

calls the local posterior function but just returns the negative log posterior instead of posterior and blobs

`Chempy.cem_function.posterior_function_local_real` (*changing\_parameter, stellar\_identifier, global\_parameters, errors, elements*)

This is the actual posterior function. But the functionality is explained in `posterior_function`.

`Chempy.cem_function.posterior_function_many_stars` (*changing\_parameter, error\_list, elements*)

The posterior function is the interface between the optimizing function and Chempy. Usually the likelihood

will be calculated with respect to a so called ‘stellar wildcard’. Wildcards can be created according to the tutorial 6. A few wildcards are already stored in the input folder. Chempy will try the current folder first. If no wildcard npy file with the name a.stellar\_identifier is found it will look into the Chempy/input/stars folder. The posterior function for many stars evaluates many Chempy instances for different stars and adds up their common likelihood. The list of stars is given in parameter.py under stellar\_identifier\_list. The names in the list must be represented by wildcards in the same folder.

INPUT:

changing\_parameter = parameter values of the free parameters as an array

error\_list = the model error list for each element

elements = the corresponding element symbols

OUTPUT:

log posterior, array of blobs

the blobs contain the likelihoods and the actual values of each predicted data point (e.g. elemental abundance value)

`Chempy.cem_function.posterior_function_many_stars_real(changing_parameter, error_list, error_element_list)`

This is the actual posterior function for many stars. But the functionality is explained in posterior\_function\_many\_stars.

`Chempy.cem_function.posterior_function_predictions(changing_parameter, a)`

This is like posterior\_function\_real. But returning the predicted elements as well.

`Chempy.cem_function.posterior_function_real(changing_parameter, a)`

This is the actual posterior function. But the functionality is explained in posterior\_function.

`Chempy.cem_function.posterior_function_returning_predictions(args)`

calls the posterior function but just returns the negative log posterior instead of posterior and blobs

`Chempy.cem_function.shorten_sfr(a)`

This function crops the SFR to the length of the age of the star and ensures that enough stars are formed at the stellar birth epoch

INPUT:

a = Modelparameters

OUTPUT:

the function will update the modelparameters, such that the simulation will end when the star is born and it will also check whether there is enough sfr left at that epoch

## Chempy.data\_to\_test module

`Chempy.data_to_test.arcturus(summary_pdf, name_string, abundances, cube, elements_to_trace, element_names, sol_table, number_of_models_overplotted, arcturus_age, produce_mock_data, use_mock_data, error_inflation)`

This is a likelihood function for the arcturus abundances compared to the model ISM abundances from some time ago (the age of arcturus which needs to be specified). The abundances are taken from Ramirez+ 2011 and all elements except for Fe are given in [X/Fe]

INPUT:

summary\_pdf = boolean, should a pdf be created?

name\_string = string to be added in the saved file name

abundances = abundances of the IMS (can be calculated from cube)

cube = the ISM mass fractions per element (A Chempy class containing the model evolution)

elements\_to\_trace = Which elements should be used for the analysis

sol\_table = solar abundance class

number\_of\_models\_overplotted = default is 1, if more the results will be saved and in the last iteration all former models will be plotted at once

produce\_mock\_data = should the predictions be saved with an error added to them (default = False)

use\_mock\_data = instead of the real data use the formerly produced mock data (default = False)

error\_inflation = a factor with which the mock data should be perturbed with the observational data (default = 1.0)

#### OUTPUT:

probabilities = each elements likelihood in a list

model\_abundances = each elements model prediction as a list

elements\_list = the names of the elements in a list

```
Chempy.data_to_test.cosmic_abundance_standard(summary_pdf, name_string, abundances, cube, elements_to_trace, solar, number_of_models_overplotted, produce_mock_data, use_mock_data, error_inflation)
```

This is a likelihood function for the B-stars (a proxy for the present-day ISM) with data from nieva przybilla 2012 cosmic abundance standard paper.

#### INPUT:

summary\_pdf = boolean, should a pdf be created?

name\_string = string to be added in the saved file name

abundances = abundances of the IMS (can be calculated from cube)

cube = the ISM mass fractions per element (A Chempy class containing the model evolution)

elements\_to\_trace = Which elements should be used for the analysis

solar = solar abundance class

number\_of\_models\_overplotted = default is 1, if more the results will be saved and in the last iteration all former models will be plotted at once

produce\_mock\_data = should the predictions be saved with an error added to them (default=False)

use\_mock\_data = instead of the real data use the formerly produced mock data (default = False)

error\_inflation = a factor with which the mock data should be perturbed with the observational data (default = 1.0)

#### OUTPUT:

probabilities = each elements likelihood in a list

model\_abundances = each elements model prediction as a list

elements\_list = the names of the elements in a list

`Chempy.data_to_test.elements_plot` (*name\_string*, *agb*, *sn2*, *sn1a*, *elements\_to\_trace*,  
*all\_elements*, *max\_entry*)

This function plots the available elements for specific yield sets.

INPUT:

*name\_string* = a string that will be added to the file name  
*agb* = agb yield class  
*sn2* = sn2 yield class  
*sn1a* = sn1a yield class  
*elements\_to\_trace* = which elements do we want to follow (a list)  
*all\_elements* = Symbols of all elements (available in the solar abundances class)  
*max\_entry* = until which element number the figure should be plotted

OUTPUT:

a figure in the current directory

`Chempy.data_to_test.fractional_yield_comparison_plot` (*yield\_name1*, *yield\_name2*,  
*yield\_class*, *yield\_class2*,  
*solar\_class*, *element*)

a function to plot a comparison between the fractional yield of two yield sets. The fractional yield is the mass fraction of the star that is expelled as the specific element. Depending on the yield class it will be net or total yield.

INPUT

*yield\_name1* = Name of the first yield class  
*yield\_name2* = Name of the second yield class  
*yield\_class* = a Chempy yield class (e.g 'Nomoto2013' from SN2\_Feedback, see tutorial)  
*yield\_class2* = a Chempy yield class (e.g 'Nomoto2013' from SN2\_Feedback, see tutorial)  
*solar\_class* = a Chempy solar class (needed for normalisation)  
*element* = the element for which to plot the yield table

OUTPUT

the figure will be saved into the current directory

`Chempy.data_to_test.gas_reservoir_metallicity` (*summary\_pdf*, *name\_string*, *abundances*,  
*cube*, *elements\_to\_trace*, *gas\_reservoir*,  
*number\_of\_models\_overplotted*, *produce\_mock\_data*, *use\_mock\_data*,  
*error\_inflation*, *solZ*)

This is a likelihood function for the present-day metallicity of the corona gas. Data is taken from the Smith cloud

INPUT:

*summary\_pdf* = boolean, should a pdf be created?  
*name\_string* = string to be added in the saved file name  
*abundances* = abundances of the IMS (can be calculated from cube)  
*cube* = the ISM mass fractions per element (A Chempy class containing the model evolution)  
*elements\_to\_trace* = Which elements should be used for the analysis

gas\_reservoir = the gas\_reservoir class containing the gas\_reservoir evolution

number\_of\_models\_overplotted = default is 1, if more the results will be saved and in the last iteration all former models will be plotted at once

produce\_mock\_data = should the predictions be saved with an error added to them (default=False)

use\_mock\_data = instead of the real data use the formerly produced mock data (default = False)

error\_inflation = a factor with which the mock data should be perturbed with the observational data (default = 1.0)

solZ = solar metallicity

OUTPUT:

probabilities = each elements likelihood in a list

model\_abundances = each elements model prediction as a list

elements\_list = the names of the elements in a list

Chempy.data\_to\_test.gaussian(*x*, *x0*, *xsig*)

function to calculate the gaussian probability (its normed to Pmax and given in log)

INPUT:

*x* = where is the data point or parameter value

*x0* = mu

*xsig* = sigma

Chempy.data\_to\_test.gaussian\_1d\_log(*x*, *x0*, *xsig*)

Chempy.data\_to\_test.likelihood\_evaluation(*model\_error*, *star\_error\_list*, *abundance\_list*,  
*star\_abundance\_list*)

This function evaluates the Gaussian for the prediction/observation comparison and returns the resulting log likelihood. The model error and the observed error are added quadratically

INPUT:

*model\_error* = the error coming from the models side

*star\_error\_list* = the error coming from the observations

*abundance\_list* = the predictions

*star\_abundance\_list* = the observations

OUTPUT:

likelihood = the summed log likelihood

Chempy.data\_to\_test.likelihood\_function(*stellar\_identifier*, *list\_of\_abundances*, *elements\_to\_trace*, *\*\*keyword\_parameters*)

This function calculates analytically an optimal model error and the resulting likelihood from the comparison of predictions and observations

INPUT:

*list\_of\_abundances* = a list of the abundances coming from Chempy

*elements\_to\_trace* = a list of the elemental symbols

OUTPUT:

likelihood = the added log likelihood

element\_list = the elements that were in common between the predictions and the observations, has the same sequence as the following arrays

model\_error = the analytic optimal model error

star\_error\_list = the observed error

abundance\_list = the predictions

star\_abundance\_list = the observations

`Chempy.data_to_test.plot_abundance_wildcard` (*stellar\_identifier, wildcard, abundance\_list, element\_list, probabilities, time\_model*)

this function plots the abundances of a stellar wildcard and the abundances of a chempy model together with the resulting likelihood values

INPUT:

stellar\_identifier = str, name of the Star

wildcard = the wildcard recarray

abundance\_list = the abundance list from the model

element\_list = the corresponding element symbols

probabilities = the corresponding single likelihoods as a list

time\_model = the time of the chempy model which is compared to the stellar abundance (age of the star from the wildcard)

OUTPUT:

This saves a png plot to the current directory

`Chempy.data_to_test.plot_processes` (*summary\_pdf, name\_string, sn2\_cube, sn1a\_cube, agb\_cube, elements, cube1, number\_of\_models\_overplotted*)

This is a plotting routine showing the different nucleosynthetic contributions to the individual elements.

