pySME usage – a quick guide

Mingjie Jian

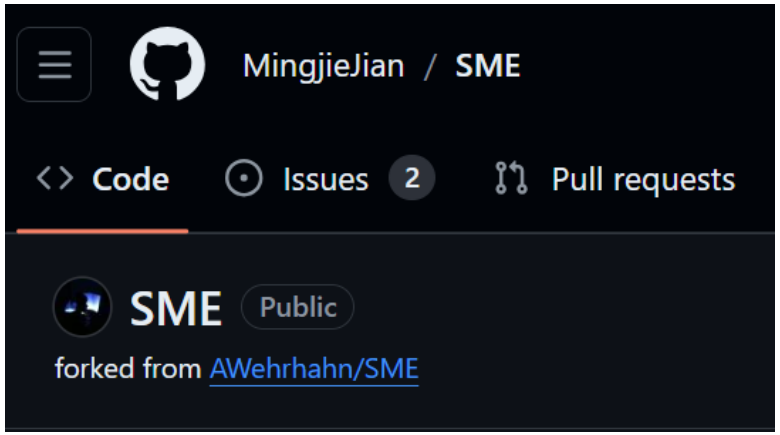
Stockholm University

2024-07-13 @ABDEC 2024

PySME

Spectroscopy Made Easier

- Spectroscopy Made Easy: spectral synthesis and parameter determination (C++ and Fortran)
  - Valenti & Piskunov 1996
- IDL wrapper: IDLSME
- Python wrapper: pysme

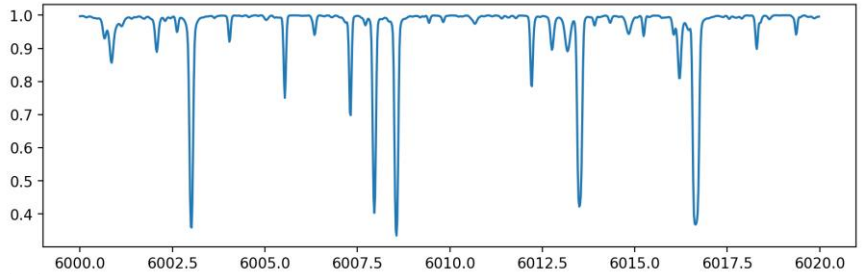


PySME
Spectroscopy Made Easier

easy to use

Spectral synthesis

```
• from pysme.sme import SME_Structure
• from pysme.linelist.vald import ValdFile
• from pysme.synthesize import synthesize_spectrum

• sme = SME_Structure()
• sme.teff, sme.logg, sme.monh = 5772, 4.44, 0
• sme.linelist = ValdFile('line.list')
• sme.wave = np.arange(6000, 6020, 0.02)
• sme = synthesize_spectrum(sme)
```



Parameter fitting

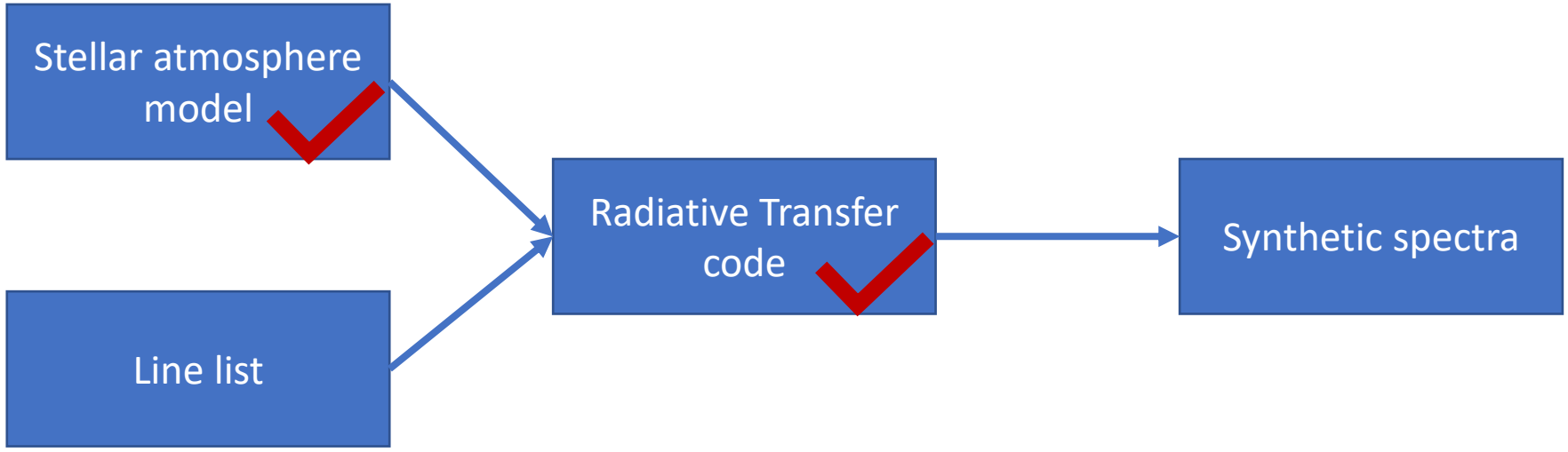
```
• from pysme.sme import SME_Structure
• from pysme.solve import solve

• sme = SME_Structure()
• sme.teff, sme.logg, sme.monh = 5772, 4.44, 0
• sme.linelist = ValdFile('line.list')
• sme.wave = wave_observe
• sme.spec = flux_observe
• sme_fit = solve(sme_fit, ['teff', 'logg', 'monh', 'vmic', 'vsini'])
• sme_fit = solve(sme_fit, ['abund Mg', 'abund Al'])
```

```
INFO - teff        6852.00437 +- 73.763
INFO - logg        4.32846 +- 0.10283
INFO - monh        -0.01311 +- 0.030663
INFO - vmic        2.16246 +- 0.11556
INFO - vsini       3.39724 +- 0.84919
INFO - v_rad       [0.] +- [[0. 0.]]
```

