pyTOPSScrape

Release 0.5

Thomas Boudreaux

Sep 09, 2022

CONTENTS

CHAPTER

ONE

PYTOPSSCRAPE

A package which aims to make the programmatic retrieval and use of high temperature radiative opacity tables from LANL somewhat simple

pyTOPSScrape provides both a command line and a python interface. The command line interface runs through the generateTOPStables script (which will have been installed to your path when you installed this package)

The python interface can mimic the full command line interface (including error checking and rate limiting) using the full_run function. If however, you wish to dig down to a more granular level the api module includes both query and convert sub modules which may be composed as needed.

CHAPTER

TWO

INSTALLATION

2.1 PyPi

```
>>> pip install pyTOPSScrape
```

2.2 GitHub

```
>>> git clone https://github.com/tboudreaux/pytopsscrape.git
>>> cd pytopsscrape
>>> python setup.py install
```

2.3 Command Line Usage Example

```
>>> generateTOPStables GS98.abun rescalings.dat -d ./rawOutput -o GS98.opac -j 20
```

CHAPTER

THREE

INPUT FILE FORMATS

pyTOPSScrape requires two input files to run. One (the first positional argument and hereafter the 'composition file') describes the base composition. The second positional argument (hereafter the 'map file') describes the set of classical compositions which will be queried from the TOPS web form. Each of these compositions will be a rescaling of the base composition (therefore the metal mass fractions wrt. Z will be maintained)

3.1 Composition file

The composition file should be in the following form

```
#STD [Fe/H] [alpha/Fe] [C/Fe] [N/Fe] [O/Fe] [r/Fe] [s/Fe] C/O X Y,Z
F -1.13 0.32 -0.43 -0.28 0.31 -1.13 -1.13 0.10 0.7584 0.2400,1.599E-03
#H He Li Be B C N O F Ne
12.00 10.898 -0.08 0.25 1.57 6.87 6.42 7.87 3.43 7.12
#Na Mg Al Si P S Cl Ar K Ca
5.11 6.86 5.21 6.65 4.28 6.31 -1.13 5.59 3.90 5.21
#Sc Ti V Cr Mn Fe Co Ni Cu Zn
2.02 3.82 2.80 4.51 4.30 6.37 3.86 5.09 3.06 2.30
#Ga Ge As Se Br Kr Rb Sr Y Zr
0.78 1.39 0.04 1.08 0.28 0.99 0.26 0.61 1.08 1.45
#Nb Mo Tc Ru Rh Pd Ag Cd In Sn
-0.80 -0.38 -99.00 -0.51 -1.35 -0.69 -1.32 -0.55 -1.46 -0.22
#Sb Te I Xe Cs Ba La Ce Pr Nd
-1.25 -0.08 -0.71 -0.02 -1.18 1.05 -0.03 0.45 -1.54 0.29
#Pm Sm Eu Gd Tb Dy Ho Er Tm Yb
-99.00 -1.30 -0.61 -1.19 -1.96 -1.16 -1.78 -1.34 -2.16 -1.42
#Lu Hf Ta W Re Os Ir Pt Au Hg
-2.16 -1.41 -2.38 -1.41 -2.00 -0.86 -0.88 -0.64 -1.34 -1.09
#Tl Pb Bi Po At Rn Fr Ra Ac Th
-1.36 -0.51 -1.61 -99.00 -99.00 -99.00 -99.00 -99.00 -99.00 -2.20
#Pa U
-99.00 -2.80
```

3.2 Map file

The map file should be in the following form

```
0.75,0.24,0.01
0.75,0.23,0.02
```

Where each row is X,Y,Z. The number of rows in this file will correspond to the number of queries issued against the TOPS web form

CHAPTER

FOUR

TESTING

pyTOPSScrape ships with a number of tests which should be run to make sure that it installed correctly on your system. Additionally, as it is reliant on an external server it is a certainty that one day it will break. To run the tests a script has been included. From the pyTOPSScrape root directory

```
>>> cd tests
>>> ./runTests.sh
```

CHAPTER

FIVE

ETIQUETTE & CACHEING

pyTOPSScrape makes use web servers hosted at Los Alamos National Labs (LANL). Before releasing this software I spoke with the T-1 group at LANL and received their assent. However, try to limit requests made against their web servers as much as possible. Obviously, if you are querying a few hundred tables because your stellar evolution code needs a few hundred opacity tables there is little to be done; however, do try and make sure that you have sorted out any and all bugs or typos in your input files before you query so that you won't have to go back and query multiple times. We want to be respectful of of the generosity of LANL here!

Additionally, pyTOPSScrape caches the raw query results to whatever directory is specified by the -d or –outputDirectory flag. This is so that if you want to implement your own converted you can do so without constantly re running the query functions. These results are cached in whatever directory is set in the –outputDirectory (or -d) command line option. To call the command line interface with cache usage enabled use the –nofetch flag. If you want to fetch tables and don't want to run the conversion set use the –noopal flag.