INPUT:

summary\_pdf = boolean, should a pdf be created?

name\_string = string to be added in the saved file name

sn2\_cube = the sn2\_feedback class

sn1a\_cube = the sn1a\_feedback class

agb\_cube = the agb feedback class

elements = which elements should be plotted

cube1 = the ISM mass fractions per element (A Chempy class containing the model evolution)

number\_of\_models\_overplotted = default is 1, if more the results will be saved and in the last iteration all former models will be plotted at once

OUTPUT:

A plotfile in the current directory

`Chempy.data_to_test.produce_wildcard_stellar_abundances` (*stellar\_identifier*,  
*age\_of\_star*, *sigma\_age*,  
*element\_symbols*, *el-*  
*ement\_abundances*,  
*element\_errors*)

This produces a structured array that can be used by the Chempy wildcard likelihood function.

INPUT:

*stellar\_identifier* = name of the files  
*age\_of\_star* = age of star in Gyr  
*sigma\_age* = the gaussian error of the age (so far not implemented in the likelihood function)  
*element\_symbols* = a list of the element symbols  
*element\_abundances* = the corresponding abundances in [X/Fe] except for Fe where it is [Fe/H]  
*element\_errors* = the corresponding gaussian errors of the abundances

OUTPUT:

it will produce a .npy file in the current directory with *stellar\_identifier* as its name.

`Chempy.data_to_test.ratio_function` (*summary\_pdf*, *name\_string*, *abundances*,  
*cube*, *elements\_to\_trace*, *gas\_reservoir*, *num-*  
*ber\_of\_models\_overplotted*, *produce\_mock\_data*,  
*use\_mock\_data*, *error\_inflation*)

This is a likelihood function for the present-day SN-ratio. Data is approximated from Manucci+2005.

INPUT:

*summary\_pdf* = boolean, should a pdf be created?  
*name\_string* = string to be added in the saved file name  
*abundances* = abundances of the IMS (can be calculated from cube)  
*cube* = the ISM mass fractions per element (A Chempy class containing the model evolution)  
*elements\_to\_trace* = Which elements should be used for the analysis  
*gas\_reservoir* = the *gas\_reservoir* class containing the *gas\_reservoir* evolution  
*number\_of\_models\_overplotted* = default is 1, if more the results will be saved and in the last iteration all former models will be plotted at once  
*produce\_mock\_data* = should the predictions be saved with an error added to them (default=False)  
*use\_mock\_data* = instead of the real data use the formerly produced mock data (default = False)  
*error\_inflation* = a factor with which the mock data should be perturbed with the observational data (default = 1.0)

OUTPUT:

*probabilities* = each elements likelihood in a list  
*model\_abundances* = each elements model prediction as a list  
*elements\_list* = the names of the elements in a list

`Chempy.data_to_test.read_out_wildcard` (*stellar\_identifier*, *list\_of\_abundances*, *ele-*  
*ments\_to\_trace*)

This function calculates analytically an optimal model error and the resulting likelihood from the comparison of predictions and observations



## INPUT:

`list_of_abundances` = a list of the abundances coming from Chempy

`elements_to_trace` = a list of the elemental symbols

## OUTPUT:

`likelihood` = the added log likelihood

`element_list` = the elements that were in common between the predictions and the observations, has the same sequence as the following arrays

`model_error` = the analytic optimal model error

`star_error_list` = the observed error

`abundance_list` = the predictions

`star_abundance_list` = the observations

`Chempy.data_to_test.sample_stars` (*weight, selection, element1, element2, error1, error2, nsample*)

This function samples stars along a chemical evolution track properly taking into account the SFR and the selection function of the stellar population (e.g. red-clump stars). It can be used to produce mock observations which can be compared to survey data.

## INPUT

`weight`: The SFR of the model

`selection`: The age-distribution of a stellar population (e.g. red-clump stars). The time intervals need to be the same as for 'weight'.

`element1` = the values of one element for the ISM of the model (same time-intervals as SFR)

`element2` = the values of the other element for the ISM

`error1` = the measurement error of that element

`error2` = measurement error

`nsample` = number of stars that should be realized

`Chempy.data_to_test.save_abundances` (*summary\_pdf, name\_string, abundances*)

a function that saves the abundances in the current directory

## INPUT:

`summary_pdf` = boolean should the abundances be saved?

`name_string` = name for the saved file

`abundances` = the abundance instance derived from the `cube_class`

## OUTPUT:

Saves the abundances as a npy file

`Chempy.data_to_test.sol_norm` (*summary\_pdf, name\_string, abundances, cube, elements\_to\_trace, element\_names, sol\_table, number\_of\_models\_overplotted, produce\_mock\_data, use\_mock\_data, error\_inflation*)

This is a likelihood function for solar abundances compared to the model ISM abundances from 4.5Gyr ago.

## INPUT:

summary\_pdf = boolean, should a pdf be created?  
 name\_string = string to be added in the saved file name  
 abundances = abundances of the IMS (can be calculated from cube)  
 cube = the ISM mass fractions per element (A Chempy class containing the model evolution)  
 elements\_to\_trace = which elements are tracked by Chempy  
 element\_names = which elements should be used for the likelihood  
 sol\_table = solar abundance class  
 number\_of\_models\_overplotted = default is 1, if more the results will be saved and in the last iteration all former models will be plotted at once  
 produce\_mock\_data = should the predictions be saved with an error added to them (default=False)  
 use\_mock\_data = instead of the real data use the formerly produced mock data (default = False)  
 error\_inflation = a factor with which the mock data should be perturbed with the observational data (default = 1.0)

#### OUTPUT:

probabilities = each elements likelihood in a list  
 model\_abundances = each elements model prediction as a list  
 elements\_list = the names of the elements in a list

`Chempy.data_to_test.star_function` (*summary\_pdf*, *name\_string*, *abundances*,  
*cube*, *elements\_to\_trace*, *gas\_reservoir*, *number\_of\_models\_overplotted*)

!!! Should only be used for plotting purposes

A likelihood function for the stellar surface mass density. It is not updated. Should only be used for plotting purposes as it shows the infall and SFR of a Chempy model

`Chempy.data_to_test.wildcard_likelihood_function` (*summary\_pdf*, *stellar\_identifier*,  
*abundances*)

This function produces Chempy conform likelihood output for a abundance wildcard that was produced before with 'produce\_wildcard\_stellar\_abundances'.

#### INPUT:

summary\_pdf = bool, should there be an output  
 stellar\_identifier = str, name of the star  
 abundances = the abundances instance from a chempy chemical evolution

#### OUTPUT:

probabilities = a list of the likelihoods for each element  
 abundance\_list = the abundances of the model for each element  
 element\_list = the symbols of the corresponding elements

These list will be used to produce the likelihood and the blobs. See cem\_function.py

`Chempy.data_to_test.yield_comparison_plot` (*yield\_name1*, *yield\_name2*, *yield\_class*,  
*yield\_class2*, *solar\_class*, *element*)

a function to plot a comparison between two yield sets. It is similar to 'yield\_plot' only that a second yield set can be plotted.

## INPUT

`yield_name1` = Name of the first yield class  
`yield_name2` = Name of the second yield class  
`yield_class` = a Chempy yield class (e.g 'Nomoto2013' from SN2\_Feedback, see tutorial)  
`yield_class2` = a Chempy yield class (e.g 'Nomoto2013' from SN2\_Feedback, see tutorial)  
`solar_class` = a Chempy solar class (needed for normalisation)  
`element` = the element for which to plot the yield table

## OUTPUT

the figure will be saved into the current directory

`Chempy.data_to_test.yield_plot` (*name\_string*, *yield\_class*, *solar\_class*, *element*)  
 This function plots [X/Fe] for the complete mass and metallicity range of a yield class.

## INPUT:

`name_string` = a string which is included in the saved file name  
`yield_class` = a Chempy yield class (e.g 'Nomoto2013' from SN2\_Feedback, see tutorial)  
`solar_class` = a Chempy solar class (needed for normalisation)  
`element` = the element for which to plot the yield table

## OUTPUT

the figure will be saved into the current directory

## Chempy.imf module

**class** `Chempy.imf.IMF` (*mmin=0.08*, *mmax=100.0*, *intervals=5000*)  
 Bases: `object`

This class represents the IMF normed to 1 in units of  $M_{\text{sun}}$ .