PySME
Spectroscopy Made Easier

easy to use

- MOOG, Turbospectrum, …
  - pymoog

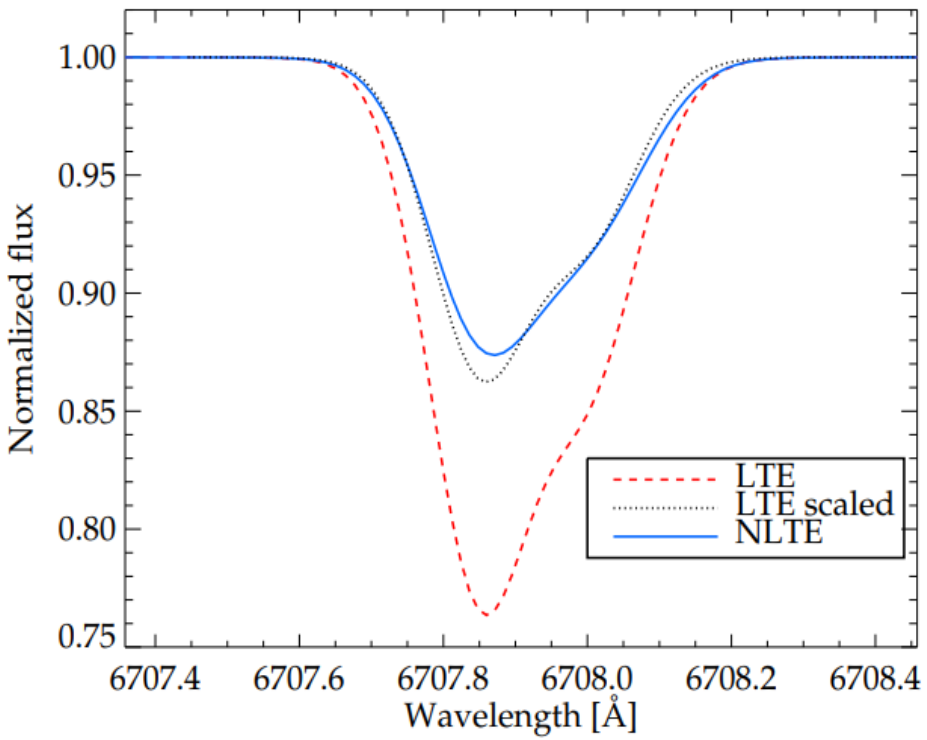


Can be (easily) download from VALD website

PySME
Spectroscopy Made Easier

accurate

- Local thermal equilibrium
  - Can use Saha-Boltzman distribution
  - Simplify the calculation
  - Not correct for some spectral lines
- Non-LTE / NLTE
  - Use pre-computed departure coefficient grids
  - Correct population

