T. E. A.

THERMOCHEMICAL EQUILIBRIUM ABUNDANCES

User Guide

Authors:
M. OLIVER BOWMAN
JASMINA BLECIC

Programmers:
M. OLIVER BOWMAN
JASMINA BLECIC

Lead Scientist:
JASMINA BLECIC

Principal Investigator:
JOSEPH HARRINGTON

Tester:
MADDISON STEMM

March 23, 2015

This document goes along with the TEA code, the TEA theory paper (Blecic et al., 2015), and the code description document. The documents are not peer-reviewed and the code is a test version only. They may not be redistributed to any third party. Please refer such requests to us. If you find this package useful for your research, please cite Blecic et al. (2015). The document and the program are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Our intent is to release this software under an open-source, reproducible-research license, once the code is mature and the first research paper describing the code has been accepted for publication in a peer-reviewed journal. We are committed to development in the open, and have posted this code on github.com so that others can test it and give us feedback. However, until its first publication and first stable release, we do not permit others to redistribute the code in either original or modified form, nor to publish work based in whole or in part on the output of this code. By downloading, running, or modifying this code, you agree to these conditions. We do encourage sharing any modifications with us and discussing them openly.

We welcome your feedback, but do not guarantee support. Please send feedback or inquiries to both:
Jasmina Blecic: jasmina@physics.ucf.edu
Joseph Harrington: jh@physics.ucf.edu

or alternatively,

Jasmina Blecic and Joseph Harrington
UCF PSB 441
4111 Libra Drive
Orlando, FL 32816-2385
USA

Thank you for testing TEA!

Contents

1 Introduction

There are two approaches to how one can calculate thermochemical equilibrium: by using equilibrium constants and reaction rates or by minimizing the free energy of the system. Although chemical equilibrium can be calculated almost trivially for several reactions present in the system, as the number of reactions increases, the number of equilibrium-constant relations becomes difficult to solve simultaneously. An advantage of the free-energy-minimization method is that each species present in the system can be treated independently, without specifying complicated sets of reactions. Therefore, just a limited set of equations needs to be solved. The Thermochemical Equilibrium Abundances (TEA) code is based on the Gibbs-free-energy minimization calculation originally developed by White et al. (1958) and Eriksson (1971). The code is written entirely in Python and is available to the scientific community under an open-source license.

TEA solves complex chemical systems with a plethora of species in a short amount of time (Section 10). The code is designed to either handle a system containing a single temperature and pressure (T, P) or a layered system of temperatures and pressures, such as one would find in a planetary atmosphere. The intended use of the TEA code is to solve equilibrium abundances for hot-Jupiter exoplanetary atmospheres, though it is not strictly limited to such a system. TEA uses the Lagrangian steepest-descent method (Blecic et al., 2015, Section 1.2) in order to minimize the free-energy of a system. This minimization followed by a short correction for validity called lambda correction (Blecic et al., 2015, Section 1.3) results in the equilibrium abundances one would expect to find at a certain temperature and pressure. The TEA output are abundances given as fractional abundances (mole mixing fractions), i.e., the ratio of each species' mole numbers to the total sum of all mole numbers in the mixture.

This is a second part of a three-part document describing the TEA code. The first part, the TEA Theory document (Blecic et al., 2015) presents the theoretical basis for the method applied, this document is the user guide, and the third is the TEA Code Description document by Blecic & Bowman. If you find this package useful, please cite Blecic et al. (2015).

This project was completed with the support of the NASA Earth and Space Science Fellowship Program, grant NNX12AL83H, held by Jasmina Blecic, PI Joseph Harrington. Project developers included graduate student Jasmina Blecic and undergraduate M. Oliver Bowman.

2 Dependencies

As TEA is written in the Python programming language, there are certain pre-requisite packages that must be installed for it to execute properly. Specifically, three major Python libraries are required to operate TEA: the general Python overhead, NumPy, a package to allow for scientific computing, and SymPy, a numerical computation package to facilitate solving a system of equations. Any other packages used by TEA are included by default in the Python overhead installation and do not need to be explicitly installed.

All of these dependencies have been tested across the popular operating systems (Linux, Mac OS X, and Microsoft Windows).

2.1 Python

Python is a high-level programming language widely used throughout scientific and engineering fields. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C or Fortran.

As Python is entirely open source and is often used for scientific calculations, it was chosen as the primary language for TEA.

TEA was written using Python 2.7.3. We tested TEA with other Python versions: 2.7.5 and 2.7.8 (current at the writing of this guide) and observed no performance difference between the versions.

2.2 NumPy

NumPy is the fundamental package for scientific computing with Python. NumPy is implemented in TEA primarily for its data-handling systems, i.e., object arrays and in/out functionality. It is also used in numerical and logarithmic calculations throughout the TEA pipeline.

TEA was written using NumPy 1.6.1. We tested TEA with other NumPy versions: 1.7.1, 1.6.1, 1.7.0, 1.8.0 and 1.9.0 (current at the writing of this guide). There is no known performance difference between the versions.

2.3 SymPy

SymPy is an open-source Python package developed to give Python a more fluent and intuitive means of solving individual or sets of equations. TEA uses SymPy to solve sets of equations during the Lagrange optimization process (Blecic et al., 2015, Section 1.2.1 Equation 26). SymPy documentation and installation procedure can be found at `http://sympy.org/`.

TEA uses SymPy 0.7.1.rc1 for optimal run-time. Newer versions have been shown as accurate, but can heavily consume resources and inflate computation time.