Input for initialisation:

`mmin` = minimal mass of the IMF  
`mmax` = maximal mass of the IMF  
`intervals` = how many steps inbetween `mmin` and `mmax` should be given

Then one of the IMF functions can be used

`self.x` = mass base  
`self.dn` = the number of stars at `x`  
`self.dm` = the masses for each mass interval `x`

**BrokenPowerLaw** (*paramet*)

**Chabrier\_1** (*paramet=(0.69, 0.079, -2.3)*)

Chabrier IMF from Chabrier 2003 equation 17 field IMF with variable high mass slope and automatic normalisation

**Chabrier\_2** (*paramet=(22.8978, 716.4, 0.25, -2.3)*)

Chabrier IMF from Chabrier 2001, IMF 3 = equation 8 parameters from table 1

**imf\_mass\_fraction** (*m<sub>low</sub>*, *m<sub>up</sub>*)

Calculates the mass fraction of the IMF sitting between *m<sub>low</sub>* and *m<sub>up</sub>*

**imf\_number\_fraction** (*m<sub>low</sub>*, *m<sub>up</sub>*)

Calculating the number fraction of stars of the IMF sitting between *m<sub>low</sub>* and *m<sub>up</sub>*

**imf\_number\_stars** (*m<sub>low</sub>*, *m<sub>up</sub>*)

**normed\_3slope** (*paramet*=(-1.3, -2.2, -2.7, 0.5, 1.0))

Three slope IMF, Kroupa 1993 as a default

**salpeter** (*alpha*=2.35)

Salpeter IMF

Input the slope of the IMF

**stochastic\_sampling** (*mass*)

The analytic IMF will be resampled according to the mass of the SSP. The IMF will still be normalised to 1

Stochastic sampling is realised by fixing the number of expected stars and then drawing from the probability distribution of the number density. Statistical properties are tested for this sampling and are safe: number of stars and masses converge.

`Chempy.imf.lifetime` (*m*, *Z*)

here we will calculate the MS lifetime of the star after Argast et al., 2000, A&A, 356, 873 INPUT:

*m* = mass in *M<sub>sun</sub>*

*Z* = metallicity in *Z<sub>sun</sub>*

OUTPUT:

returns the lifetime of the star in Gyrs

`Chempy.imf.slope_imf` (*x*, *p1*, *p2*, *p3*, *kn1*, *kn2*)

Is calculating a three slope IMF

INPUT:

*x* = An array of masses for which the IMF should be calculated

*p1*..*p3* = the slopes of the power law

*kn1*, *kn2* = Where the breaks of the power law are

OUTPUT:

An array of frequencies matching the mass base array *x*

## Chempy.infall module

**class** `Chempy.infall.INFALL` (*t*, *sfr*)

Bases: `object`

This class provides the infall mass over time and is matched to the SFR class

**constant** (*paramet*=1)

Constant gas infall of amount in *M<sub>sun</sub>/pc<sup>2</sup>/Gyr* (default is 1) For test purposes only.

**exponential** (*paramet*=(-0.24, 0.0, 1.0))

Exponential gas infall rate in *M<sub>sun</sub>/pc<sup>2</sup>/Gyr*. The exponent is *b \* t + c*, whole thing shifted up by *d* and normalised by *e* to the SFR. Default is *b* = -0.15 and *e* = 1, rest 0

**gamma\_function** (*mass\_factor=1, a\_parameter=2, loc=0, scale=3*)

the gamma function for *a\_parameter* = 2 and *loc* = 0 produces a peak at *scale* so we have a two parameter sfr. Later we can also release *a* to have a larger diversity in functional form.

**linear** (*paramet=(6.3, -0.5)*)

Linear gas infall rate (usually decreasing) in  $\text{Msun/pc}^2/\text{Gyr}$  with an initial infall rate of *start* (default 6.5) and a decrease/increase of *slope* \* *t* from above (default -0.5)

**polynomial** (*paramet=[-0.003, 0.03, -0.3, 5.0]*)

Polynomial gas infall rate in  $\text{Msun/pc}^2/\text{Gyr}$ . *coeff*: 1D array of coefficients in decreasing powers. The number of *coeff* given determines the order of the polynomial. Default is  $-0.004t^3 + 0.04t^2 - 0.4t + 6$  for okay-ish results

**sfr\_related** ()

the infall will be calculated during the Chempy run according to the star formation efficiency usually following a Kennicutt-Schmidt law

**class** Chempy.infall.PRIMORDIAL\_INFALL (*elements, solar\_table*)

Bases: object

**primordial** ()

This returns primordial abundance fractions.

**sn2** (*paramet*)

This can be used to produce alpha enhanced initial abundances

the fractions of the CC SN feedback and the iron abundance in dex needs to be specified

**solar** (*metallicity\_in\_dex, helium\_fraction=0.285*)

solar values scaled to a specific metallicity

INPUT

metallicity\_in\_dex = helium\_fraction =

## Chempy.making\_abundances module

Chempy.making\_abundances.**abundance\_to\_mass\_fraction** (*all\_elements, all\_masses, all\_abundances, abundances, symbols*)

Calculating mass fractions from abundances.

INPUT:

*all\_elements* = list of all elements from solar abundance instance

*all\_masses* = list of corresponding masses from solar abundances

*all\_abundances* = solar abundances (not needed)

*abundances* = the abundances

*symbols* = a list of the elemental symbols corresponding to the abundances

OUTPUT:

the fractions as an array

`Chempy.making_abundances.abundance_to_mass_fraction_normed_to_solar` (*all\_elements, all\_masses, all\_abundances, abundances, symbols*)

Calculating mass fractions normed to solar from abundances.

INPUT:

`all_elements` = list of all elements from solar abundance instance  
`all_masses` = list of corresponding masses from solar abundances  
`all_abundances` = solar abundances (not needed)  
`abundances` = the abundances  
`symbols` = a list of the elemental symbols corresponding to the abundances

OUTPUT:

the fractions as an array

`Chempy.making_abundances.mass_fraction_to_abundances` (*cube, solar\_abundances*)  
calculating the abundances in dex from mass fractions

INPUT:

`cube` = cube table instance  
`solar_abundances` = solar abundance table instance

OUTPUT:

`abundances`  
`element_names`  
`element_numbers`

## **Chempy.optimization module**

`Chempy.optimization.creating_chain` (*a, startpoint*)

This function creates the initial parameter values for an MCMC chain.

INPUT:

`a` = default parameter values from `parameter.py`  
`startpoint` = from where the pointcloud of walkers start in a small sphere

OUTPUT:

returns the array of the initial startpoints

`Chempy.optimization.gaussian_log` (*x, x0, xsig*)

`Chempy.optimization.minimizer_global` (*changing\_parameter, tol, maxiter, verbose, result*)

This is a function that minimizes the posterior coming from global optimization

INPUT:

changing\_parameter = the global SSP parameters (parameters that all stars share)  
tol = at which change in posterior the minimization should stop  
maxiter = maximum number of iteration  
verbose = print or print not result (bool)  
result = the complete parameter set is handed over as an array of shape(len(stars),len(all parameters)).  
From those the local ISM parameters are taken

OUTPUT:

rex.x = for which global parameters the minimization returned the best posterior

`Chempy.optimization.minimizer_initial` (*identifier*)

This is a function that is minimizing the posterior of Chempy from initial conditions (and can be called in multiprocesses)

INPUT:

a = model parameters

OUTPUT:

res.x = the free Chempy parameter for which the posterior was minimal (log posterior is maximized)

`Chempy.optimization.minimizer_local` (*args*)

`Chempy.optimization.one_chain` (*args*)

This function is testing the startpoint of an MCMC chain and tries to find parameter values for which Chempy does not return -inf

`Chempy.optimization.posterior_probability` (*x, a*)

Just returning the posterior probability of Chempy and the list of blobs

## Chempy.parameter module

**class** `Chempy.parameter.ModelParameters`

Bases: `object`

In this class the model parameters are specified. It contains a lot of information which is (not always) necessary to run Chempy. The individual definitions are given as comments.