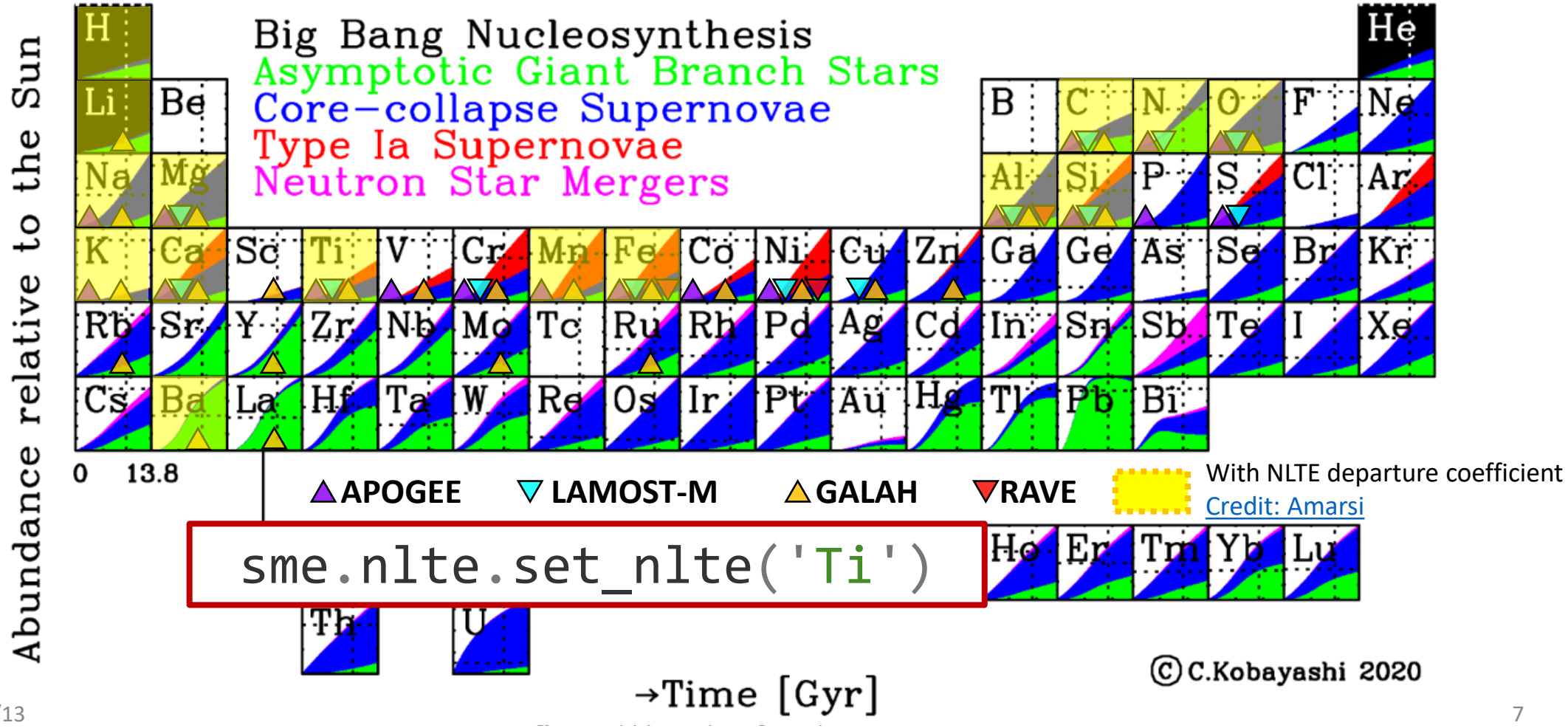


Lind+2013

PySME

Spectroscopy Made Easier

accurate



pySME how-tos

How to use pySME? Questions and answers

How to install pysme?

- For now: Mingjie's github
  - git clone https://github.com/MingjieJian/SME
  - cd SME
  - pip install .
- Long-term: `pip install pysme-astro`

- How large of the disk volume is needed?
  - 5-50G (in ~/.sme)

github.com/MingjieJian/SME

How to generate a synthetic spectra?

```
• from pysme.sme import SME_Structure
• from pysme.linelist.vald import ValdFile
• from pysme.synthesize import synthesize_spectrum

• sme = SME_Structure()
• sme.teff, sme.logg, sme.monh = 5772, 4.44, 0
• sme.linelist = ValdFile('line.list')
• # Either
• sme.wran = [[6700, 6800]]
• # Or
• sme.wave= np.array(6700, 6800, 0.02)
• sme = synthesize_spectrum(sme)
• wave, flux = sme.wave[0], sme.synth[0]
```

- Make the wavelength range of wave/wran consistent with that of the line list.
- Avoid using large number of lines in synthesis – it would take ages. Consider doing it chunk by chunk.

How to set the elemental abundances?