As an example, if you have already queried the TOPS web form using the command

```
>>> generateTOPStables GS98.abun rescalings.dat -d ./rawOutput -o GS98.opac -j 20 --
↪noopal
```

this will save all the raw output to the directory ./rawOutput (if you run the first example from this docs page this will also cache the results). You can then convert these to DSEP's OPAL format using the command

```
>>> generateTOPStables GS98.abun rescalings.dat -d ./rawOutput -o GS98.opac -j 20 --
↪nofetch
```

Notes

Website [1]_

Paper Describing Opacity Tables [2]_

[1] https://aphysics2.lanl.gov/apps/

[2] Colgan, James, et al. "A new generation of Los Alamos opacity tables." The Astrophysical Journal 817.2 (2016): 116.

CHAPTER

SIX

PYTOPSSCRAPE PACKAGE

6.1 Subpackages

6.1.1 pyTOPSScrape.api package

6.1.1.1 Submodules

6.1.1.2 pyTOPSScrape.api.api module

Author: Thomas M. Boudreaux

Created: September 2021

Last Modified: September 2022

Psuedo API for querying TOPS webform

pyTOPSScrape.api.api.**TOPS_query**(*mixString: str*, *mixName: str*, *nAttempts: int*) → bytes
Query TOPS form and retry n times

Parameters

- **mixString** (*string*) – string in the form of: "massFrac0 Element0 massFrac1 Element1 ..." which will be submitted in the webform for mixture
- **mixName** (*string*) – name to be used in the webform
- **nAttemptes** (*int*) – How many times to reattempt after a failure.

Returns **tableHTML** – Table queried from TOPS cite.

Return type bytes

pyTOPSScrape.api.api.**TOPS_query_async_distributor**(*compList: list*, *outputDirectory: str*, *njobs: int = 10*)
Distributes TOPS query jobs to different threads and gathers the results together. Writes out output.

Parameters

- **comList** (*list*) – 3D list containing the mass frac for each element for each rescaled composition requested. For n compositions this should be of the shape (n, 30, 2). n compositions, 30 elements, then the first element of the last axis is the elemental symbol (i.e. H, He, Li, Be, etc...) and the second element is the mass fraction.
- **outputDirectory** (*str*) – Path to directory to save TOPS query results to.
- **njobs** (*int, default=10*) – Number of concurrent jobs to allow at a time.

pyTOPSScrape.api.api.**call**(*aMap: str*, *aTable: str*, *outputDir: str*, *jobs: int*)

Main TOPS psuedo API call function. Will save results to outputDir with file format OP:IDX_X_Y_Z.dat where IDX is the ID of the composition (parallel to DSEP composition ID), X is the classical Hydrogen mass fraction, Y is the classical Helium mass fraction, and Z is the classical metal mass fraction.

Parameters

- **aMap** (*str*) – Path to the list of classical compositions to be used. List should be given as an ascii file where ecach row is X,Y,Z
- **aTable** (*str*) – Path to chemical abundance table to be used as base composition.
- **outputDir** (*str*) – Path to directory save TOPS query results into
- **jobs** (*int*) – Number of threads to query TOPS webform on

Examples

If you have some map of rescalings you would like to used at "./rescalings.dat" and you have a base composition in the correct form at "./comp.dat" then you can generate and cache the raw output for those rescalings of that composition using

```
>>> call("./rescalings.dat", "./comp.dat", "./cache", 5)
```

This will save the cache results to the folder ./cache (note that this folder must exist before calling call. Moreover, this will query using 5 workers. You may increase this number to make call run faster; however, this will only work to a point. I find that around 20 workes is about the most that gives me any speed increase. This will somewhat depend on your computer though.

pyTOPSScrape.api.api.**parse_table**(*html: bytes*) → str

Parse the bytes table returned from mechanize into a string

Parameters **html** (*bytes*) – bytes table retuend from mechanize bowser at second TOPS submission form

Returns **table** – parsed html soruce in the form of a string

Return type string

pyTOPSScrape.api.api.**query_and_parse**(*compList: list*, *outputDirectory: int*, *i: int*, *nAttempts: int = 10*)

Async coroutine to query TOPS webform, parse the output, and write that to disk.

Parameters

- **comList** (*list*) – 3D list containing the mass frac for each element for each rescaled composition requested. For n compositions this should be of the shape (n, 30, 2). n compositions, 30 elements, then the first element of the last axis is the elemental symbol (i.e. H, He, Li, Be, etc…) and the second element is the mass fraction.
- **outputDirectory** (*str*) – Path to write out results of TOPS webquery
- **i** (*int*) – Index of composition so file name can properly keep track of where it is, even in parallel processing.
- **nAttempts** (*int, default=10*) – Number of time to retry TOPS query before failing out

pyTOPSScrape.api.api.**submit_TOPS_form**(*mixString: str*, *mixName: str*, *massFrac: bool = True*) → bytes

Open the Los Alamos opacity website, submit a given composition and then return the resultant table.

Parameters

- **mixString** (*string*) – string in the form of: "massFrac0 Element0 massFrac1 Element1 ..." which will be submitted in the webform for mixture
- **mixName** (*string*) – name to be used in the webform
- **massFrac** (*bool, default=True*) – Submit as massFrac instead of numberFrac

Returns **tableHTML** – Table quired from TOPS cite.