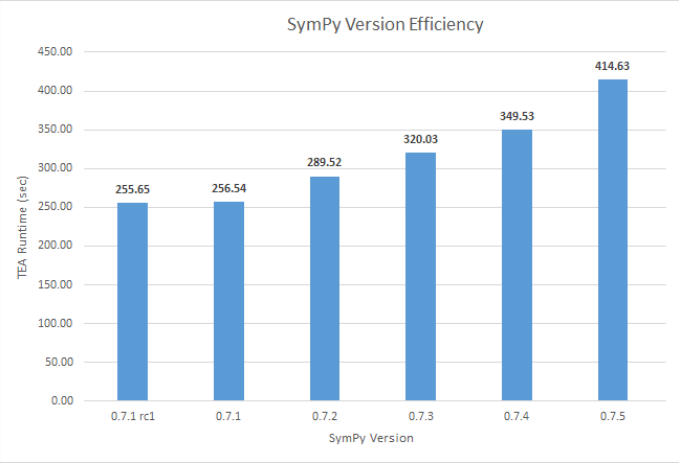

Figure 1: Performance benchmarks for a sample TEA execution with recent SymPy versions. Hardware specs: Intel i7-3770K 3.50GHz [CPU].

We performed a test comparing SymPy version 0.7.1.rc1 with more recent versions (Figure 1). The test was performed on a machine with an Intel i7-3770K 3.50GHz processor and an input file that contains 100 temperature and pressure points, 5 input species, and 13 output species. The test shows the speed of execution between different SymPy versions using the flag `rational = False` in the `solve()` SymPy routine. This flag is introduced to improve accuracy, however, we see no accuracy improvements in the final abundances by setting `rational = True`.

When `rational = False`, the total computation time decreases in general with respect to the runs where `rational = True`. However, this varies in each SymPy version. The continuous increase in the total run-time of 25 – 50 seconds is observed for every new version. On the other hand, if set to `True`, the total run-time jumps to about 6000 seconds (the `solve()` routine is used at every T, P point and in each iteration cycle). Thus, our `solve()` routine uses the flag `rational = False`, and although we recommend installing the fastest 0.7.1.rc1 version, any newer SymPy version can be used with a slight overall run-time increase.

3 Work-flow & Modular Format

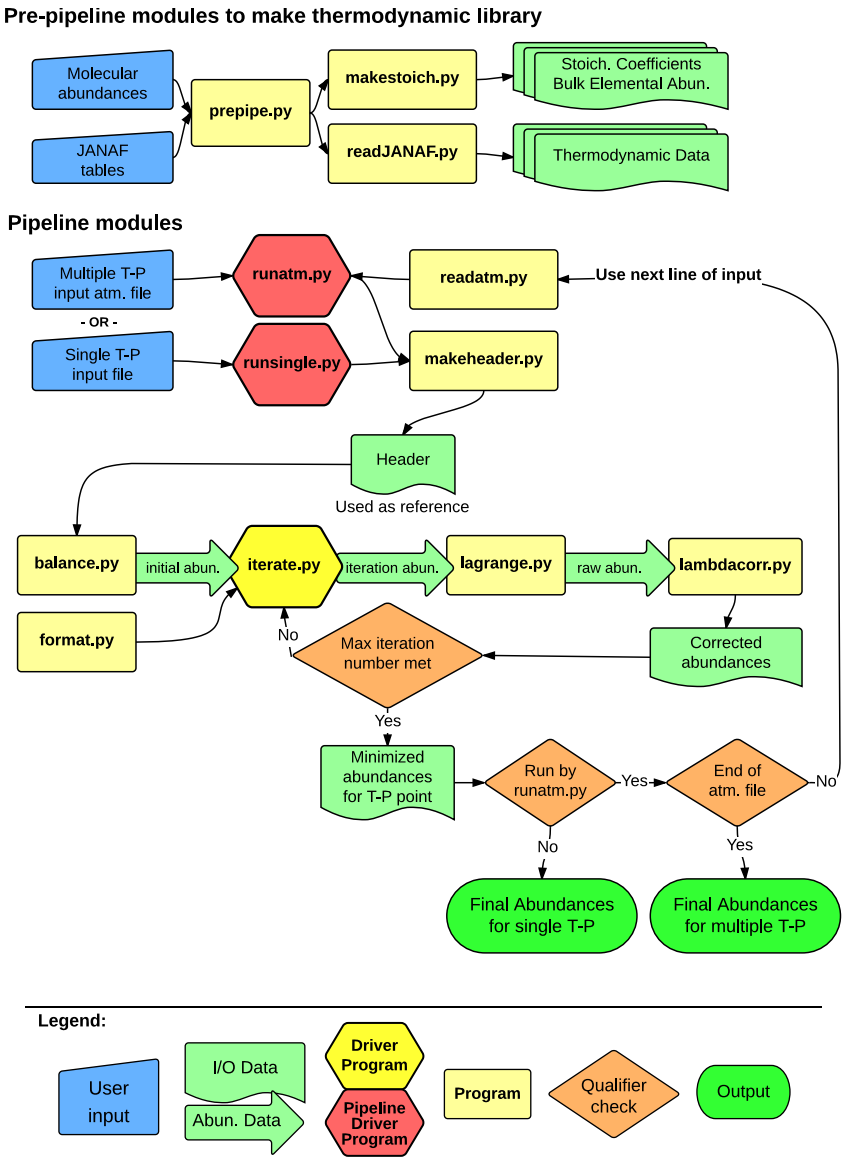

Figure 2: Layout of the pre-pipeline and pipeline packages.