- `sme_fit.abund['P'] += [P/Fe] - sme_fit.monh`
- `sme_fit.abund['Be'] = A(Be) - sme_fit.monh`

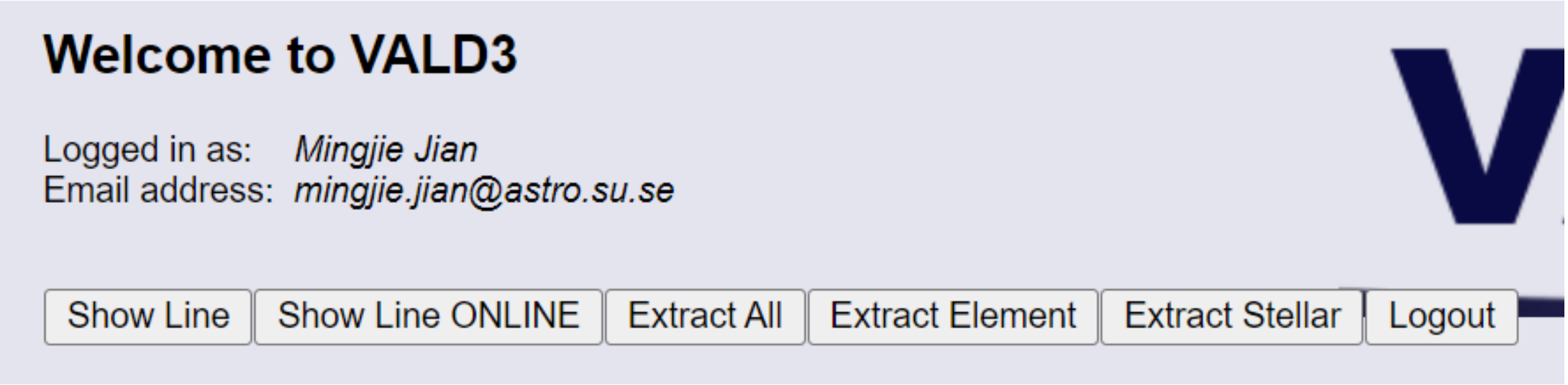
- Always check if `sme_fit.abund` looks reasonable after you modify it!

How to apply broadening?

- Microturbulance
  - sme.vmic
- Macroturbulance
  - sme.vmac
- Stellar rotation
  - sme.vsini
- Instrumental broadening
  - sme.iptype = gauss/sinc/table
  - sme.ipres = 50000

How to get my line list?

- pysme is compatible with the VALD line list database:
  - http://vald.astro.uu.se/~vald/php/vald.php
- Register an account then log in.



How to get my line list?

Show Line ONLINE

Show Line - ONLINE EXTRACTION - EXPERIMENTAL

Approximate wavelength [ ] Å (air)

Wavelength window [ ] Å (air)

Element [+ ionization] [ ]

Linelist configuration

- (●) Default
- ( ) Custom

Isotopic scaling of oscillator strength

- (●) On
- ( ) Off

*(Takes precendence over the default value set in the **unit selection** dialog)*

[Submit request] [Reset form]

How to get my line list?

Extract All

Starting wavelength :	Å (air)
Ending wavelength :	Å (air)
Extraction format :	○ Short format ◉ Long format
Retrieve data via	○ Email ◉ FTP
Hyperfine structure	☑ Include HFS splitting
Require lines to have a known value of :	☐ Radiative damping constant ☐ Stark damping constant ☐ Van der Waals damping constant ☐ Landé factor ☐ Term designation
Linelist configuration	○ Default ◉ Custom
Unit selection	*Energy unit: eV - Medium: air - Wavelength unit: angstrom - VdW syntax: default*
Optional comment for request	
Submit request	Reset form

- FTP: Maximum 100000 lines per query.
- Avoid too many lines
  - Remove unnecessary molecular lines (e.g, TiO for FGK type stars)
  - Query the database chunk by chunk.

How to fit paras?

```
• from pysme.sme import SME_Structure
• from pysme.solve import solve

• sme = SME_Structure()
• sme.teff, sme.logg, sme.monh = 5772, 4.44, 0
• sme.linelist = ValdFile('line.list')
• sme.wave = wave_observe
• sme.spec = flux_observe
• sme.uncs = uncs_observe
• sme_fit = solve(sme_fit, ['teff', 'logg', 'monh', 'vmic',
  'vsini'])
• sme_fit = solve(sme_fit, ['abund Mg', 'abund Al'])
```

Always think of what parameters to fit and which wavelength ranges to use!

Line list

The VALD short\long format output

Line list loading and manipulation

- `sme.linelist = ValdFile('line.list')`

```
        species     wlcent  gflog    excit  j_lo    e_upp  j_up  lande_lower  \
655774     Pr 2  4800.0060 -3.868   0.2162   5.0   2.7985   5.0         0.86
655776     Pr 2  4800.0072 -3.090   0.2162   5.0   2.7985   5.0         0.86
655785     La 1  4800.0170 -0.640   1.2350   3.5   3.8173   2.5         0.89
655786     Fe 2  4800.0171 -1.231  12.8714   3.5  15.4536   3.5         1.24
655790     CH 1  4800.0210 -2.532   1.0838  20.5   3.6660  20.5        99.00
...         ...        ...    ...      ...   ...      ...   ...          ...
```

- The line list object is very similar with pandas.DataFrame.
  - `linelist[linelist['species'] == 'Fe 1']`
  - `linelist[linelist['species'] != 'TiO 1']`
  - `linelist[linelist['wlcent'] > 5500]`

Line list – VALD download

- Only use “long” format if using NLTE
  - Since short format will lose the term configuration required by NLTE calculation.
- When using “extract stellar”
  - Set the detection threshold to ~0.001 (0 means all lines included, 1 means no lines included).