`ISM_parameters = [-0.3, 0.55, 0.5]`

`ISM_parameters_to_optimize = ['log10_starformation_efficiency', 'log10_sfr_scale', 'ou`

`N_0 = 0.0017782794100389228`

`SSP_parameters = [-2.29, -2.75, -0.8]`

`SSP_parameters_to_optimize = ['high_mass_slope', 'log10_N_0', 'log10_sn1a_time_delay']`

`S_0 = 1`

`a_parameter = 2`

`agbmmax = 8`

`agbmmin = 0.5`

`arcturus_age = 7.1`

`basic_infall_index = 2`

```

basic_infall_name = 'sfr_related'
basic_infall_name_list = ['exponential', 'constant', 'sfr_related', 'peaked_sfr', 'gam
basic_sfr_index = 1
basic_sfr_name = 'gamma_function'
basic_sfr_name_list = ['model_A', 'gamma_function', 'prescribed', 'doubly_peaked']
beta_error_distribution = [True, 1, 10]
bhmmmax = 100.0
bhmmmin = 100.0
calculate_model = True
chabrier_para1 = 0.69
chabrier_para2 = 0.079
check_processes = False
constraints = {'N_0': (0.0, 1.0), 'a_parameter': (0.0, None), 'c_infall': (None, No
cosmic_accretion_element_fractions = [0.76, 0.24]
cosmic_accretion_elements = ['H', 'He']
dummy = 0.0
element_names = ['He', 'C', 'N', 'O', 'F', 'Ne', 'Na', 'Mg', 'Al', 'Si', 'P', 'S', 'Ar
elements_to_trace = ['Al', 'Ar', 'B', 'Be', 'C', 'Ca', 'Cl', 'Co', 'Cr', 'Cu', 'F', 'F
end = 13.5
error_inflation = 1.0
error_marginalization = True
flat_model_error_prior = [0.0, 1.0, 51]
gas_at_start = 0.0
gas_power = 1.0
gas_reservoir_mass_factor = 1.0
gibbs_sampler_maxiter = 10
gibbs_sampler_tolerance = 0.1
high_mass_slope = -2.29
imf_parameter = (0.69, 0.079, -2.29)
imf_type_index = 1
imf_type_name = 'Chabrier_1'
imf_type_name_list = ['normed_3slope', 'Chabrier_1', 'Chabrier_2', 'salpeter', 'Broken
interpolation_index = 1
interpolation_list = ['linear', 'logarithmic']
interpolation_scheme = 'logarithmic'
m = 1000

```



```
mass_factor = 1.0
mass_steps = 5000
maxiter_minimization = 500
mburn = 1
mcmc_tolerance = 0.5
min_mcmc_iterations = 300
mmax = 100
mmin = 0.1
name_infall = 'solar'
name_infall_index = 1
name_infall_list = ['primordial', 'solar', 'simple', 'alpha']
name_string = 'Chempy_default'
ndim = 6
number_of_models_overplotted = 1
nwalkers = 64
only_net_yields_in_process_tables = True
outflow_feedback_fraction = 0.5
p0 = array([-2.29, -2.75, -0.8 , -0.3 , 0.55, 0.5 ])
parameter_names = ['$\\alpha\\mathrm{IMF}$', '$\\log_{10}\\left(\\mathrm{N}\\right)\\mathrm{\\left(\\right)}$', 'gas_at_start']
percentage_of_bh_mass = 0.25
percentage_to_remnant = 0.13
priors = {'N_0': (0.001, 3.0, 1), 'a_parameter': (3.0, 3.0, 0), 'gas_at_start': (0.0, 0.0, 0)}
produce_mock_data = False
sagbmmax = 8.0
sagbmmin = 8.0
save_state_every = 1
send_email = False
sfr_beginning = 0
sfr_factor_for_cosmic_accretion = 1.0
sfr_scale = 3.5
sn1a_exponent = 1.12
sn1a_parameter = [0.0017782794100389228, 0.15848931924611134, 1.12, 0.0]
sn1a_time_delay = 0.15848931924611134
sn1amax = 8
sn1amin = 1
sn2_to_hn = 1.0
```

```

sn2mmax = 100.0
sn2mmin = 8.0
solar_abundance_name = 'Asplund09'
solar_abundance_name_index = 1
solar_abundance_name_list = ['Lodders09', 'Asplund09', 'Asplund05_pure_solar', 'Asplund09']
starformation_efficiency = 0.5011872336272722
start = 0
stellar_identifier = 'Proto-sun'
stellar_identifier_list = ['Proto-sun']
stellar_lifetimes = 'Argast_2000'
stellar_lifetimes_index = 0
stellar_lifetimes_list = ['Argast_2000', 'Raiteri_1996']
stochastic_IMF = False
summary_pdf = False
testing_output = False
time_delay_functional_form = 'maoz'
time_delay_functional_form_list = ['normal', 'maoz', 'gamma_function']
time_delay_index = 1
time_steps = 28
to_optimize = array(['high_mass_slope', 'log10_N_0', 'log10_sn1a_time_delay', 'log10_s'])
tol_minimization = 0.1
total_mass = 1
use_mock_data = False
verbose = 0
yield_table_name_1a = 'Seitenzahl'
yield_table_name_1a_index = 2
yield_table_name_1a_list = ['Iwamoto', 'Thielemann', 'Seitenzahl']
yield_table_name_agb = 'Karakas_net_yield'
yield_table_name_agb_index = 2
yield_table_name_agb_list = ['Karakas', 'Nugrid', 'Karakas_net_yield', 'Ventura', 'Karakas']
yield_table_name_hn = 'Nomoto2013'
yield_table_name_hn_index = 0
yield_table_name_hn_list = ['Nomoto2013']
yield_table_name_sn2 = 'Nomoto2013'
yield_table_name_sn2_index = 2
yield_table_name_sn2_list = ['chieffi04', 'Nugrid', 'Nomoto2013', 'Portinari', 'chieffi04']

```

```
zero_model_error = False
```

## Chempy.plot\_mcmc module

`Chempy.plot_mcmc.plot_element_correlation(directory)`

This is an experimental plotting routine. It can read the mcmc folder content and plot element / parameter / posterior correlations.

For that the name-list of the blobs needs to be provided which can be generated running Chempy in the ‘testing\_output’ mode.

`Chempy.plot_mcmc.plot_mcmc_chain(directory, set_scale=False, use_scale=False, only_first_star=True)`

This routine takes the output from ‘restructure\_chain’ function and plots the result in a corner plot `set_scale` and `use_scale` can be used to put different PDFs on the same scale, in the sense that the plot is shown with the same axis range.

In the paper this is used to plot the Posterior in comparison to the prior distribution.

`Chempy.plot_mcmc.plot_mcmc_chain_with_prior(directory, use_prior=False, only_first_star=True, plot_true_parameters=True, plot_only_SSP_parameter=True)`

This routine takes the output from ‘restructure\_chain’ function and plots the result in a corner plot `set_scale` and `use_scale` can be used to put different PDFs on the same scale, in the sense that the plot is shown with the same axis range.

In the paper this is used to plot the Posterior in comparison to the prior distribution.

`Chempy.plot_mcmc.restructure_chain(directory, parameter_names=[' $\alpha_{\mathrm{IMF}}$ ', ' $\log_{10}(\mathrm{N}_{\mathrm{Ia}})$ ', ' $\log_{10}(\tau_{\mathrm{Ia}})$ ', ' $\log_{10}(\mathrm{SFE})$ ', ' $\log_{10}(\mathrm{SFR}_{\mathrm{peak}})$ ', ' $x_{\mathrm{out}}$ '])`

This function restructures the chains and blobs coming from the emcee routine so that we have a flattened posterior PDF in the end.

The following files need to be there: `flatchain`, `flatlnprobability`, `flatmeanposterior` and `flatstdposterior`. `flatblobs` is optional

INPUT:

`directory` = name of the folder where the files are located

`parameter_names` = the names of the parameters which where explored in the MCMC

OUTPUT:

a few convergence figures and arrays will be saved into `directory`

## Chempy.sfr module

`class Chempy.sfr.SFR(start, end, time_steps)`

Bases: `object`

The SFR class holds the star formation history and the time-steps of Chempy

`doubly_peaked(S0=45.07488, peak_ratio=1.0, decay=2.0, t0=2.0, peak1t0=0.5, peak1sigma=0.5)`

a doubly peaked SFR with quite a few parameters

**gamma\_function** (*S0=45.07488, a\_parameter=2, loc=0, scale=3*)

the gamma function for *a\_parameter* = 2 and *loc* = 0 produces a peak at scale so we have a two parameter sfr. Later we can also release *a* to have a larger diversity in functional form.

**model\_A** (*S0=45.07488, t0=5.6, t1=8.2*)

This was the method to load the Just Jahreiss 2010 Model A from txt

**prescribed** (*mass\_factor, name\_of\_file*)

a method to read in prescribed SFR from textfile x time is given in log years. our time is in linear Gyrs

## **Chempy.solar\_abundance module**

**class** Chempy.solar\_abundance.solar\_abundances

Bases: object

The solar abundances class. Holds information on the element names, symbols, mass numbers and photospheric abundances.

**AG89** ()

Photospheric abundances and errors are loaded from Anders & Grevesse 1989. Also the element fractions are calculated together with X, Y and Z the Hydrogen, Helium and metallicity fraction.

**Asplund05\_apogee\_correction** ()

Photospheric abundances and errors are loaded from Asplund+ 2005 but corrected for the APOGEE scale. Also the element fractions are calculated together with X, Y and Z the Hydrogen, Helium and metallicity fraction. It is not sure for which elements from Asplund+ 2005 the apogee consortium has used the photospheric or the meteoritic abundances. Therefore I try here to use only the photospheric except for elements without photospheric values.