Return type bytes

6.1.1.3 pyTOPSScrape.api.convert module

Author: Thomas M. Boudreaux

Created: September 2021

Last Modified: September 2022

Main conversion code for TOPS api, responsible for takine many TOPS results and merging them into a single OPAL formate high temperature opacity file.

pyTOPSScrape.api.convert.**TOPS_2_OPAL**(*outputDirectory: str*, *aTable: str*, *aMap: str*, *output: str*, *nonRect: bool = False*)

Main conversoin utility to go between some set of TOPS tables and an OPAl table. Will take a set of 126 TOPS tables where each one is the opacity for one composition over a number of temperature and densities and rearange them into one large file with 126 tables within it. Each table will be over a range of temperatures and R values. To get to R val interpolation is used.

Parameters

- **outputDirectory** (*str*) – Path to directory where TOPS query results are stored
- **aTable** (*str*) – Path to a reference abundance table to use when filling the header with compositional information
- **aMap** (*str*) – Path to the abundance map. This should be an ascii file where each row is X,Y,Z. Each row will correspond to one rescaled composition which will be queried.
- **output** (*str*) – Path to save final OPAL formated table too
- **nonRect** (*bool, default=False*) – Flag to control whether output tables will be rectangular or have their corners cut off in a way consistant which how DSEP expects OPAL tables.

pyTOPSScrape.api.convert.**comp_list_2_dict**(*compList: List[Tuple[str, float, float]]*) → dict

Take a list containing compsoition information for a star in the form of [('Element Symbol', massFraction, numberFraction),...] and convert that into a dictionary of the form: {'Element Symbol': (massFraction, numberFraction),..}.

Parameters **compList** (*Iterator*) – list of the form: [(ElementSymbol, massFrac, numberFrac, ZmassFrac, massFrac Uncertanity, numberFrac Uncertanity, ZMassFrac Uncertanity),...]

Returns Dictionary of the form {'Element':(massFrac, numberFrac, ZmassFrac, massFrac Uncertanity, numberFrac Uncertanity, ZMassFrac Uncertanity),...}

Return type dict

pyTOPSScrape.api.convert.**convert_rho_2_LogR**(*rho: numpy.ndarray*, *LogT: numpy.ndarray*, *RMO: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Maps a given kappa(rho,logT) parameter space onto a kappa(LogR, LogT) field through interpolation. The final field is the field that DSEP needs. :param rho: mass density array of size n :type rho: np.ndarray :param LogT:

LogT array of size m :type LogT: np.ndarray :param RMO: Opacity Array of size m x n :type RMO: np.ndarray

Returns

- **targetLogR** (*np.ndarray(shape=19)*) – Log R values which dsep requires
- **targetLogT** (*np.ndarray(shape=70)*) – Log T values which dsep requires
- **Opacity** (*np.ndarray(shape=(70, 19))*) – Opacity array now interpolated into LogR, LogT space from rho LogT space and sampled at the exact LogR and LogT values required.

pyTOPSScrape.api.convert.**format_OPAL_header**(*compDict: dict*) → str

Writes the header of the opacity table that DSEP expects. This is written to be the same length (and basically the same contents) of the header from the OPACITY project. Not sure if that is required; however, if so I am matching it.

Parameters **compDict** (*dict*) – dictionary in the form: {'Element': (massFrac, numFrac), ... } used to fill up the header with composition information. This is meant to be the "solar" compositon of whatever mix you are using so... [Fe/H] = 0.0, [alpha/H] = 0.0, a(He) = 10.93

Returns The Header to be prepended to the opacity table file

Return type string

pyTOPSScrape.api.convert.**format_OPAL_table**(*tableDict: List[dict]*, *compDict: dict*) → str

Given a dictionary of tables and a composition Dictionary for solar composition in a given mixture (AGSS08, GS98, etc...) merge all the information together into a string which can be written to disk and would be the format of an opacoty project table (what DSEP expects)

Parameters

- **tableDict** (*dict*) – dictionary of table elements, containing a "Summary" entry (metadata) and a "Table" entry. All occcurences of the the string "TNUM" will be replaced with the index+1 of where that table occurs in the file.
- **compDict** (*dict*) – dictionary in the form: {'Element': (massFrac, numFrac), ... } used to fill up the header with composition information. This is meant to be the "solar" compositon of whatever mix you are using so... [Fe/H] = 0.0, [alpha/H] = 0.0, a(He) = 10.93

Returns **OPALFormatted** – Opacity Project formated table as a string which can be written to disk

Return type string

pyTOPSScrape.api.convert.**format_TOPS_to_OPAL**(*TOPSTable: str*, *comp: tuple*, *tnum: int*, *upperNonRect: Optional[numpy.ndarray] = None*) → Tuple[str, float, float, float, str, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Take the path to a table queried from the TOPS web form and fully convert it into a table which can be directly read by DSEP. (Note this function does not write anything to disk; however, the return products can be written to disk)

Parameters

- **TOPSTable** (*string*) – path to table queried from TOPS web form
- **comp** (*dict*) – composition dictionary
- **tnum** (*int*) – table number
- **upperNonRect** (*np.ndarray, optional*) – array describing how to fill the top of the table non rectangurally