Line list – ValdFile object 1

But it doesn't work for VALD line list – maybe need external line depth?

- acknowledgement
  - Acknowledgement string for using VALD
- add
  - Add a new line to the existing linelist
- append
  - Append a linelist to this one
- atomic
  - list(float) of size (nlines, 8): Data array passed to C library
- citation_info
  - Self explained
- columns (? property)
- cull
  - Remove lines from the linelist that are weaker than the cutoff
- cull_percentage
  - Remove a percentage of the lines in the linelist, removing the weakest lines first
- extra
  - list(float) of size (nlines, 3): additional line level information for NLTE calculation
- from_dict

Line list – ValdFile object 2

- fom_IDL_SME
  - extract LineList from IDL SME structure keywords
- guess_format (?)
- identify_valdtype
  - Determines whether the file was created with extract_all, extract_stellar, or extract_element and whether it is in long or short format
- index (?)
- load
  - Read line data file from the VALD extract service
- loads (?)
- lulande
  - list(float) of size (nlines, 2): Lower and Upper Lande factors
- medium (?)
- mro
  - Return a type's method resolution order.
- parse_abund
  - Parse VALD abundance lines from a VALD line data file
- parse_columns (?)
- parse_header
  - Parse header line from a VALD line data file
  - and sets the internal parameters

Line list – ValdFile object 3

- parse_line_error
  - Transform Line Error flags into relative error values
- parse_linedata
  - Parse line data from a VALD line data file
- parse_nlines (?)
- parse_references (?)
- parse_valdatmo
  - Parse VALD model atmosphere line from a VALD line data file
- sort
  - Sort the linelist
- species
  - list(str) of size (nlines,): Species name of each line
- string_columns
  - Self explained
- to_dict
  - Self explained
- trim
  - Remove lines from the linelist outside the specified wavelength range

VALD linelist format – 1

species:	A string identifier including the element and ionization state or the molecule
atom number:	Identifies the species by the atomic number (i.e. the number of protons)
ionization:	The ionization state of the species, where 1 is neutral (?)
wlcent:	The central wavelength of the line in Angstrom
excit:	The excitation energy in ?
gflog:	The log of the product of the statistical weight of the lower level and the oscillator strength for the transition.
gamrad:	The radiation broadening parameter
gamqst:	A broadening parameter
gamvw:	van der Waals broadening parameter
lande:	The lande factor
depth:	An arbitrary depth estimation of the line
reference:	A citation where this data came from

ignored in the radiative transfer calculations and therefore does not need to be set

VALD linelist format – 2

lande_lower:	The lower Lande factor
lande_upper:	The upper Lande factor
j_lo:	The spin of the lower level
j_up:	The spin of the upper level
e_upp:	The energy of the upper level
term_lower:	The electron configuration of the lower level
term_upper:	The electron configuration of the upper level
error:	An uncertainty estimate for this linedata

The SME structure - 1

- abund -> 
  - elemental abundances
- accft
  - `leastsquares_ftol`
- accgt
  - `leastsquares_gtol`
- accrt
  - `float: minimum accuracy for synthethized spectrum at wavelength grid points in sme.wint`
- accwi
  - `float: minimum accuracy for linear spectrum interpolation vs. wavelength.`
- accxt
  - `leastsquares_xtol`

- atmo -> 
  - model atmosphere data
- atomic
  - Atomic linelist data, usually passed to the C library
- citation
  - A method, self explained
- citation_info
  - A string, self explained
- cont
  - continuum intensities
- create_citation
  - A method, self explained
- cscale
  - Continumm polynomial coefficients for each wavelength segment

The SME structure – 2

- cscale_bounds
  - bounds for the continuum parameters
- cscale_degree
  - Polynomial degree of the continuum as determined by cscale_flag
- cscale_flag
  - Flag that describes how to correct for the continuum
- cscale_ftol
  - tolerance for the continuum least squares fit
- cscale_gtol
  - tolerance for the continuum least squares fit
- cscale_jac
  - str: jacobian approximation used in the continuum fit
- cscale_loss
  - str: loss function for the continuum fit
- cscale_method
  - str: least squares method used in the continuum fit

The SME structure – 3

- cscale_type
  - str: Flag that determines the algorithm to determine the continuum
- cscale_xscale
  - array of 'jac', Scale of each continuum parameter
- cscale_xtol
  - float: tolerance for the continuum least squares fit
- fitparameters
  - list: parameters to fit
- fitresults
  - Fitresults: fit results data
- from_dict
- gam6
  - float: van der Waals scaling factor
- h2broad
  - bool: Whether to use H2 broadening or not
- id
  - str: DateTime when this structure was created