The TEA code is written in a modular format (Figure 2) where each scientific package has one specific goal, from creating thermodynamic libraries to producing minimized values. These packages are supported by several file control packages that do not perform calculations on their

own, but instead provide input-output (I/O) support for reading and writing files and data. All packages have one of three roles: scientific calculation, file or data structure support, or execution of the calculation programs over temperature and pressure points in an iterative manner.

The code itself is split into two main sections: a collection of single-run setup programs (pre-pipeline) and the main iterative loop (pipeline).

3.1 Pre-Pipeline

The pre-pipeline is made to lay out the thermodynamic libraries and stoichiometric information TEA will use in the main pipeline. It uses the JANAF tables and elemental abundances file (Sections 6.4 and 6.5 respectively) as inputs. The pre-pipeline is executed using the wrapper program `prepipe.py` which in turn executes both `readJANAF.py` and `makestoich.py` (see respective sections in the TEA Code Description by Blecic & Bowman). The pre-pipeline is executed prior to the release, and the thermochemical libraries and stoichiometric information are distributed with the code. Thus, the user does not need to execute the code. However, if updated JANAF tables are provided in the same format as currently used in TEA (last downloaded October 2012), the pre-pipeline execution will populate the thermochemical libraries and stoichiometric values with new information.

3.2 Main Pipeline

Similarly to the pre-pipeline, the main TEA pipeline is executed with one of two driver programs depending on the nature of the system the user has specified: a system with constant temperature and pressure (single T, P) or a full atmospheric systems (multiple T, P). Both drivers follow the same general flow of execution and differ in the number of unique T, P points solved and the file structure of the input, intermediary, and output files. For each T, P point, the pipeline establishes an initial set of mass-balanced abundances and proceeds to iteratively apply Lagrange minimization and lambda correction before settling on a final set of equilibrium abundances (Blecic et al., 2015). Should the user need to solve system of more than one T, P point, the pipeline driver will additionally repeat this process over each T, P point. The final abundances are given as fractional abundances, mole mixing fractions.

4 Installing TEA

TEA is available to the scientific community via the open-source development site GitHub.com. To download TEA from the github repository, execute in terminal:
`git clone https://github.com/dzesmin/TEA`.

FINDME: add here tar file and packing instructions
The following directories and files are included in the download:

1. **readme** - a file with basic instructions
2. **doc** - directory containing:
   (a) `examples` - directory with following examples:
      i. `singleTP` - directory with single T, P example run
      ii. `multiTP` - directory with multiple T, P example run

iii. `quick_example` - directory with a quick example run
iv. `PT` - directory containing several examples of the pressure and temperature profile

(b) `TEA Theory document, Blecic et al. (2015)` - document containing theory part of the code
(c) `TEA User Guide` - document containing the user guide
(d) `TEA Code Description` - document containing program description
(e) `install_guide.txt` - file containing instruction how to install, set, and execute TEA

3. **janaf** - directory containing JANAF tables in their raw format (downloaded October 2012)
4. **lib** - directory containing abundances file, thermochemical data, and stoichiometric information:

(a) `abundances.txt` - elemental abundances file, based on the elemental solar abundances given by Asplund et al. (2009), Table 1.
(b) `TEA.cfg` - TEA configuration file
(c) `gdata` - directory containing thermochemical data of interest from JANAF tables
(d) `stoich.txt` - stoichiometric data file
(e) `conversion_record_sorted.txt` - record of converted JANAF files produces after readJANAF.py is executed. The run will produces unsorted conversion_record.txt file. To sort the content alphabetically, execute in terminal:
`sort conversion_record.txt >conversion_record_sorted.txt`

5. **prepipe** - directory containing source files to produce thermochemical library and stoichiometric information: directory `gdata/`, `conversion_record.txt`, and `stoich.txt` (see respective sections in the TEA Code Description by Blecic & Bowman):

(a) `prepipe.py` (*) - wrapper for `makestoich.py` and `readJANAF.py`
(b) `makestoich.py` (*) - makes `stoch.txt` file with stoichiometric infomation
(c) `readJANAF.py` (*) - makes the `gdata/` directory with converted JANAF tables and the `conversion_record.txt` file

Asterisk (*) indicates modules that must be executable in *nix (e.g., Linux) systems.

6. **tea** - directory containing tea source files (see respective sections in the TEA Code Description by Blecic & Bowman):

(a) `balance.py` (*) - solves mass balance equation
(b) `format.py` - auxiliary program to manages I/O operations
(c) `iterate.py` (*) - executes iteration loop
(d) `lagrange.py` - applies Lagrange method
(e) `lambdacorr.py` - applies lambda correction method
(f) `makeheader.py` - writes header files
(g) `readatm.py` - reads multiple T, P file
(h) `runatm.py` (*) - runs TEA for multiple T, P case
(i) `runsingle.py` (*) - runs TEA for single T, P case

(j) `makeatm.py` (*) - makes multiple T, P file
(k) `plotTEA.py` (*) - plots TEA output
(l) `readconfig.py` - reads configuration file

Asterisk (*) indicates modules that must be executable in *nix (e.g., Linux) systems.