**Asplund05\_pure\_solar** ()

Photospheric abundances and errors are loaded from Asplund+ 2005. Also the element fractions are calculated together with X, Y and Z the Hydrogen, Helium and metallicity fraction. It is not sure for which elements from Asplund+ 2005 the apogee consortium has used the photospheric or the meteoritic abundances. Therefore I try here to use only the photospheric except for elements without photospheric values.

**Asplund09** ()

Photospheric abundances and errors are loaded from Asplund+ 2009. Also the element fractions are calculated together with X, Y and Z the Hydrogen, Helium and metallicity fraction.

**Lodders09** ()

Photospheric abundances and errors are loaded from Lodders+ 2009. Also the element fractions are calculated together with X, Y and Z the Hydrogen, Helium and metallicity fraction.

## Chempy.time\_integration module

```
class Chempy.time_integration.ABUNDANCE_MATRIX(time, sfr, infall, list_of_elements,
                                                infall_symbols, infall_fractions,
                                                gas_at_start, gas_at_start_symbols,
                                                gas_at_start_fractions,
                                                gas_reservoir_mass_factor,
                                                outflow_feedback_fraction,
                                                check_processes, starforma-
                                                tion_efficiency, gas_power,
                                                sfr_factor_for_cosmic_accretion,
                                                cosmic_accretion_elements, cos-
                                                mic_accretion_element_fractions)
```

Bases: object

This class contains all information necessary to characterize the chemical evolution of the open-box one-zone Chempy model.

It calculates the mass flow between the different components. And can advance the chemical evolution when the enrichment from the SSP is provided.

**advance\_one\_step** (*index, ssp\_yield, sn2\_yield, agb\_yield, sn1a\_yield, bh\_yield*)

This method advances the chemical evolution one time-step.

INPUT:

*index* = which time step should be filled up

*ssp\_yield* = yield of the ssp

*sn2\_yield* = yield of sn2 only

*agb\_yield* = yield of agb only

*sn1a\_yield* = yield of sn1a only

*bh\_yield* = yield of bh only

## Chempy.weighted\_yield module

```
class Chempy.weighted_yield.SSP(output, z, imf_x, imf_dm, imf_dn, time_steps, ele-
                                ments_to_trace, stellar_lifetimes, interpolation_scheme,
                                only_net_yields_in_process_tables)
```

Bases: object

The simple stellar population class can calculate the enrichment over time for an SSP from a few assumptions and input yield tables.

**agb\_feedback** (*agb\_elements, agb\_yields, agb\_metallicities, agb\_mmin, agb\_mmax, frac-*  
*tions\_in\_gas*)

AGB enrichment calculation adds the feedback to the total SSP table and also to the self.agb\_yield table.

INPUT:

*agb\_elements* = which elements are provided by the yield table, list containing the symbols

*agb\_yields* = the yield table provided by Chempy's AGB yield class

*agb\_metallicities* = the metallicities of that table

*agb\_mmin* = the minimal mass of the AGB stars (default 0.5) in Msun

`agb_mmax` = the maximum mass of the AGB stars (default 8) in Msun

`fractions_in_gas` = the birth material of the SSP (will be mixed into the enrichment as unprocessed material)

OUTPUT:

the classes table will be filled up and a `sn2_table` will be added (so that the individual processes can be tracked)

**bh\_feedback** (*bhmin, bhmax, element\_list, fractions\_in\_gas, percentage\_of\_bh\_mass*)

BH enrichment routine, just no enrichment for a specific mass range. A set percentage is fed back into the ISM

**Inputs:** Min/Max black hole mass (40-100 is default - see parameter file).

Element list to be calculated

Fractions of each element in the ISM gas

Percentage of BH progenitor fed back into the ISM (75% default)

**post\_agb\_feedback** (*mmin, mmax, element\_list, fractions\_in\_gas, percentage\_to\_remnant*)

just to produce no new elements for stars between agb and sn2, like in kobayashi 2011

**sn1a\_feedback** (*sn1a\_elements, sn1a\_metallicities, sn1a\_yields, time\_delay\_functional\_form, sn1a\_min, sn1a\_max, time\_delay\_parameter, ssp\_mass, stochastic\_IMF*)

Calculating the SN1a feedback over time

INPUT:

`sn1a_elements` = Which elements are provided by the yield table

`sn1a_metallicities` = metallicities in the yield table

`sn1a_yields` = yield table

`time_delay_functional_form` = which functional form of the delay time should be used ('normal', 'maoz', 'gamma\_function'). Maoz is the default and the others are not tested. Check for functionality

`sn1a_min` = the minimum mass from which sn1a can occur (does not matter for maoz)

`sn1a_max` = the maximum mass from which SN Ia can occur (does not matter for maoz)

`time_delay_parameter` = a tuple containing the parameters for the specific functional form

`ssp_mass` = the mass of the SSP

`stochastic_IMF` = bool, do we want to use stochastic explosions

OUTPUT:

the classes table will be filled up and a `sn2_table` will be added (so that the individual processes can be tracked)

for MAOZ functional form the following parameters are in `time_delay_parameter`:

`N_0` = Number of SNIa exploding per Msun over the course of 15Gyr

`tau_8` = The delay time when the first SN Ia explode (usually 40Myr are anticipated because then 8Msun stars start to die but our Prior is more at 160Myr)

`s_eponent` = the time decay exponent

`dummy` = not in use anymore

**sn2\_feedback** (*sn2\_elements, sn2\_yields, sn2\_metallicities, sn2\_mmin, sn2\_mmax, fractions\_in\_gas*)

Calculating the CC-SN feedback over time. The routine is sensitive to the ordering of the masses in the yield table, it must begin with the smallest increase to the biggest value.

INPUT:

sn2\_elements = which elements are provided by the yield table, list containing the symbols

sn2\_yields = the yield table provided by Chempys SN2 yield class

sn2\_metallicities = the metallicities of that table

sn2\_mmin = the minimal mass of the CC-SN (default 8) in Msun

sn2\_mmax = the maximum mass of the CC-SN (default 100) in Msun

fractions\_in\_gas = the birth material of the SSP (will be mixed into the enrichment as unprocessed material)

OUTPUT:

the classes table will be filled up and a sn2\_table will be added (so that the individual processes can be tracked)

**sn2\_feedback\_IRA** (*sn2\_elements, sn2\_yields, sn2\_metallicities, sn2\_mmin, sn2\_mmax, fractions\_in\_gas*)

The mass fraction of the IMF between sn2\_mmin and sn2\_mmax is fed back instantaneously to the ISM according to the relative yields of sn2. The interpolation is linear in mass and metallicity. Also the mass transformed into remnants is calculated. The routine is sensitive to the ordering of the masses in the yield table, it must begin with the smallest increase to the biggest value.

`Chempy.weighted_yield.imf_mass_fraction_non_nativ` (*imf\_dm, imf\_x, mlow, mup*)

Function to determine the mass fraction of the IMF between two masses

INPUT:

imf\_dm = imf\_class.dm

imf\_x = imf\_class.x

mlow = lower mass

mup = upper mass

OUTPUT:

the mass fraction in this mass range

`Chempy.weighted_yield.lifetime_Argast` (*m, Z\_frac*)

here we will calculate the MS lifetime of the star after Argast et al., 2000, A&A, 356, 873

INPUT:

m = mass in Msun

Z\_frac = fractions of metals of the stellar composition

Z = metallicity in Zsun

OUTPUT:

returns the lifetime of the star in Gyrs

`Chempy.weighted_yield.lifetime_Raiteri` (*m, Z*)

INPUT:

m = mass in Msun  
Z = metallicity in Zsun  
returns the lifetime of the star in Gyrs

## Chempy.wrapper module

`Chempy.wrapper.Chempy(a)`

Chemical evolution run with the default parameters using the net yields.

INPUT:

a = ModelParameters() from parameter.py

OUTPUT:

cube = The ISM evolution class

abundances = The abundances of the ISM

`Chempy.wrapper.Chempy_gross(a)`

Chemical evolution run with the default parameters but now using solar scaled material (testing the worse case when total yields provided).

INPUT:

a = ModelParameters() from parameter.py

OUTPUT:

cube = The ISM evolution class

abundances = The abundances of the ISM

**class** `Chempy.wrapper.SSP_wrap(a)`

This is the wrapper around the SSP function. It preloads the needed classes and calls all nucleosynthetic enrichment processes when the enrichment is calculated.

**calculate\_feedback** (*z, elements, element\_fractions, time\_steps*)

The feedback is calculated for the initializes SSP.

INPUT:

z = metallicity of the SSP in mass fraction (not normed to solar!)

elements = which elements to follow

element\_fractions = the birth material of the SSP in the same order as 'elements'

time\_steps = the time-steps for which the enrichment of the SSP should be calculated (usually the time-steps until the end of the chempy simulation)

`Chempy.wrapper.initialise_stuff(a)`

Convenience function initialising the solar abundance, SFR and infall with the default values provided in parameter.py as a

`Chempy.wrapper.mcmc(a)`

Convenience function to use the MCMC. A subdirectory mcmc/ will be created in the current directory and intermediate chains will be stored there. The chains are not actually flattened as the file name suggests.