The SME structure – 4

- import_mask
  - Import the mask of another sme structure and apply it to this one
- ip_x
  - array: Instrumental broadening table in x direction
- ip_y
  - array: Instrumental broadening table in y direction
- ipres
  - float, array: Instrumental resolution for instrumental broadening
- iptype
  - str: instrumental broadening type
- leastsquares_ftol
  - float: minimum accuracy of the best fit cost
- leastsquares_gtol
  - float: minimum accuracy of the gradient of the least squares fit
- leastsquares_jac
  - str: leastsquares jacobian calculation, see scipy least_squares for details, default: '2-point'
- leastsquares_loss
  - str: leastsquares loss to use, see scipy least_squares for details, default: 'linear'

The SME structure – 5

- leastsquares_method
  - str: leastsquares method to use, see scipy least_squares for details, default: 'dogbox'.
- leastsquares_xscale
  - str, arraylike: leastsquare x-scale to use, see scipy least_squares for details, default: 1
- leastsquares_xtol
  - float: minimum accuracy of the parameters in the fitting procedure
- linelist
  - LineList: spectral line information
- load
  - Load SME data from disk
- log
  - float: surface gravity in log10(cgs)
- mask
  - Iliffe_vector of shape (nseg, ...): mask defining good and bad points for the fit
- mask_bad, mask_cont, mask_good, mask_line, mask_vrad

The SME structure – 6

- meta
  - dict: Arbitrary extra information
- monh
  - float: metallicity in log scale relative to the base abundances
- mu
  - mu=cos(theta) values to calculate radiative transfer at mu values
- nlte ->

  - NLTE: nlte calculation data
- nmu
  - Number of mu array
- normalize_by_continuum
  - bool: Whether to normalize the synthetic spectrum by the synthetic continuum spectrum or not
- nseg
  - int: Number of wavelength segments

The SME structure – 7

- save
  - Save the whole SME structure to disk.
- spec
  - observed spectrum
- species
  - Names of the species of each spectral line
- specific_intensities_only
  - bool: Whether to keep the specific intensities or integrate them together
- synth
  - synthetic spectrum
- system_info
  - information about the host system running the calculation for debugging
- teff
  - float: effective temperature in Kelvin
- telluric
  - telluric spectrum that is multiplied with synth during the fit

The SME structure – 8

- to_dict
- uncs
  - uncertainties of the observed spectrum
- version
  - str: PySME version used to create this structure
- vmac
  - float: macro turbulence in km/s
- vmic
  - float: micro turbulence in km/s
- vrad
  - radial velocity of each segment in km/s
- vrad_bounds
  - float: radial velocity limits in km/s
- vrad_flag
  - flag that determines how the radial velocity is determined
- vrad_ftol
  - tolerance for the radial velocity least squares fit
- vrad_gtol

The SME structure – 9

- vrad_jac
  - str: jacobian approximation used in the radial velocity fit
- vrad_limit
- vrad_loss
  - str: loss function for the radial velocity fit
- vrad_method
  - str: least squares method used in the radial velocity fit
- vrad_xscale
  - array or 'jac': scale of the vrad parameter
- vrad_xtol
  - float: tolerance for the radial velocity least squares fit
- vsini
  - projected rotational velocity in km/s
- wave
  - wavelength
- wran
  - beginning and end wavelength points of each segment

Abundance – 1 (back: )

- This is the SME_Structure.abund object.

- citation_info
- elem
  - Return the standard abbreviation for each element.
- elem_dict
  - Return the position of each element in the raw array
- empty_pattern()
  - Return an abundance pattern with value None for all elements.
- from_dict
- fromtype
  - Return a copy of the input abundance pattern, transformed from the input type to the 'H=12' type.
- get_element()
  - Get the abundance value of an element
- get_pattern()
  - Transform the specified abundance pattern from type used internally by SME to the requested type.
- monh
  - float: metallicity in log scale relative to the base abundances

Abundance – 2

- pattern
  - Abundance pattern in the initial format
- set_pattern_by_name()
  - Set the abundance pattern to one of the predefined options
- set_pattern_by_value()
  - Set the abundance pattern using an dict of pattern value
- solar
  - Return solar abundances of asplund 2009
- to_dict
  - Convert abundance pattern to dict.
- totype
  - Return a copy of the input abundance pattern, transformed from the 'H=12' type to the output type.
- type
  - Self explained.
- update_pattern
  - Update the abundance pattern for several elements at once
  - Be careful on the format

Atmosphere – 1

- This is the SME_Structure.atmo object.