The modules that perform scientific calculations are: `balance.py, lagrange.py`, and `lambdacorr.py`. The modules that perform file or data structure support are: `prepipe.py, makestoich,py, readJANAF.py, format.py, makeheader.py, readatm.py, plotTEA.py`, and `readconfig.py`; while the modules that perform execution of the calculation programs are: `iterate.py, makeatm.py, runsingle.py` and `runatm.py`.

To properly install and run TEA, a list of steps is provided in the `install_guide.txt` placed in the `doc` directory. The install guide describes the general purpose of the TEA code, lists the Python packages needed for proper TEA execution, and gives instructions how to configure TEA input, execute TEA, and plot its output. Command lines for executing example runs (given in `../TEA/doc/examples/` directory) are provided with the list of potential user errors.

5 Quick Example

The following script gives an example of the TEA run on Linux operating systems. Copy and paste the commands into the command prompt.

Create a working directory:
mkdir TEA_example
cd TEA_example

Clone the repository to the working directory:
git clone https://github.com/dzesmin/TEA TEA
cd TEA

Make run directory to place the configuration file and outputs:
mkdir run
cd run
cp ../doc/examples/quick_example/TEA.cfg TEA.cfg

Run TEA using the pre-atmopsheric file from the doc/examples folder:
../tea/runatm.py ../doc/examples/quick_example/quick_example.atm run_example

To plot results do:
../tea/plotTEA.py run_example/results/run_example.tea H2,H2O,CO

Check the output by comparing the results between doc/examples/quick_example/results/ and run/run_example/res
folders and doc/examples/plots/ and run/plots/ folders.

6 Program Inputs

TEA had two modes of execution: single and multiple T, P. Both modes require the pre-pipeline to be executed so the thermochemical library data (`gdata/`) and stoichiometric values (`stoich.txt`) are properly populated. Inputs for the pre-pipeline are the JANAF tables and elemental abundances profile provided with the TEA package.

TEA is configured with an ASCII file, TEA.cfg. Both single and multiple T, P runs, require an ASCII file carrying elemental abundances in logarithmic (dex, decimal exponent) units (further called abundances file). For single T, P runs, TEA requires a custom made input file with a single temperature and pressure point. For multiple T, P runs, TEA requires a file carrying a list of temperature and pressure points, a multiple T, P file, further called a 'pre-atmosphere file'. This file can be custom made or created by running the TEA supporting program `makeatm.py`. If `makeatm.py` module is used, a list of pressure and temperature points must be stored in an ASCII file, called presure-temperature profile (PT-profile) file (more on each of the input files in the following sections).

While the JANAF tables (located in `../TEA/janaf/`), thermochemical data (located in `../TEA/lib/gdata/`), stoichiometric information (located in `../TEA/lib/stoich.txt`), and abundances (located in `../TEA/lib/abundances.txt`) are provided with the package distribution, users may prepare their own versions of these files granted the format is correct and acceptable to TEA's file reader. A custom-made single T, P input, multiple T, P input, and a PT-profile file must also be in the TEA acceptable format (see Sections 6.2 and 6.3).

6.1 TEA.cfg

The TEA package uses a single configuration file. This file is named `TEA.cfg` and is located in the `../TEA/lib/` directory. Before TEA is executed, the user needs to open a working directory outside of the main TEA package, copy the TEA.cfg to the working directory, and edit it with the correct information.

The configuration file contains two sections: `TEA SECTION` that manages run-time restrictions that are not specific to any physical system, and `PRE-ATM SECTION` that manages configuration of the `makeatm.py` module and how a multiple T, P, also called a pre-atmosphere file is produced. `TEA SECTION` has eight parameters the user can change to control how TEA should run and produce files: the number of iterations to run per T, P point, whether or not to save header files, whether or not to save intermediate files, the option to show additional debugging info on-screen, the option to show the time required to run each step of the pipeline, the path to the main TEA package, the path to the abundances file, and the path to the current working directory. `PRE-ATM SECTION` has four parameters specific to the chemical system of interest: the path to the pressure and temperature file, the desired name of the pre-atmosphere file, the list of elemental species (must have names as they appear in the periodic table) and the list of output species (must include all elemental and desired molecular species named as they appear in `gdata/` folder or `conversion_record_sorted.txt`). Below is the default appearance of the configuration file:

```
# ============================================================
# Configuration file containing two sections:
# 1.  TEA section with parameters and booleans to run and debug TEA.
# 2.  PRE-ATM section with parameters to make pre-atmosphere file.
```

```
# ==================================================================

# ========================== TEA SECTION ===========================
# Change the parameters below to control how TEA runs.  The default
# number of iterations in 'maxiter' parameter is the optimal
# value for common molecular species in hot Jupiters.
# Following 'maxiter' parameter, next four parameters are for
# debugging purposes only.  Setting them to 'False' will ensure the
# fastest execution.
# ==================================================================

# ========    Sets maximum number of iteration   ========
maxiter      = 100    # (Def: 100)   Number of iterations

# ========         Controls output files         ========
save_headers = False  # (Def: False) Preserve headers
save_outputs = False  # (Def: False) Preserve intermediate outputs

# ========    Controls debugging and tracking    ========
doprint      = False  # (Def: False) Enable various debug printouts
times        = False  # (Def: False) Enable time printing

# ========       Location of TEA package        ========
location_TEA = ../TEA/

# ========      Location of abundances file     ========
abun_file = ../TEA/tea/lib/abundances.txt

# ========     Location of working directory    ========
location_out = .

# ======================== PRE-ATM SECTION =========================
# Execution of this section is optional.  The user can produce a TEA
# pre-atmosphere file by running makeatm.py, or make a custom-made
# file in the format that TEA can read it.  See the correct format
# in the ../TEA/doc/examples/multiTP/ folder.

# Change the parameters below to control how pre-atmosphere file is
# made.  Before executing the module make a pressure-temperature
# file.  Run makeatm.py as:  makeatm.py <RESULTS_DIR_NAME>

# ========     Pressure and temperature file    ========
PT_file = ../TEA/tea/doc/examples/PT.dat

# ========       pre-atmosphere filename       ========
# Use extension .atm.  File will be placed in atm_inputs/.
pre_atm_name = pre_atm.atm
```

```
# ========          Input elements names          ========
# MUST have names as they appear in periodic table.
input_elem = H C N O

# ========          Output species names          ========
# MUST have names as they appear in gdata/ folder.
# MUST include all elemental species.
output_species = H_g C_g N_g O_g H2_ref CO_g CH4_g H2O_g N2_ref NH3_g
```