The MCMC will sample the volume of best posterior for the likelihood functions that are declared in parameter.py. Default is ['sol\_norm', 'gas\_reservoir', 'sn\_ratio'] which corresponds to 'Sun+' from the paper.



`Chempy.wrapper.mcmc_multi` (*changing\_parameter, error\_list, elements*)

Convenience function to use the MCMC for multiple zones (and therefore multiple observations). A subdirectory `mcmc/` will be created in the current directory and intermediate chains will be stored there. The MCMC will sample the volume of best posterior for the likelihood functions that are declared in `parameter.py`. Default is a list of Proto-sun, Arcturus and B-stars. The MCMC uses many walkers and can use multiple threads. Each walker will evaluate a series of Chempy zones and add their posterior together which then will be returned.

INPUT:

`changing_parameter` = the parameter vector for initialization (will usually be found from minimization before). The initial chain will be created by jittering slightly the initial parameter guess

`error_list` = the vector of element errors

`elements` = the corresponding element symbols

OUTPUT:

The function will create a folder and store the chain as well as the predicted element values

The MCMC stops when the convergence criteria is met, which is when the median posterior of all walkers does not change much inbetween 200 steps anymore.

`Chempy.wrapper.multi_star_optimization` ()

This function will optimize the parameters of all stars in a hierachical manner (similar to gibbs sampling)

INPUT:

`a` = will be loaded from `parameter.py` (prepare all variables there)

OUTPUT:

`log_list` = a list of intermediate results (so far only for debugging)

`Chempy.wrapper.send_email` (*thread\_count, iteration\_count, posterior\_beginning, posterior\_end, parameters, time*)

## Chempy.yields module

**class** `Chempy.yields.AGB_feedback`

Bases: `object`

**Karakas** ()

loading the yield table of Karakas 2010.

**Karakas16\_net** ()

load the Karakas 2016 yields send by Amanda and Fishlock 2014 for  $Z = 0.001$ . With slight inconsistencies in the mass normalisation and not sure which Asplund2009 solar abundances she uses

**Karakas\_net\_yield** ()

loading the yield table of Karakas 2010.

**Nomoto2013** ()

Nomoto2013 agb yields up to 6.5Msun and are a copy of Karakas2010. Only that the yields here are given as net yields which does not help so much

**Nugrid** ()

loading the Nugrid intermediate mass stellar yields NuGrid stellar data set. I. Stellar yields from H to Bi for stars with metallicities  $Z = 0.02$  and  $Z = 0.01$

**Ventura** ()

Ventura 2013 net yields from Paolo himself

**one\_parameter** (*elements, element\_fractions*)

Another problem: He and the remnant mass fraction is not constrained in the APOGEE data. Maybe these can be constrained externally by yield sets or cosmic abundance standard or solar abundances.

**class** Chempy.yields.Hypernova\_feedback

Bases: object

**Nomoto2013** ()

Nomoto2013 sn2 yields from 13Msun onwards

**class** Chempy.yields.SN1a\_feedback

Bases: object

**Iwamoto** ()

Iwamoto99 yields building up on Nomoto84

**Seitenzahl** ()

Seitenzahl 2013 from Ivo txt

**Thielemann** ()

Thielemann 2003 yields as compiled in Travaglio 2004

**class** Chempy.yields.SN2\_feedback

Bases: object

**Nomoto2013** ()

Nomoto2013 sn2 yields from 13Msun onwards

**Nomoto2013\_net** ()

Nomoto2013 sn2 yields from 13Msun onwards

**Nugrid** ()

loading the Nugrid sn2 stellar yields NuGrid stellar data set. I. Stellar yields from H to Bi for stars with metallicities  $Z = 0.02$  and  $Z = 0.01$  The wind yields need to be added to the *exp* explosion yields. No r-process contribution but s and p process from AGB and massive stars delayed and rapid SN Explosiom postprocessing is included. Rapid is not consistent with very massive stars so we use the 'delayed' yield set mass in remnants not totally consistent with paper table: [ 6.47634087, 2.67590435, 1.98070676] vs. [6.05,2.73,1.61] see table 4 same with  $z=0.02$  but other elements are implemented in the right way:[ 3.27070753, 8.99349996, 6.12286813, 3.1179861 , 1.96401573] vs. [3,8.75,5.71,2.7,1.6] we have a switch to change between the two different methods (rapid/delay explosion)

**Portinari** ()

Loading the yield table from Portinari1998.

**chieffi04** ()

Loading the yield table of chieffi04.

**chieffi04\_net** ()

Loading the yield table of chieffi04 corrected for Anders & Grevesse 1989 solar scaled initial yields

**francois** ()

Loading the yield table of Francois et. al. 2004. Taken from the paper table 1 and 2 and added O H He from WW95 table 5A and 5B where all elements are for  $Z=Z_{\text{sun}}$  and values for  $M_{\text{sun}} > 40$  have been stayed the same as for  $M_{\text{sun}}=40$ . Values from 11-25  $M_{\text{sun}}$  used case A from WW95 and 30-40  $M_{\text{sun}}$  used case B.

**one\_parameter** (*elements, element\_fractions*)

This function was introduced in order to find best-fit yield sets where each element has just a single yield (no metallicity or mass dependence). One potential problem is that sn2 feedback has a large fraction of Neon  $\sim 0.01$ , the next one missing is Argon but that only has 0.05%. This might spoil the metallicity derivation a bit. Another problem: He and the remnant mass fraction is not constrained in the APOGEE

data. Maybe these can be constrained externally by yield sets or cosmic abundance standard or solar abundances.

## Module contents

## 1.2 Installation

- Using pip:

```
pip install git+https://github.com/jan-rybizki/Chempy.git
```

- Manually:

```
git clone https://github.com/jan-rybizki/Chempy
cd Chempy
python setup.py intall
```



Look into the very extensive [jupyter tutorial](#). Here is a short preview:

```
# Loading the default parameters
from Chempy.parameter import ModelParameters
a = ModelParameters()

# Evaluating the Chempy posterior at the maximum prior parameters
from Chempy.cem_function import cem
a.testing_output = True
a.summary_pdf = True
a.observational_constraints_index = ['gas_reservoir', 'sn_ratio', 'sol_norm']
posterior, blobs = cem(a.p0, a)
```

---

## 2.1 References

- Chempy paper



## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### C

- Chempy, [31](#)
- Chempy.cem\_function, [3](#)
- Chempy.data\_to\_test, [7](#)
- Chempy.imf, [15](#)
- Chempy.infall, [16](#)
- Chempy.making\_abundances, [17](#)
- Chempy.optimization, [18](#)
- Chempy.parameter, [19](#)
- Chempy.plot\_mcmc, [23](#)
- Chempy.sfr, [23](#)
- Chempy.solar\_abundance, [24](#)
- Chempy.time\_integration, [25](#)
- Chempy.weighted\_yield, [25](#)
- Chempy.wrapper, [28](#)
- Chempy.yields, [29](#)



## A

ABUNDANCE\_MATRIX (class in Chempy.time\_integration), 25  
 abundance\_to\_mass\_fraction() (in module Chempy.making\_abundances), 17  
 abundance\_to\_mass\_fraction\_normed\_to\_solar() (in module Chempy.making\_abundances), 17  
 advance\_one\_step() (Chempy.time\_integration.ABUNDANCE\_MATRIX method), 25  
 AG89() (Chempy.solar\_abundance.solar\_abundances method), 24  
 AGB\_feedback (class in Chempy.yields), 29  
 agb\_feedback() (Chempy.weighted\_yield.SSP method), 25  
 agbmmx (Chempy.parameter.ModelParameters attribute), 19  
 agbmmn (Chempy.parameter.ModelParameters attribute), 19  
 arcturus() (in module Chempy.data\_to\_test), 7  
 arcturus\_age (Chempy.parameter.ModelParameters attribute), 19  
 Asplund05\_apogee\_correction() (Chempy.solar\_abundance.solar\_abundances method), 24  
 Asplund05\_pure\_solar() (Chempy.solar\_abundance.solar\_abundances method), 24  
 Asplund09() (Chempy.solar\_abundance.solar\_abundances method), 24

## B

basic\_infall\_index (Chempy.parameter.ModelParameters attribute), 19  
 basic\_infall\_name (Chempy.parameter.ModelParameters attribute), 19  
 basic\_infall\_name\_list (Chempy.parameter.ModelParameters attribute), 20  
 basic\_sfr\_index (Chempy.parameter.ModelParameters attribute), 20  
 basic\_sfr\_name (Chempy.parameter.ModelParameters attribute), 20  
 basic\_sfr\_name\_list (Chempy.parameter.ModelParameters attribute), 20  
 beta\_error\_distribution (Chempy.parameter.ModelParameters attribute), 20  
 bh\_feedback() (Chempy.weighted\_yield.SSP method), 26  
 bhmmax (Chempy.parameter.ModelParameters attribute), 26  
 bhmmin (Chempy.parameter.ModelParameters attribute), 20  
 BrokenPowerLaw() (Chempy.imf.IMF method), 15