- abund
  - elemental abundances. May be different from SME_structure.abund
- citation
- citation_info
- create_citation
- depth
  - str: flag that determines whether to use RHOX or TAU for calculations
- dtype (?)
- from_dict
- geom
  - str: the geometry of the atmopshere model
- height
  - array: height of the spherical model
- interp
  - str: flag that determines whether to use RHOX or TAU for interpolation

Atmosphere – 2

- logg
- lonh
  - L/H value
- method
  - str: whether the data source is a grid or a fixed atmosphere
- monh
- names
  - The names of the attributes
- ndep
  - N of depth
- opflag
  - opacity flags
- radius
  - float: radius of the spherical model
- rho
  - array: density profile
- rhox
  - array: mass column density

Atmosphere – 3

- source
  - str: datafile name of this data, or atmosphere grid/file
- tau
  - array: continuum optical depth
- teff
- temp
  - array: temperature profile in Kelvin
- to_dict
- vturb
  - float: turbulence velocity in km/s
- wlstd
  - float: wavelength standard deviation
- xna
  - array: number density of atoms in 1/cm**3
- xne
  - array: number density of electrons in 1/cm**3

NLTE – 1

- This is the SME_Structure.nlte object.

- abund_format
  - str: which abundance format to use for comparison
- citation
- citation_info
- create_citation
- elements
  - list: elements for which nlte calculations will be performed
- first (?)
- flags
  - array: contains a flag for each line, whether it was calculated in NLTE (True) or not (False)
- from_dict

NLTE – 2

- get_grid
  - Read and interpolate the NLTE grid for the current element and parameters
- grid_data
- grids
  - dict: nlte grid datafiles for each element
- min_energy_diff
  - float: difference between energy levels that are still matched. If None will default to the smallest non zero difference between energy levels in the grid.
- remove_nlte
  - Remove an elementfrom the NLTE calculations
- selection
  - str: which selection algorithm to use to match linelist and departure coefficients
- solar
  - str: defines which default to use as the solar metallicitiies
  - Not used now?

NLTE – 3

- sub_grid_size
  - array of shape (4,): defines size of nlte grid cache.Each entry is for one parameter abund, teff, logg, monh
- to_dict
- update_coefficients
  - pass departure coefficients to C library

Line by line description for solve

- 730: the fun is _residuals, used for calculating the residual between observed and synthesized spectra;
- 731: jac is the Jacobian matrix of the fun, i.e., residuals. Here jac is a callable, so pysme have its own way to calculate the Jacobian matrix, and least_square function will directly use it. Note that it is the same as "2-point".
- x0 is the initial parameter values.

```
728             with print_to_log():
729                 res = least_squares(
730                     self._residuals,
731                     jac=self._jacobian,
732                     x0=p0,
733                     bounds=bounds,
734                     loss=sme.leastsquares_loss,
735                     f_scale=self.f_scale,
736                     method=sme.leastsquares_method,
737                     x_scale=sme.leastsquares_xscale,
738                     # These control the tolerance, for early termination
739                     # since each iteration is quite expensive
740                     xtol=sme.accxt,
741                     ftol=sme.accft,
742                     gtol=sme.accgt,
743                     verbose=2,
744                     args=(sme, spec, uncs, mask),
745                     kwargs={
746                         "bounds": bounds,
747                         "segments": segments,
748                         "step_sizes": step_sizes,
749                         "method": sme.leastsquares_jac,
750                     },
751                 )
752                 # The jacobian is altered by the loss function
753                 # This lets us keep the original for our uncertainty estimate
754                 res.jac = self._latest_jacobian
755
```

Line by line description for solve

- bounds is the bounds on the paras. pysme will give bounds on Teff, logg and monh depend on the grids, on velocities from 0 to c, no abunds if available (?).
- loss is the loss function for estimating how similar the spectra are. pysme use linear loss function.
- f_scale is the scaled factor for the spectra, but no effect with loss=linear (?). pysme has a setting of f_scale (a number).

```
728                 with print_to_log():
729                     res = least_squares(
730                         self._residuals,
731                         jac=self._jacobian,
732                         x0=p0,
733                         bounds=bounds,
734                         loss=sme.leastsquares_loss,
735                         f_scale=self.f_scale,
736                         method=sme.leastsquares_method,
737                         x_scale=sme.leastsquares_xscale,
738                         # These control the tolerance, for early termination
739                         # since each iteration is quite expensive
740                         xtol=sme.accxt,
741                         ftol=sme.accft,
742                         gtol=sme.accgt,
743                         verbose=2,
744                         args=(sme, spec, uncs, mask),
745                         kwargs={
746                             "bounds": bounds,
747                             "segments": segments,
748                             "step_sizes": step_sizes,
749                             "method": sme.leastsquares_jac,
750                         },
751                     )
752                     # The jacobian is altered by the loss function
753                     # This lets us keep the original for our uncertainty estimate
754                     res.jac = self._latest_jacobian
755
```