By default, TEA is set to run with 100 iterations and will not produce any additional files beyond the inputs and results files described in Sections 7.4 and 7.5. The convergence of any system's solution is unique to that system, though through extensive testing we have found that 100 iterations is often more than enough for a system's solution to successfully converge. Increasing the number of iterations TEA should run beyond 100 will significantly increase over-all run-time but may allow more complex systems (such as those with a plethora of output species) to converge to a more accurate solution. This default is the optimal value for common molecular species in hot-Jupiter atmospheres. When the maximum iteration is reached, TEA stops further abundance calculations for that T, P point and places the resulting values in the relevant results file.

For additional debugging, both the `doprint` and `times` booleans should be set to `True`. This will allow calculations and data passed between each step to be printed in on-screen while TEA is run, as well as to show the amount of time each package took to execute.

6.2 Single T, P

The single T, P calculation for TEA requires an input file consisting of three pieces of data: temperature (K), pressure (bar), and list of chemical species allowed to exist in the system. The list of chemical species must conform to the naming convention produced by `readJANAF.py` (placed in the `gdata/` folder and `conversion_record_sorted.txt` file), and also must include all mono-atomic species in the system. This convention consists of the chemical formula of the species, the state (according to the JANAF naming structure, Section 6.4) and the isomer name, if applicable, all separated by underscores (e.g., H_g, N2_ref, CH4_g, etc).

For ease of organization, such input files are named "`singleTP_<DESCRIPTION>`," though this is not strictly required. The name of this file is passed to the TEA main pipeline via `runsingle.py`'s command-line argument, described in Section 8.4. Note that for each run of TEA using a single T, P point, the user must produce their own input file in the format described below. The following is an example of the single T, P input file (provided in the `../TEA/doc/examples/singleTP/inputs/inp_Example.txt`) with the corresponding key:

```
3500            # Temperature in Kelvin
51.0344729      # Pressure in bar
H_g             # List of species allowed in the system
H2_ref          #  Names for these must match those
H2O_g           #  produced by readJANAF.py
N_g             #
NH_g            #
```

```
NO_g            #
N2_ref          #
O_g             #
OH_g            #
O2_ref          #
```

6.3 Multiple T, P

The multiple T, P calculation for TEA is designed for use in a planetary atmosphere and, accordingly, requires a file containing lists of pressure (bar), temperature (K), elements present in the system (with thier elemental abundances), and a list of chemical species allowed to exist in the system. As with the single T, P file, the list of chemical species must conform to the naming convention produced by `readJANAF.py`, and also must include all mono-atomic species in the system. Such multiple T, P files will be henceforth referred to as pre-atmosphere files in contrast to the final pipeline output, that we will call `atmosphere files`.

For ease of organization, such input files are named "`multiTP_<DESCRIPTION>`," though this is not strictly required. The name of this file is passed to the TEA pipeline via `runatm.py`'s command-line argument, described in Section 8.5. Below is an example of a `pre-atmosphere input file`, (`../TEA/doc/examples/multiTP/inputs/multiTP_Example.atm`), with the corresponding key:

```
# This is a TEA pre-atmosphere input file.
# TEA accepts a file in this format to produce species abundances as
# a function of pressure and temperature.
# Output species must be added in the line immediately following the
# #SPECIES marker and must be named to match JANAF names.
# Units:  pressure (bar), temperature (K), abundance (unitless).

#SPECIES
H_g C_g N_g O_g H2_ref CO_g CH4_g H2O_g N2_ref NH3_g

#TEADATA
 #Pressure   Temp     H               C               ...
 1.0000e-05  100.00   9.9917414e-01   2.6893119e-04   ...
 1.1768e-05  129.29   9.9917414e-01   2.6893119e-04   ...
 1.3849e-05  158.59   9.9917414e-01   2.6893119e-04   ...
 ...         ...      ...             ...             ...
 7.2208e+01  2941.41  9.9917414e-01   2.6893119e-04   ...
 8.4975e+01  2970.71  9.9917414e-01   2.6893119e-04   ...
 1.0000e+02  3000.00  9.9917414e-01   2.6893119e-04   ...
```

Additional data may be added to the multiple T, P - pre-atmosphere file if the user sees fit (specifically, additional content may be placed above the #SPECIES marker). TEA will intelligently seek out only the information relevant to its calculations. As the pre-atmosphere file must first give the abundances of elements present in the system, use of any dex abundances must first be converted to mole fractions as explained in Section 6.5.