## C

calculate\_feedback() (Chempy.wrapper.SSP\_wrap method), 28  
 calculate\_model (Chempy.parameter.ModelParameters attribute), 20  
 cem() (in module Chempy.cem\_function), 3  
 cem2() (in module Chempy.cem\_function), 4  
 cem\_real() (in module Chempy.cem\_function), 4  
 cem\_real2() (in module Chempy.cem\_function), 4  
 Chabrier\_1() (Chempy.imf.IMF method), 15  
 Chabrier\_2() (Chempy.imf.IMF method), 15  
 chabrier\_para1 (Chempy.parameter.ModelParameters attribute), 20  
 chabrier\_para2 (Chempy.parameter.ModelParameters attribute), 20  
 check\_processes (Chempy.parameter.ModelParameters attribute), 20  
 Chempy (module), 31  
 Chempy() (in module Chempy.wrapper), 28  
 Chempy.cem\_function (module), 3  
 Chempy.data\_to\_test (module), 7  
 Chempy.imf (module), 15  
 Chempy.infall (module), 16  
 Chempy.making\_abundances (module), 17  
 Chempy.optimization (module), 18  
 Chempy.parameter (module), 19

Chempy.plot\_mcmc (module), 23  
 Chempy.sfr (module), 23  
 Chempy.solar\_abundance (module), 24  
 Chempy.time\_integration (module), 25  
 Chempy.weighted\_yield (module), 25  
 Chempy.wrapper (module), 28  
 Chempy.yields (module), 29  
 Chempy.gross() (in module Chempy.wrapper), 28  
 chieffi04() (Chempy.yields.SN2\_feedback method), 30  
 chieffi04\_net() (Chempy.yields.SN2\_feedback method), 30  
 constant() (Chempy.infall.INFALL method), 16  
 constraints (Chempy.parameter.ModelParameters attribute), 20  
 cosmic\_abundance\_standard() (in module Chempy.data\_to\_test), 8  
 cosmic\_accretion\_element\_fractions (Chempy.parameter.ModelParameters attribute), 20  
 cosmic\_accretion\_elements (Chempy.parameter.ModelParameters attribute), 20  
 creating\_chain() (in module Chempy.optimization), 18

## D

doubly\_peaked() (Chempy.sfr.SFR method), 23  
 dummy (Chempy.parameter.ModelParameters attribute), 20

## E

element\_names (Chempy.parameter.ModelParameters attribute), 20  
 elements\_plot() (in module Chempy.data\_to\_test), 8  
 elements\_to\_trace (Chempy.parameter.ModelParameters attribute), 20  
 end (Chempy.parameter.ModelParameters attribute), 20  
 error\_inflation (Chempy.parameter.ModelParameters attribute), 20  
 error\_marginalization (Chempy.parameter.ModelParameters attribute), 20  
 exponential() (Chempy.infall.INFALL method), 16  
 extract\_parameters\_and\_priors() (in module Chempy.cem\_function), 4

## F

flat\_model\_error\_prior (Chempy.parameter.ModelParameters attribute), 20  
 fractional\_yield\_comparison\_plot() (in module Chempy.data\_to\_test), 9  
 francois() (Chempy.yields.SN2\_feedback method), 30

## G

gamma\_function() (Chempy.infall.INFALL method), 16

gamma\_function() (Chempy.sfr.SFR method), 24  
 gas\_at\_start (Chempy.parameter.ModelParameters attribute), 20  
 gas\_power (Chempy.parameter.ModelParameters attribute), 20  
 gas\_reservoir\_mass\_factor (Chempy.parameter.ModelParameters attribute), 20  
 gas\_reservoir\_metallicity() (in module Chempy.data\_to\_test), 9  
 gaussian() (in module Chempy.cem\_function), 4  
 gaussian() (in module Chempy.data\_to\_test), 10  
 gaussian\_1d\_log() (in module Chempy.data\_to\_test), 10  
 gaussian\_log() (in module Chempy.cem\_function), 4  
 gaussian\_log() (in module Chempy.optimization), 18  
 get\_prior() (in module Chempy.cem\_function), 4  
 gibbs\_sampler\_maxiter (Chempy.parameter.ModelParameters attribute), 20  
 gibbs\_sampler\_tolerance (Chempy.parameter.ModelParameters attribute), 20  
 global\_optimization() (in module Chempy.cem\_function), 5  
 global\_optimization\_error\_returned() (in module Chempy.cem\_function), 5  
 global\_optimization\_real() (in module Chempy.cem\_function), 5

## H

high\_mass\_slope (Chempy.parameter.ModelParameters attribute), 20  
 Hypernova\_feedback (class in Chempy.yields), 30

## I

IMF (class in Chempy.imf), 15  
 imf\_mass\_fraction() (Chempy.imf.IMF method), 15  
 imf\_mass\_fraction\_non\_nativ() (in module Chempy.weighted\_yield), 27  
 imf\_number\_fraction() (Chempy.imf.IMF method), 16  
 imf\_number\_stars() (Chempy.imf.IMF method), 16  
 imf\_parameter (Chempy.parameter.ModelParameters attribute), 20  
 imf\_type\_index (Chempy.parameter.ModelParameters attribute), 20  
 imf\_type\_name (Chempy.parameter.ModelParameters attribute), 20  
 imf\_type\_name\_list (Chempy.parameter.ModelParameters attribute), 20  
 INFALL (class in Chempy.infall), 16  
 initialise\_stuff() (in module Chempy.wrapper), 28  
 interpolation\_index (Chempy.parameter.ModelParameters attribute), 20  
 interpolation\_list (Chempy.parameter.ModelParameters attribute), 20

interpolation\_scheme (Chempy.parameter.ModelParameters attribute), 20

ISM\_parameters (Chempy.parameter.ModelParameters attribute), 19

ISM\_parameters\_to\_optimize (Chempy.parameter.ModelParameters attribute), 19

Iwamoto() (Chempy.yields.SN1a\_feedback method), 30

## K

Karakas() (Chempy.yields.AGB\_feedback method), 29

Karakas16\_net() (Chempy.yields.AGB\_feedback method), 29

Karakas\_net\_yield() (Chempy.yields.AGB\_feedback method), 29

## L

lifetime() (in module Chempy.imf), 16

lifetime\_Argast() (in module Chempy.weighted\_yield), 27

lifetime\_Raiteri() (in module Chempy.weighted\_yield), 27

likelihood\_evaluation() (in module Chempy.data\_to\_test), 10

likelihood\_function() (in module Chempy.data\_to\_test), 10

linear() (Chempy.infall.INFALL method), 17

Lodders09() (Chempy.solar\_abundance.solar\_abundances method), 24

lognorm() (in module Chempy.cem\_function), 5

lognorm\_log() (in module Chempy.cem\_function), 5

## M

m (Chempy.parameter.ModelParameters attribute), 20

mass\_factor (Chempy.parameter.ModelParameters attribute), 20

mass\_fraction\_to\_abundances() (in module Chempy.making\_abundances), 18

mass\_steps (Chempy.parameter.ModelParameters attribute), 21

maxiter\_minimization (Chempy.parameter.ModelParameters attribute), 21

mburn (Chempy.parameter.ModelParameters attribute), 21

mcmc() (in module Chempy.wrapper), 28

mcmc\_multi() (in module Chempy.wrapper), 28

mcmc\_tolerance (Chempy.parameter.ModelParameters attribute), 21

min\_mcmc\_iterations (Chempy.parameter.ModelParameters attribute), 21

minimizer\_global() (in module Chempy.optimization), 18

minimizer\_initial() (in module Chempy.optimization), 19

minimizer\_local() (in module Chempy.optimization), 19

21

mmin (Chempy.parameter.ModelParameters attribute), 21

model\_A() (Chempy.sfr.SFR method), 24

ModelParameters (class in Chempy.parameter), 19

multi\_star\_optimization() (in module Chempy.wrapper), 29

## N

N\_0 (Chempy.parameter.ModelParameters attribute), 19

name\_infall (Chempy.parameter.ModelParameters attribute), 21

name\_infall\_index (Chempy.parameter.ModelParameters attribute), 21

name\_infall\_list (Chempy.parameter.ModelParameters attribute), 21

name\_string (Chempy.parameter.ModelParameters attribute), 21

ndim (Chempy.parameter.ModelParameters attribute), 21

Nomoto2013() (Chempy.yields.AGB\_feedback method), 29

Nomoto2013() (Chempy.yields.Hypernova\_feedback method), 30

Nomoto2013() (Chempy.yields.SN2\_feedback method), 30

Nomoto2013\_net() (Chempy.yields.SN2\_feedback method), 30

normed\_3slope() (Chempy.imf.IMF method), 16

Nugrid() (Chempy.yields.AGB\_feedback method), 29

Nugrid() (Chempy.yields.SN2\_feedback method), 30

number\_of\_models\_overplotted (Chempy.parameter.ModelParameters attribute), 21

nwalkers (Chempy.parameter.ModelParameters attribute), 21

## O

one\_chain() (in module Chempy.optimization), 19

one\_parameter() (Chempy.yields.AGB\_feedback method), 29

one\_parameter() (Chempy.yields.SN2\_feedback method), 30

only\_net\_yields\_in\_process\_tables (Chempy.parameter.ModelParameters attribute), 21