Returns

- **metaLine** (*str*) – metadata extracted from table

- **X** (*float*) – Hydrogen mass fraction
- **Y** (*float*) – Helium mass fraction
- **Z** (*float*) – Metal mass fraction
- **fullTable** (*str*) – The full table as a string which may be directly written to a file. Note that this includes all the relevant white space to allow this table to be parsed by DSEP.
- **LogR** (*np.ndarray(shape=19)*) – Log R values which dsep expects
- **LogT** (*np.ndarray(shape=70)*) – Log Temperature values which dsep expects.
- **LogRMO** (*np.ndarray(shape=(70, 19))*) – Log rossland mean opacities for the LogT and LogR arrays

pyTOPSScrape.api.convert.**format_opal_comp_table**(*LogR: numpy.ndarray*, *LogT: numpy.ndarray*, *LogRMO: numpy.ndarray*, *TNUM: int*, *comp: Optional[dict] = None*, *upperNonRect: Optional[numpy.ndarray] = None*, *nonRect: bool = False*) → Tuple[str, str]

Take in all the information from a given TOPS tables and format it to the proper format for DSEP to undersand. Leave in some placeholders so that in future table can be labeld as the proper number.

Parameters

- **LogR** (*ndarray*) – The Log R value array (horizontal axis of table)
- **LogT** (*ndarray*) – The Log Temperature value array (vertical axis)
- **LogRMO** (*ndarray*) – all of the RMO values associated with R and T
- **TNUM** (*int*) – Table number
- **comp** (*dict, optional*) – composition dictionary. If not provided placeholders are left in place so that it may be filled later on
- **upperNonRect** (*ndarray, default=None*) – Array describing how to fill the top of the table non rectangurally This array should be of the shape (nRowsPerTable * nTables, 3). So if you have 5 tables each with 70 rows then this array should have a shape of (350,3). The first column of this array correspond to the table that the row is a member of, the second column correspond to the row in that table that the row is. so the first row of the first table would be at upperNonRect[0,:] = [0,0,...] while the first row of the second table would be at upperNonRect[n,:] [1,0,...]. The final column desribes how many of the elements, counting from the left of the opacity table should be blanked out to 99.999 (the sentinal value DSEP uses for non entries). So a row of [2,55,8] would mean that for the 56th row in the 3rd table blank out the firts 8 opacity values (opacities for the first 8 values of logR).
- **nonRect** (*bool, default = False*) – Flag to control whether output tables will be rectangular or have their corners cut off in a way consistant which how DSEP expects OPAL tables.

Returns

- **metaLine** (*str*) – header line for each table, may or may not have placeholders in it
- **fullTable** (*str*) – full table to be places in opacity file

pyTOPSScrape.api.convert.**rebuild_formated_tables**(*formatedTables: List[dict]*, *interpRMO: numpy.ndarray*, *pContents: numpy.ndarray*, *upperNonRect: Optional[numpy.ndarray]*, *nonRect: bool = False*) → List[dict]

Iterate over a list of opacity tables and a list of desired chemical compositions then replace the contents of the

table list with the newly updated RMOs frmo the interpolation.

Parameters

- **formatedTables** (*list of dicts*) – List of dictionaries holding three axis each. X, Z, and LogRMO. These are the "observed" values to be interpolared
- **interpRMO** (*ndarray*) – RMOs after interpolation to LogR-LogT space from rho-LogT space.
- **pContents** (*np.array(shape=(n, 3))*) – Numpy array of all the compositions of length n. For a dsep n=126. Along the second axis the first column is X, the second is Y, and the third is Z.
- **upperNonRect** (*ndarray*) – Array describing how to fill the top of the table non rectangularly This array should be of the shape (nRowsPerTable * nTables, 3). So if you have 5 tables each with 70 rows then this array should have a shape of (350,3). The first column of this array correspond to the table that the row is a member of, the second column correspond to the row in that table that the row is. so the first row of the first table would be at upperNonRect[0,:] = [0,0,...] while the first row of the second table would be at upperNonRect[n,:] [1,0,...]. The final column desribes how many of the elements, counting from the left of the opacity table should be blanked out to 99.999 (the sentinal value DSEP uses for non entries). So a row of [2,55,8] would mean that for the 56th row in the 3rd table blank out the firts 8 opacity values (opacities for the first 8 values of logR).
- **nonRect** (*bool, default = False*) – Flag to control whether output tables will be rectangular or have their corners cut off in a way consistant which how DSEP expects OPAL tables.

Returns **formatedTables** – List of dictionaries holding three axis each. X, Z, and LogRMO. These have been updated to reflect the compositions in pContents.

Return type list of dicts

6.1.1.4 pyTOPSScrape.api.utils module

Author: Thomas M. Boudreaux

Created: September 2021

Last Modified: September 2021

Utilities to help with the TOPS query api

pyTOPSScrape.api.utils.**format_TOPS_string**(*compList: list*) → str

Format the composition list from pasrse_abundance_file into a string in the form that the TOPS web form expects for a mass fraction input.

Parameters **compList** (*list*) – composition list in the form of: [('Element', massFrac, numFrac),...]

Returns **TOPS_abundance_string** – string in the form of: "massFrac0 Element0 massFrac1 Element1 ..."