Such a pre-atmosphere file can be custom made, granted the above format is obeyed. Quick production of a `pre-atmosphere file` can be accomplished using a supporting module

`makeatm.py`. This module is configured with the second section of the TEA.cfg file, `PRE-ATM SECTION`. The module requires a pressure-temperature (PT) profile file, carrying the list of T, P points in the following format:

```
# P (bar) T (K)
1.0000e-05 100.00
1.1768e-05 129.29
1.3849e-05 158.59
1.6298e-05 187.88
1.9179e-05 217.17
 ...         ...
 ...         ...
```

Several examples of the PT profile files are given in the examples folder, `../TEA/doc/examples/PT/`. The path to the PT-profile file need to be provided in TEA.cfg. Other parameters needed to ensure that `makeatm.py` is properly executed are: name of the pre-atmosphere file, list of elemental species, and the list of output species. The format of elemental and molecular species must be followed as described in Section 6.1. To run `makeatm.py` read Section 8.3.

6.4 JANAF Tables

The Joint Army Navy Air Force (JANAF) Thermochemical Tables are publicly distributed by the National Institute of Standards and Technology (NIST) and contain thermodynamical data for many chemical species. For the purposes of TEA, only three columns of the available data are used: temperature, free energy function, and heat of formation (Section 7.1).

The raw, ASCII versions of the JANAF tables available for distribution are not machine-readable by default due to comments within the files to mark transitions or changes in state. To account for this, the `readJANAF.py` module is made to remove these disjointed comments and allow for easy machine-reading of the remaining data. Only these comments are removed and data remains untouched from that found at the source.

The NIST-JANAF files are available for both on-line viewing and download from `http://kinetics.nist.gov/janaf/`. Along with the tables, provided are the documentations and naming conventions for each chemical species that has an appropriate NIST-JANAF file. The raw versions of these tables are included in every TEA installation in the `../TEA/janaf/` directory.

6.5 Abundances

The elemental abundances chosen as the baseline for the TEA pipeline are those provided by Asplund et al. (2009), Table 1. They are given in logarithmic, dex (decimal exponent) format and are meant to emulate conditions within the photosphere of the sun. In the code, dex abundances are converted into mole fraction abundances, where dex abundance of each element is converted to number density and then divided by the hydrogen number density. These abundances are used to ensure the mass balance is maintained in all TEA calculations.

Should the user choose to access or otherwise alter the abundances used for TEA, these values are housed in `abundances.txt`. Aside from the elemental solar abundances, this file carries additional information about each atomic species (beginning with deuterium, ending with uranium).

Columns listed are, in order: the element's atomic number, the element's atomic symbol, the element's dex abundance, the element's full name, and the element's atomic mass. User changes should be applied to the third column containing the dex abundances, while the other columns should remain unmodified.

7 Program Outputs

7.1 Pre-pipeline

The pre-pipeline produces two components that TEA will use in the main pipeline in any run: the thermodynamic libraries and the file carrying stoichiometric information. This information is produced from the JANAF tables and elemental abundance file passed as inputs to the pre-pipeline. The primary function of the pre-pipeline outputs is to establish the stoichiometric and thermodynamic rules TEA must obey in order to maintain a system at equilibrium (Blecic et al., 2015).

The pre-pipeline is divided into two components, each of which are responsible for one of the outputs that are produced: `makestoich.py` that produces the stoichiometric information and `readJANAF.py` that creates the thermodynamic library. More information on the programs themselves can be found in their respective sections in the TEA Code Description by Blecic & Bowman.

The stoichiometric file created by the pre-pipeline is named `stoich.txt` and is placed in `../TEA/lib/`. This file has three sections containing information derived from the input abundance profile and JANAF tables, namely the list of available elements in TEA, their respective dex abundances, and a list of each species used in the JANAF tables with corresponding stoichiometric counts of each element they contain. Below is an example to show the structure of the file if only a few JANAF files were used (note that `b` represents the dex elemental abundance from Section 6.5):

```
         b  0.  12.00  10.93  1.05  1.38  2.70  8.43  7.83  8.69  ...
   Species   D      H     He    Li    Be     B     C     N     O  ...
       CH4   0      4      0     0     0     0     1     0     0  ...
       H2O   0      2      0     0     0     0     0     0     1  ...
        N2   0      0      0     0     0     0     0     2     0  ...
```

The thermodynamic library created by running `readJANAF.py` is a representation of the TEA-required thermodynamic data from JANAF tables, made in a more machine-readable format. This library is placed in `../TEA/lib/gdata/` and contains three columns with temperature, free energy function ($-[G^o - H^o(T_r)]/T$) and heat of formation ($\Delta_f H^o$) values for each JANAF species. Note that each thermodynamic file is only produced if it contains values corresponding to 1 bar, calculations for other pressures are performed within the main pipeline. The name of each file in the library matches the name of the species in the JANAF table with an additional marker for state (e.g., `H2O_g`, `O2_ref`, `Mg(OH)2_g`, etc, where "ref" denotes standard state). For more information on naming conventions and notations, see the JANAF Volume 1 Intro found at `http://kinetics.nist.gov/janaf/janaf4pdf.html`.

7.2 Intermediate Files

To facilitate debugging or verification of scientific method, TEA can optionally produce output files after each step it takes towards finding the minimized Gibbs-free energy of a system. These

intermediate files are split into two parts: the raw output of the Lagrange method, and the corrected mole numbers resulting from lambda correction. Each iteration produces two of each file (one human-read, one machine-read), and in the case of the multiple T, P systems, each T, P point will produce a full set for each of its iterations.