outflow\_feedback\_fraction (Chempy.parameter.ModelParameters attribute), 21

## P

p0 (Chempy.parameter.ModelParameters attribute), 21

parameter\_names (Chempy.parameter.ModelParameters attribute), 21

- percentage\_of\_bh\_mass (Chempy.parameter.ModelParameters attribute), 21
- percentage\_to\_remnant (Chempy.parameter.ModelParameters attribute), 21
- plot\_abundance\_wildcard() (in module Chempy.data\_to\_test), 11
- plot\_element\_correlation() (in module Chempy.plot\_mcmc), 23
- plot\_mcmc\_chain() (in module Chempy.plot\_mcmc), 23
- plot\_mcmc\_chain\_with\_prior() (in module Chempy.plot\_mcmc), 23
- plot\_processes() (in module Chempy.data\_to\_test), 11
- polynomial() (Chempy.infall.INFALL method), 17
- Portinari() (Chempy.yields.SN2\_feedback method), 30
- post\_agb\_feedback() (Chempy.weighted\_yield.SSP method), 26
- posterior\_function() (in module Chempy.cem\_function), 5
- posterior\_function\_for\_minimization() (in module Chempy.cem\_function), 6
- posterior\_function\_local() (in module Chempy.cem\_function), 6
- posterior\_function\_local\_for\_minimization() (in module Chempy.cem\_function), 6
- posterior\_function\_local\_real() (in module Chempy.cem\_function), 6
- posterior\_function\_many\_stars() (in module Chempy.cem\_function), 6
- posterior\_function\_many\_stars\_real() (in module Chempy.cem\_function), 7
- posterior\_function\_predictions() (in module Chempy.cem\_function), 7
- posterior\_function\_real() (in module Chempy.cem\_function), 7
- posterior\_function\_returning\_predictions() (in module Chempy.cem\_function), 7
- posterior\_probability() (in module Chempy.optimization), 19
- prescribed() (Chempy.sfr.SFR method), 24
- primordial() (Chempy.infall.PRIMORDIAL\_INFALL method), 17
- PRIMORDIAL\_INFALL (class in Chempy.infall), 17
- priors (Chempy.parameter.ModelParameters attribute), 21
- produce\_mock\_data (Chempy.parameter.ModelParameters attribute), 21
- produce\_wildcard\_stellar\_abundances() (in module Chempy.data\_to\_test), 11
- R**
- ratio\_function() (in module Chempy.data\_to\_test), 12
- read\_out\_wildcard() (in module Chempy.data\_to\_test), 12
- restructure\_chain() (in module Chempy.plot\_mcmc), 23
- S**
- S\_0 (Chempy.parameter.ModelParameters attribute), 19
- sagbmmmax (Chempy.parameter.ModelParameters attribute), 21
- sagbmmmin (Chempy.parameter.ModelParameters attribute), 21
- salpeter() (Chempy.imf.IMF method), 16
- sample\_stars() (in module Chempy.data\_to\_test), 13
- save\_abundances() (in module Chempy.data\_to\_test), 13
- save\_state\_every (Chempy.parameter.ModelParameters attribute), 21
- Seitenzahl() (Chempy.yields.SN1a\_feedback method), 30
- send\_email (Chempy.parameter.ModelParameters attribute), 21
- send\_email() (in module Chempy.wrapper), 29
- SFR (class in Chempy.sfr), 23
- sfr\_beginning (Chempy.parameter.ModelParameters attribute), 21
- sfr\_factor\_for\_cosmic\_accretion (Chempy.parameter.ModelParameters attribute), 21
- sfr\_related() (Chempy.infall.INFALL method), 17
- sfr\_scale (Chempy.parameter.ModelParameters attribute), 21
- shorten\_sfr() (in module Chempy.cem\_function), 7
- slope\_imf() (in module Chempy.imf), 16
- sn1a\_exponent (Chempy.parameter.ModelParameters attribute), 21
- SN1a\_feedback (class in Chempy.yields), 30
- sn1a\_feedback() (Chempy.weighted\_yield.SSP method), 26
- sn1a\_parameter (Chempy.parameter.ModelParameters attribute), 21
- sn1a\_time\_delay (Chempy.parameter.ModelParameters attribute), 21
- sn1amax (Chempy.parameter.ModelParameters attribute), 21
- sn1ammin (Chempy.parameter.ModelParameters attribute), 21
- sn2() (Chempy.infall.PRIMORDIAL\_INFALL method), 17
- SN2\_feedback (class in Chempy.yields), 30
- sn2\_feedback() (Chempy.weighted\_yield.SSP method), 26
- sn2\_feedback\_IRA() (Chempy.weighted\_yield.SSP method), 27
- sn2\_to\_hn (Chempy.parameter.ModelParameters attribute), 21
- sn2mmax (Chempy.parameter.ModelParameters attribute), 21
- sn2mmin (Chempy.parameter.ModelParameters attribute), 22
- sol\_norm() (in module Chempy.data\_to\_test), 13

[solar\(\)](#) (Chempy.infall.PRIMORDIAL\_INFALL method), 17  
[solar\\_abundance\\_name](#) (Chempy.parameter.ModelParameters attribute), 22  
[solar\\_abundance\\_name\\_index](#) (Chempy.parameter.ModelParameters attribute), 22  
[solar\\_abundance\\_name\\_list](#) (Chempy.parameter.ModelParameters attribute), 22  
[solar\\_abundances](#) (class in Chempy.solar\_abundance), 24  
[SSP](#) (class in Chempy.weighted\_yield), 25  
[SSP\\_parameters](#) (Chempy.parameter.ModelParameters attribute), 19  
[SSP\\_parameters\\_to\\_optimize](#) (Chempy.parameter.ModelParameters attribute), 19  
[SSP\\_wrap](#) (class in Chempy.wrapper), 28  
[star\\_function\(\)](#) (in module Chempy.data\_to\_test), 14  
[starformation\\_efficiency](#) (Chempy.parameter.ModelParameters attribute), 22  
[start](#) (Chempy.parameter.ModelParameters attribute), 22  
[stellar\\_identifier](#) (Chempy.parameter.ModelParameters attribute), 22  
[stellar\\_identifier\\_list](#) (Chempy.parameter.ModelParameters attribute), 22  
[stellar\\_lifetimes](#) (Chempy.parameter.ModelParameters attribute), 22  
[stellar\\_lifetimes\\_index](#) (Chempy.parameter.ModelParameters attribute), 22  
[stellar\\_lifetimes\\_list](#) (Chempy.parameter.ModelParameters attribute), 22  
[stochastic\\_IMF](#) (Chempy.parameter.ModelParameters attribute), 22  
[stochastic\\_sampling\(\)](#) (Chempy.imf.IMF method), 16  
[summary\\_pdf](#) (Chempy.parameter.ModelParameters attribute), 22

## T

[testing\\_output](#) (Chempy.parameter.ModelParameters attribute), 22  
[Thielemann\(\)](#) (Chempy.yields.SN1a\_feedback method), 30  
[time\\_delay\\_functional\\_form](#) (Chempy.parameter.ModelParameters attribute), 22  
[time\\_delay\\_functional\\_form\\_list](#) (Chempy.parameter.ModelParameters attribute), 22  
[time\\_delay\\_index](#) (Chempy.parameter.ModelParameters attribute), 22  
[time\\_steps](#) (Chempy.parameter.ModelParameters attribute), 22

[to\\_optimize](#) (Chempy.parameter.ModelParameters attribute), 22  
[tol\\_minimization](#) (Chempy.parameter.ModelParameters attribute), 22  
[total\\_mass](#) (Chempy.parameter.ModelParameters attribute), 22

## U

[use\\_mock\\_data](#) (Chempy.parameter.ModelParameters attribute), 22

## V

[Ventura\(\)](#) (Chempy.yields.AGB\_feedback method), 29  
[verbose](#) (Chempy.parameter.ModelParameters attribute), 22

## W

[wildcard\\_likelihood\\_function\(\)](#) (in module Chempy.data\_to\_test), 14

## Y

[yield\\_comparison\\_plot\(\)](#) (in module Chempy.data\_to\_test), 14  
[yield\\_plot\(\)](#) (in module Chempy.data\_to\_test), 15  
[yield\\_table\\_name\\_1a](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_1a\\_index](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_1a\\_list](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_agb](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_agb\\_index](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_agb\\_list](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_hn](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_hn\\_index](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_hn\\_list](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_sn2](#) (Chempy.parameter.ModelParameters attribute), 22  
[yield\\_table\\_name\\_sn2\\_index](#) (Chempy.parameter.ModelParameters attribute), 22

yield\_table\_name\_sn2\_list  
(Chempy.parameter.ModelParameters attribute), [22](#)

## Z

zero\_model\_error (Chempy.parameter.ModelParameters attribute), [22](#)