Return type string

pyTOPSScrape.api.utils.**validate_extant_tables**(*path: str*, *prefix: str*) → bool

Check if there is a quiried table from TOPS for every number frac file generated by the program passed to call_num_frac.

Parameters

- **path** (*string*) – Path to where the results of the number frac and TOPS query files are stored
- **prefix** (*string*) – start prefix given to all abundance / number frac files

Returns **validated** – Whether or not all number frac files have a corresponding TOPS opacity table

Return type bool

6.1.1.5 Module contents

The api module provides query and convert functionality for pyTOPSScrape.

The function you are most likley interested in here is call().

6.1.2 pyTOPSScrape.err package

6.1.2.1 Submodules

6.1.2.2 pyTOPSScrape.err.err module

6.1.2.3 Module contents

6.1.3 pyTOPSScrape.misc package

6.1.3.1 Subpackages

6.1.3.1.1 pyTOPSScrape.misc.dataFiles package

6.1.3.1.1.1 Module contents

6.1.3.2 Submodules

6.1.3.3 pyTOPSScrape.misc.utils module

Author: Thomas M. Boudreaux

Created: Febuary 2021

Last Modified: September 2022

Opacity utility functions

pyTOPSScrape.misc.utils.**get_target_log_R**() → numpy.ndarray
Get the ndarray for the LogR values that DSEP expects

Returns **targetLogR** – Array of LogR values expected by DSEP in opacity table

Return type np.ndarray

pyTOPSScrape.misc.utils.**get_target_log_T**() → numpy.ndarray
Get the ndarray for the LogT values that DSEP expects

Returns **targetLogT** – Array of LogT values expected by DSEP in opacity table

Return type np.ndarray

pyTOPSScrape.misc.utils.**load_non_rect_map**() → numpy.ndarray

Load the upper non rectabtular map from numpy binary which DSEP requires for the high temperature opacity files.

Returns **upperNonRect** – Upper non rectangular map which DSEP requires.

Return type np.ndarray

6.1.3.4 Module contents

6.1.4 pyTOPSScrape.parse package

6.1.4.1 Submodules

6.1.4.2 pyTOPSScrape.parse.abundance module

Author: Thomas M. Boudreaux

Created: May 2021

Last Modified: September 2022

Module responsible for the parsing and handeling of chemical composition files in the form of

```
#STD [Fe/H] [alpha/Fe] [C/Fe] [N/Fe] [O/Fe] [r/Fe] [s/Fe] C/O X Y,Z
F -1.13 0.32 -0.43 -0.28 0.31 -1.13 -1.13 0.10 0.7584 0.2400,1.599E-03
#H He Li Be B C N O F Ne
12.00 10.898 -0.08 0.25 1.57 6.87 6.42 7.87 3.43 7.12
#Na Mg Al Si P S Cl Ar K Ca
5.11 6.86 5.21 6.65 4.28 6.31 -1.13 5.59 3.90 5.21
#Sc Ti V Cr Mn Fe Co Ni Cu Zn
2.02 3.82 2.80 4.51 4.30 6.37 3.86 5.09 3.06 2.30
#Ga Ge As Se Br Kr Rb Sr Y Zr
0.78 1.39 0.04 1.08 0.28 0.99 0.26 0.61 1.08 1.45
#Nb Mo Tc Ru Rh Pd Ag Cd In Sn
-0.80 -0.38 -99.00 -0.51 -1.35 -0.69 -1.32 -0.55 -1.46 -0.22
#Sb Te I Xe Cs Ba La Ce Pr Nd
-1.25 -0.08 -0.71 -0.02 -1.18 1.05 -0.03 0.45 -1.54 0.29
#Pm Sm Eu Gd Tb Dy Ho Er Tm Yb
-99.00 -1.30 -0.61 -1.19 -1.96 -1.16 -1.78 -1.34 -2.16 -1.42
#Lu Hf Ta W Re Os Ir Pt Au Hg
-2.16 -1.41 -2.38 -1.41 -2.00 -0.86 -0.88 -0.64 -1.34 -1.09
#Tl Pb Bi Po At Rn Fr Ra Ac Th
-1.36 -0.51 -1.61 -99.00 -99.00 -99.00 -99.00 -99.00 -99.00 -2.20
#Pa U
-99.00 -2.80
```

Where each number is a(i) for the ith element and lines starting with # are comments.

pyTOPSScrape.parse.abundance.**a_to_mfrac**(*a*, *amass*, *X*)

Convert $a(i)$ for the i^{th} element to a mass fraction using the expression

$$a(i) = \log(1.008) + \log(F_i) - [\log(X) + \log(m_i)] + 12$$

Or, equivilenetly, to go from $a(i)$ to mass fraction

$$F_i = \left[\frac{X m_i}{1.008}\right] \times 10^{a(i)-12}$$

Where F_i is the mass fraction of the i^{th} element, X is the Hydrogen mass fraction, and m_i is the ith element mass in hydrogen masses.

Parameters

- **a** (*float*) – $a(i)$ for the i^{th} element. For example for He chem might be 10.93. For Hydrogen it would definititionally be 12.
- **amass** (*float*) – Mass of i^{th} element given in atomic mass units.
- **X** (*float*) – Hydrogen mass fraction

Returns **mf** – Mass fraction of i^{th} element.