To activate production of these files, the `save_outputs` variable in `TEA.cfg` must be set to `True`. These files will contain the data identical to that passed between the pipeline programs via memory, and is useful for checking the progression of a minimization. Two different files are produced at each iteration: one containing pre-lambda correction mole values (denoted with "-nocorr" in the file name) and another containing post-lambda correction values. In addition, each of these files is again split between a human-readable format (denoted with "-visual" in the file name) and a machine-readable format (denoted with "-machine-read"), both of which contain the same data. Note that these files are all produced for the user and do not influence TEA's execution. The format of these files is identical to that found in the results files explained in Section 7.4.

The files themselves will be located in the `outputs/` directory and are nested under a directory the user defines in the pipeline driver program command line, below the current working directory. If TEA is using multiple T, P points, the temperature and pressure of that run are also used in the directory name. For example, if the description passed to the pipeline driver program is `TEA_run` and is at 1e-3 bar and 1500K, the produced files will be in the `outputs/TEA_run_1500K_1e-02bar/` directory. Single T, P runs will produce a similar directory but with no information for temperature and pressure.

7.3 Auxiliary Header Files

For each T, P point TEA will create an auxiliary file, called `header`, containing relevant chemical information. Such a file is created to accomplish two main goals, the first of which is to coalesce the data from the various inputs (single T, P input file or multiple T, P `pre-atmosphere` input file, JANAF tables, elemental abundances, and stochiometric coefficients) into one easy-to-access file to reduce file read-time. This also accomplishes the second goal which will allow for easy debugging should the user need to confirm the inputs being read into TEA. Once a header file is produced, it is effectively the only source of information TEA needs to complete for the abundance calculations at each T, P point. Below is an example header file produced by TEA:

```
# This is a header file for one T, P.
# Contains following data:
# pressure (bar), temperature (K), elemental abundances (b, unitless),
# species names, stoichiometric values of each element in the species
# (a), and chemical potentials.

51.0344729
3500.0
b 0.99944292339 6.7570634538e-05 0.000489505975044
```

```
# Species  H  N  O  Chemical potential
      H_g  1  0  0  -9.73713019872
   H2_ref  2  0  0  -21.4036696373
    H2O_g  2  0  1  -38.9040887478
      N_g  0  1  0  -5.68391721765
     NH_g  1  1  0  -14.622593565
     NO_g  0  1  1  -28.3762364925
   N2_ref  0  2  0  -28.9641106188
      O_g  0  0  1  -14.4687823951
     OH_g  1  0  1  -26.683677898
   O2_ref  0  0  2  -30.8998942938
```

Each header file is produced at the beginning of a T, P abundance calculation by `makeheader.py`. The file itself is created and controlled entirely by the main TEA pipeline and requires no user interactions or modifications to function. Preservation of these files is optional, however, and can be controlled using the `save_headers` boolean in the configuration file (Section 6.1). The files themselves will be located in `headers/` and are nested under a directory the user defines in the pipeline driver program command line, below the current working directory. If TEA is using multiple T, P points, the temperature and pressure of that run are used in the header's file name to differentiate between the points.

A header file contains the six components needed for the abundance calculations: pressure in bars, temperature in Kelvin, mole fraction of elemental abundances, output species, stoichiometric values per species, and chemical potential per species. Note that the order of the elemental abundances matches those in the stoichiometric array; more information on how this data is obtained and calculated from the various input sources can be found in the TEA Code Description by Blecic & Bowman.

7.4 Single T, P

Regardless of the production of intermediate output files, the execution of a single temperature and pressure run of TEA using `runsingle.py` will produce two results files containing the output abundance data for the system. As with intermediate outputs, each file is defined to be either human-readable or machine-readable, both of which contain the same information in differing formats. Specifically, the data these files contain are the following: location of the header file, number of iterations, species list, initial mole numbers, final mole numbers, change in mole numbers, final abundances, total initial moles, total final moles, and change in total moles. When used alongside the header file for the TEA run in question (Section 7.3), all initial and final conditions for the system can be easily extracted.

The two results files are placed in the `results/` directory and are nested under the directory named after the description passed to the driver program, below the current working directory. For example, if the description passed to `runsingle.py` is `TEA_run`, the produced files will be in the `./TEA_run/results/` directory. The files produced are named "results-machine-read.txt" and "results-visual.txt" for machine- and human-readability, respectively. Below is an example of a human-read results file:

```
   This .txt file is for visual use only.
These results are for the "C:./headers/header_Example.txt" run.
Iterations complete after 100 runs at 51.0344729 atm and 3500.0 K.
```

```
 Species |     Initial x |       Final x |          Delta |     Final Abun
     H_g |  0.0000100000 |  0.0266365865 |   0.0266265865 |  5.1914920e-02
  H2_ref |  0.4992669557 |  0.4859215667 |  -0.0133453890 |  9.4706501e-01
   H2O_g |  0.0004395060 |  0.0004740815 |   0.0000345755 |  9.2398867e-04
     N_g |  0.0000275706 |  0.0000000879 |  -0.0000274828 |  1.7127204e-07
    NH_g |  0.0000100000 |  0.0000001048 |  -0.0000098952 |  2.0421056e-07
    NO_g |  0.0000100000 |  0.0000000154 |  -0.0000099846 |  3.0046018e-08
  N2_ref |  0.0000100000 |  0.0000336813 |   0.0000236813 |  6.5645089e-05
     O_g |  0.0000100000 |  0.0000004731 |  -0.0000095269 |  9.2214693e-07
    OH_g |  0.0000100000 |  0.0000149356 |   0.0000049356 |  2.9109567e-05
  O2_ref |  0.0000100000 |  0.0000000002 |  -0.0000099998 |  3.0881100e-10

                  Initial Total Mol:  0.4998040323
                    Final Total Mol:  0.5130815330
                Change in Total Mol:  0.0132775007
```

Should the user desire to use these results within other programs, ”results-machine-read.txt” is designed to be easily read by any programming language. Data appears line-by-line in the order described above, however, final abundances are not contained in this file.