Line by line description for solve

- method is the algorithm to perform minimization. pysme default is dogbox.
- x_scale is the scale factor for the wavelength. pysme default is 1.

```
728                 with print_to_log():
729                     res = least_squares(
730                         self._residuals,
731                         jac=self._jacobian,
732                         x0=p0,
733                         bounds=bounds,
734                         loss=sme.leastsquares_loss,
735                         f_scale=self.f_scale,
736                         method=sme.leastsquares_method,
737                         x_scale=sme.leastsquares_xscale,
738                         # These control the tolerance, for early termination
739                         # since each iteration is quite expensive
740                         xtol=sme.accxt,
741                         ftol=sme.accft,
742                         gtol=sme.accgt,
743                         verbose=2,
744                         args=(sme, spec, uncs, mask),
745                         kwargs={
746                             "bounds": bounds,
747                             "segments": segments,
748                             "step_sizes": step_sizes,
749                             "method": sme.leastsquares_jac,
750                         },
751                     )
752                     # The jacobian is altered by the loss function
753                     # This lets us keep the original for our uncertainty estimate
754                     res.jac = self._latest_jacobian
755
```

Line by line description for solve

- xtol is the tolerance for termination by the change of the independent variables (fitting parameters).
- ftol is the tolerance for termination by the change of the dependent variables (flux).
- gtol is the tolerance for termination by the norm of the gradient.
- The iteration will stop if any these three tol are smaller than the threshold (?).

```
728             with print_to_log():
729                 res = least_squares(
730                     self._residuals,
731                     jac=self._jacobian,
732                     x0=p0,
733                     bounds=bounds,
734                     loss=sme.leastsquares_loss,
735                     f_scale=self.f_scale,
736                     method=sme.leastsquares_method,
737                     x_scale=sme.leastsquares_xscale,
738                     # These control the tolerance, for early termination
739                     # since each iteration is quite expensive
740                     xtol=sme.accxt,
741                     ftol=sme.accft,
742                     gtol=sme.accgt,
743                     verbose=2,
744                     args=(sme, spec, uncs, mask),
745                     kwargs={
746                         "bounds": bounds,
747                         "segments": segments,
748                         "step_sizes": step_sizes,
749                         "method": sme.leastsquares_jac,
750                     },
751                 )
752                 # The jacobian is altered by the loss function
753                 # This lets us keep the original for our uncertainty estimate
754                 res.jac = self._latest_jacobian
755
```

Line by line description for solve

- verbose is the level of algorithm's verbosity, default will display the fitting details.

```
728                 with print_to_log():
729                     res = least_squares(
730                         self._residuals,
731                         jac=self._jacobian,
732                         x0=p0,
733                         bounds=bounds,
734                         loss=sme.leastsquares_loss,
735                         f_scale=self.f_scale,
736                         method=sme.leastsquares_method,
737                         x_scale=sme.leastsquares_xscale,
738                         # These control the tolerance, for early termination
739                         # since each iteration is quite expensive
740                         xtol=sme.accxt,
741                         ftol=sme.accft,
742                         gtol=sme.accgt,
743                         verbose=2,
744                         args=(sme, spec, uncs, mask),
745                         kwargs={
746                             "bounds": bounds,
747                             "segments": segments,
748                             "step_sizes": step_sizes,
749                             "method": sme.leastsquares_jac,
750                         },
751                     )
752                     # The jacobian is altered by the loss function
753                     # This lets us keep the original for our uncertainty estimate
754                     res.jac = self._latest_jacobian
755
```

Output description for solve

- Iteration: iteration number
- nfev: number of the function called
- Cost: the value of target function (_residuals)
- Cost reduction: The reduction amount of the target function compared with the last iteration.
  - ftol termination
- Step norm: The change in the norm of independent parameters (fitting parameters)
  - xtol termination
- Optimality: The norm of the gradient.
  - gtol termination

```
INFO - Don't forget to cite your sources. Use sme.citation()
WARNING - SME Structure has no uncertainties, using all ones instead
INFO - Fitting Spectrum with Parameters: abund C
INFO -    Iteration     Total nfev        Cost      Cost reduction    Step norm     Optimality
INFO -        0              1         1.3989e-02                                    7.43e-04
INFO -        1              2         1.3973e-02      1.61e-05       3.75e-02       5.00e-05
INFO - `gtol` termination condition is satisfied.
INFO - Function evaluations 2, initial cost 1.3989e-02, final cost 1.3973e-02, first-order optimality 5.00e-05.
INFO - Abund c          8.42753 +- 0.12672
INFO - v_rad            [0.] +- [[0. 0.]]
```