Return type float

pyTOPSScrape.parse.abundance.**get_atomic_masses**()
Return a dict of atomic masses from Hydrogen all the way to plutonium

Returns **amasses** – Dicionary of atomic masses in atomic mass units indexed by elemental symbol.

Return type dict of floats

pyTOPSScrape.parse.abundance.**get_base_composition**(*aTablePath: str*) → Tuple[list, float, float, float]
For some abundance path return the "base" composition, this is mainly to be used for headers.

Parameters **aTablePath** (*str*) – Path to the abundance table in the form as described in the parseChemFile module documentation

Returns

- *list* – list of the composition in the form [('Element',massFrac,numberFrac),...]
- *float* – Hydrogen mass fraction
- *float* – Helium mass fraction
- *float* – Metal mass fraction

pyTOPSScrape.parse.abundance.**mfrac_to_a**(*mfrac*, *amass*, *X*, *Y*)
Convert mass fracition of a given element to a for that element at a given hydrogen mass fraction using the equation

$$a(i) = \log(1.008) + \log(F_i) - [\log(X) + \log(m_i)]$$

Where F_i is the mass fraction for the i^{th} element and m_i is the mass fraction for the i^{th} element.

Parameters

- **mfrac** (*float*) – Mass fraction of the ith element.
- **amass** (*float*) – Mass of the ith element in atomic mass units.
- **X** (*float*) – Hydrogen mass fraction
- **Y** (*float*) – Helium mass fraction, will be used as reference if X = 0

Returns **a** – a for the ith element

Return type float

pyTOPSScrape.parse.abundance.**open_and_parse**(*path*)
Open and parse the contents of a chemical composition file

Parameters **path** (*str*) – Path to open file

Returns

parsed –

Dictionary with two indexes.

- **Abundance Ratio** Includes the indexes:
  - STD (*str*)
  - [Fe/H] (*float*)
  - [alpha/Fe] (*float*)
  - [C/Fe] (*float*)
  - [N/Fe] (*float*)
  - [O/Fe] (*float*)
  - [r/Fe] (*float*)
  - [s/Fe] (*float*)
  - C/O (*float*)
  - X (*float*)
  - Y (*float*)
  - Z (*float*)
- **RelativeAbundance** Includes an index for each chemical symbol given in the file format definition provided in the module documentation. These are all floats.

Return type dict

pyTOPSScrape.parse.abundance.**open_chm_file**(*path*)

Open a chemical composition file (format defined in the module documentation). Split the contents by line then remove all lines which start with #. Finally split each line by both whitespace and commas.

Parameters **path** (*str*) – Path to file to open

Returns **contents** – List of list of strings. The outter index selects the row, the inner index selectes the column within the row.

Return type list

pyTOPSScrape.parse.abundance.**parse**(*contents: list*) → dict

Parse chem file in the format described in the module documentation.

The abuundance ratios and abundances on the first row are added to a dict under the key ['AbundanceRatio'] and sub indexed by the comments above each entry (Note that these are not read; rather, they are assumed to be the same in every file). The subsequent values (on all other rows) are added to the same dict under the key ['RelativeAbundance'] and sub indexed by their chemical symbols.

Parameters **contents** (*list*) – List of list of strings. The outter index selects the row, the inner index selected the column in the row and at each coordinate is a string which can be cast as a float. The one exception is that string at 0,0 is a charectar.

Returns

extracted –

Dictionary with two indexes.

- **Abundance Ratio** Includes the indexes:

- STD (*str*)
- [Fe/H] (*float*)
- [alpha/Fe] (*float*)
- [C/Fe] (*float*)
- [N/Fe] (*float*)
- [O/Fe] (*float*)
- [r/Fe] (*float*)
- [s/Fe] (*float*)
- C/O (*float*)
- X (*float*)
- Y (*float*)
- Z (*float*)

- **RelativeAbundance** Includes an index for each chemical symbol given in the file format from the module documentation. These are all floats.

Return type dict

pyTOPSScrape.parse.abundance.**parse_abundance_map**(*path: str*) → numpy.ndarray

Parse Hydrogen, Helium, and metal mass fraction out of a csv where each row is one composition, the first column is X, second is Y, and the third is Z. Comments may be included in the file if the first non white space character on the line is a hash.

Parameters **path** (*str*) – Path to the abundance map. This should be an ascii file where each row contains X, Y, and Z (comma delimited with no white space). Each row will define one set of tables to be queried with the idea being that the entire file describes the entire set of tables to be queried.

Returns **pContents** – numpy array of all the compositions of length n where n is the number of rows whos first non white space character was not a hash. For a DSEP n=126. Along the second axis the first column is X, the second is Y, and the third is Z.

Return type np.ndarray(shape=(n,3))

6.1.4.3 pyTOPSScrape.parse.opal module

pyTOPSScrape.parse.opal.**load_opal**(*path: str*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Load both the opacity table as well as the LogR and LogT for each table. LogT and LogR are not actually taken from the table; rather, they are assumed to be the correct LogT and LogR and are simply constructed from the general LogR and LogT constuctors already present in pysep.

Parameters **path** (*str*) – path to opacity table to load.

Returns

- **LogT** (*np.ndarray(dtype='float64')*) – Numpy array of shape (70,) with each required LogT value in it.
- **LogR** (*np.ndarray(dtype='float64')*) – Numpy array of shape (19,) wich each required LogR value in it.

- **p** (*np.ndarray(dtype='float64')*) – array of shape (126, 70, 19) where the first axis is the composition axis for the 126 compositions which dsep expects, the second is the temperature axis, and the third is the R axis.

pyTOPSScrape.parse.opal.**parse_OPAL_opacity_table**(*path: str*) → numpy.ndarray

Parse the 126 tables out of a properly formated opacity table which dsep can understand. This idetifies all lines starting with Table # after the summary section and uses those to index where the tables begin. Given that dsep opacity tables are not square and that numpy can only handel rectangular data all rows are padded to the length of the longest row with np.nan. Therefore, nan(s) should be interprited as locations where the opacity table was undefined.

Parameters **path** (*str*) – path to opacity table

Returns **p** – array of shape (126, 70, 19) where the first axis is the composition axis for the 126 compositions which dsep expects, the second is the temperature axis, and the third is the R axis.

Return type np.ndarray

6.1.4.4 pyTOPSScrape.parse.tops module

Author: Thomas M. Boudreaux

Created: September 2021

Last Modified: September 2022

Functions for parsing raw output from TOPS webform

Examples

Say you have some raw output from the TOPS webform saved to a file called "rawOutputTest.dat", then you may parse that with

```
>>> rho, LogT, RMO = load_tops("rawOutputTest.dat")
```

pyTOPSScrape.parse.tops.**extract_composition_path**(*path: str*) → Tuple[float, float, float]

Given the name of a TOPS return file (named in the format OP:n_X_Y_Z.dat) extract X, Y, and Z

Parameters **path** (*string*) – path to TOPS return file

Returns

- **X** (*float*) – Hydrogen mass fraction
- **Y** (*float*) – Helium mass fraction
- **Z** (*Metal mass fraction*)

pyTOPSScrape.parse.tops.**load_tops**(*TOPSTable: str*, *n: int = 100*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Given the path to a file queried from the TOPS webform put it into a computer usable form of 3 arrays. One array of mass density, one of LogT and one of log Rossland Mean Opacity

Parameters

- **TOPSTable** (*string*) – Path to file queried from TOPS webform
- **n** (*int, default=100*) – The size of density grid used in TOPS query form.

Returns

- **rho** (*np.ndarray(shape=n)*) – Array of mass densities (in cgs) parsed from TOPS table.

- **LogT** (*np.ndarray(shape=m)*) – Array of temperatures (in Kelvin) parsed from TOPS table.
- **OPALTableInit** (*np.ndarray(shape=(m,n))*) – Array of Rossland Mean Opacities parsed from TOPS table.

6.1.4.5 Module contents

6.1.5 pyTOPSScrape.scripts package

6.1.5.1 Submodules

6.1.5.2 pyTOPSScrape.scripts.main module

pyTOPSScrape.scripts.main.**full_run**(*kwargs: dict*)

Main function to call all the right functions in order to get the TOPS formated ATOMIC opacity tables downloaded and converted to the proper format for dsep to understand.

Parameters **kwargs** (*dict*) – Dictionary containing all of the configuration information requiried for this function to run. This must include

- **abunTable (*str*)** Path of checmical abundance table to use for composition. Format of this table is defined in the ext module documentation.
- **outputDirectory (*str*)** Directory to save results of numFrac executable to.
- **nofetch (*bool*)** If no fetch is true then the TOPS webform will not be called. In this case all of the raw TOPS formated opacity tables must have already been downloaded.
- **hardforce (*bool*)** Delete all existing directories that this function would need. Use with caution. This has its own flag in an attempt to mitigate server load on TOPS. Essentially the program really tries to force you to not query TOPS if it detects that the files have already been downloaded.
- **force (*bool*)** If true will allow program to refetch if not all the TOPS files were downloaded (if the directory is not complete). Will not overwride the TOPS directory if all of the files were downloaded.
- **noopal (*bool*)** Flag whether or not to run the conversion of the TOPS tables to OPAL/DSEP formated table. If opal is set True this will happen, otherwise no conversion will take place. If you set opal to falce and nofetch to true the program will not do anything.
- **output (*str*)** Path to save DSEP/OPAL formated opacity table to.
- **jobs (*int*)** Number of threads to run TOPS query with

Notes

If there is an issue on the server end with TOPS this program will attempt to retry 10 times per composition by default (defined with the nAttepmts variable).

6.1.5.3 Module contents

6.2 Module contents

6.2.1 pyTOPSScrape

A package which aims to make the programmatic retrieval and use of high temperature radiative opacity tables from LANL somewhat simple

pyTOPSScrape provides both a command line and a python interface. The command line interface runs through the generateTOPStables script (which will have been installed to your path when you installed this package)

The python interface can mimic the full command line interface (including error checking and rate limiting) using the full_run function. If however, you wish to dig down to a more granular level the api module includes both query and convert sub modules which may be composed as needed.

6.2.2 Installation

6.2.2.1 PyPi

```
>>> pip install pyTOPSScrape
```

6.2.2.2 GitHub

```
>>> git clone https://github.com/tboudreaux/pytopsscrape.git
>>> cd pytopsscrape
>>> python setup.py install
```

6.2.2.3 Command Line Usage Example

```
>>> generateTOPStables GS98.abun rescalings.dat -d ./rawOutput -o GS98.opac -j 20
```

6.2.3 Input File Formats

pyTOPSScrape requires two input files to run. One (the first positional argument and hereafter the 'composition file') describes the base composition. The second positional argument (hereafter the 'map file') describes the set of classical compositions which will be queried from the TOPS web form. Each of these compositions will be a rescaling of the base composition (therefore the metal mass fractions wrt. Z will be maintained)

6.2.3.1 Composition file

The composition file should be in the following form

```
#STD [Fe/H] [alpha/Fe] [C/Fe] [N/Fe] [O/Fe] [r/Fe] [s/Fe] C/O X Y,Z
F -1.13 0.32 -0.43 -0.28 0.31 -1.13 -1.13 0.10 0.7584 0.2400,1.599E-03
#H He Li Be B C N O F Ne
12.00 10.898 -0.08 0.25 1.57 6.87 6.42 7.87 3.43 7.12
#Na Mg Al Si P S Cl Ar K Ca
5.11 6.86 5.21 6.65 4.28 6.31 -1.13 5.59 3.90 5.21
#Sc Ti V Cr Mn Fe Co Ni Cu Zn
2.02 3.82 2.80 4.51 4.30 6.37 3.86 5.09 3.06 2.30
#Ga Ge As Se Br Kr Rb Sr Y Zr
0.78 1.39 0.04 1.08 0.28 0.99 0.26 0.61 1.08 1.45
#Nb Mo Tc Ru Rh Pd Ag Cd In Sn
-0.80 -0.38 -99.00 -0.51 -1.35 -0.69 -1.32 -0.55 -1.46 -0.22
#Sb Te I Xe Cs Ba La Ce Pr Nd
-1.25 -0.08 -0.71 -0.02 -1.18 1.05 -0.03 0.45 -1.54 0.29
#Pm Sm Eu Gd Tb Dy Ho Er Tm Yb
-99.00 -1.30 -0.61 -1.19 -1.96 -1.16 -1.78 -1.34 -2.16 -1.42
#Lu Hf Ta W Re Os Ir Pt Au Hg
-2.16 -1.41 -2.38 -1.41 -2.00 -0.86 -0.88 -0.64 -1.34 -1.09
#Tl Pb Bi Po At Rn Fr Ra Ac Th
-1.36 -0.51 -1.61 -99.00 -99.00 -99.00 -99.00 -99.00 -99.00 -2.20
#Pa U
-99.00 -2.80
```

6.2.3.2 Map file

The map file should be in the following form

```
0.75,0.24,0.01
0.75,0.23,0.02
```

Where each row is X,Y,Z. The number of rows in this file will correspond to the number of queries issued against the TOPS web form

6.2.4 Testing

pyTOPSScrape ships with a number of tests which should be run to make sure that it installed correctly on your system. Additionally, as it is reliant on an external server it is a certainty that one day it will break. To run the tests a script has been included. From the pyTOPSScrape root directory

```
>>> cd tests
>>> ./runTests.sh
```

6.2.5 Etiquette & Cacheing

pyTOPSScrape makes use web servers hosted at Los Alamos National Labs (LANL). Before releasing this software I spoke with the T-1 group at LANL and received their assent. However, try to limit requests made against their web servers as much as possible. Obviously, if you are querying a few hundred tables because your stellar evolution code needs a few hundred opacity tables there is little to be done; however, do try and make sure that you have sorted out any and all bugs or typos in your input files before you query so that you won't have to go back and query multiple times. We want to be respectful of of the generosity of LANL here!

Additionally, pyTOPSScrape caches the raw query results to whatever directory is specified by the -d or –outputDirectory flag. This is so that if you want to implement your own converted you can do so without constantly re running the query functions. These results are cached in whatever directory is set in the –outputDirectory (or -d) command line option. To call the command line interface with cache usage enabled use the –nofetch flag. If you want to fetch tables and don't want to run the conversion set use the –noopal flag.

As an example, if you have already queried the TOPS web form using the command

```
>>> generateTOPStables GS98.abun rescalings.dat -d ./rawOutput -o GS98.opac -j 20 --
↪noopal
```

this will save all the raw output to the directory ./rawOutput (if you run the first example from this docs page this will also cache the results). You can then convert these to DSEP's OPAL format using the command

```
>>> generateTOPStables GS98.abun rescalings.dat -d ./rawOutput -o GS98.opac -j 20 --
↪nofetch
```

Notes

Website [1]_

Paper Describing Opacity Tables [2]_

[1] https://aphysics2.lanl.gov/apps/

[2] Colgan, James, et al. "A new generation of Los Alamos opacity tables." The Astrophysical Journal 817.2 (2016): 116.

PYTHON MODULE INDEX

p

INDEX

Q

R

S

T

V