7.5 Multiple T, P

Similar to a single temperature and pressure execution of TEA, if `save_outputs` is set to `True` in the TEA.cfg, a multiple T, P run using `runatm.py` will produce the two files outlined above for each T, P point contained in the system. Should the pre-atmosphere file for a particular run contain 100 unique T, P points, for example, `runatm.py` will produce 100 sets of results files (one for each unique T, P point). For each unique T, P point in the system, a directory will be made describing the temperature and pressure of that point, inside which the two results files ”results-machine-read.txt” and ”results-visual.txt” are placed. In addition to these results files, the multiple T, P execution of TEA will also produce a complete ’atmosphere file’ for the system with extension `.tea`. The format of this file is similar to the pre-atmosphere file described in Section 6.3 with a different commentary section and and columns containing the final abundances (mixing fractions) instead of the elemental abundance columns the columns will contain the final abundances (mixing fractions) for each of the output species in the system. If `save_outputs` is set to `False` in the TEA.cfg, a multiple T, P run will produce only the final atmosphere file containing final mole-fraction abundances for each species in the system.

These results file are placed in the `results/` directory and are nested under the directory named after the description passed to the driver program, below the current working directory. For example, if the description passed to `runatm.py` is `TEA_run`, the produced files will be in the `./TEA_run/results/` directory).

8 Executing TEA

TEA has two main execution modes: single T, P and multiple T, P run. In addition to these, TEA uses several supporting programs that provide tools to create TEA inputs and to plot TEA output. The `../TEA/prepipe/` folder houses modules to populate thermochemical data and stoichiometric information. The `makeatm.py` module facilitate the production of a multiple

T, P pre-atmosphere file. The `plotTEA.py` module plots desired TEA output.

Described below is the main output structure after TEA execution and a summary of each execution: what are the input files, what are the commands to run the code, and what are the outputs.

8.1 Directory Structure

Before TEA can be executed, the user must create a main TEA working directory outside of the TEA main package directory. In this directory, all future TEA runs will be placed under the directories named by the user on each run (the argument passed to TEA upon execution, `<DIRECTORY_NAME>`, read below and see the directory layout).

The user, then, must copy the TEA.cfg file (located in `../TEA/lib/`) to this working directory. Once the TEA working directory is made and the TEA.cfg is placed inside, no additional copy of the TEA.cfg file is needed. For each run of TEA, the user should edit this configuration file with the desired values. Upon TEA execution, the most recent configuration file will be copied to the `inputs/` directory below the directory named by the user, documenting the settings of that run.

When `runsingle.py`, `runatm.py`, or `makeatm.py` is run, the user must provide the name of the directory that will receive the current run's inputs and outputs on the command line. The argument name is `<DIRECTORY_NAME>`. This directory will be located right below the TEA working directory. Inside this directory, the `inputs/` directory will be created, as well as the `atm_inputs/` directory should the user choose to create a pre-atmosphere input file using `makeatm.py`. These directories will contain all inputs used for each TEA run. For TEA driver programs (single and multiple T, P runs) those are: the single or multiple T, P input file, abundances file, and `TEA.cfg`. For `makeatm.py` executions, in addition to these files the `atm_inputs/` directory will also contain the PT-profile file.

Upon running `runatm.py` for that particular pre-atmosphere file, the user should chose the same directory name (`<DIRECTORY_NAME>`) as previously used for the `makeatm.py` run, so all inputs and outputs are placed in the same directory that already contains `atm_inputs/`. If `makeatm.py` is not executed and the pre-atmosphere file is either custom made or used from previous runs, the user should provide a new directory name. In this case the structure is the same, except that the `atm_inputs/` directory will be missing.

After TEA's execution, the results of the run (see Sections 7.4 and 7.5) are placed in the `results/` directory, nested under the directory named by the user on the command line (`<DIRECTORY_NAME>`). Depending on the variable assignments in TEA.cfg for `save_outputs` and `save_headers`, two additional directories may be created: `outputs/` to contain all intermediate outputs described in Section 7.2 and `headers/` to contain all header files described in Section 7.3.

Below are examples of a TEA working directory structure for two multiple T, P runs, with or without `makeatm.py` execution (left and right, respectively) and with intermediate outputs ignored or preserved (top and bottom, respectively). The directory names for these examples, mirroring the `<DIRECTORY_NAME>` argument for TEA execution, are `Jupiter100layers` and `WASP43b`:

```
----------------------- save_outputs = False --------------------------
TEA_work_dir/
         ./TEA.cfg       ./Jupiter100layers/             ./WASP43b/
```

```
                                        ./atm_inputs/           ./inputs/
                                        ./inputs/              ./results/
                                        ./results/

----------------------- save_outputs = True -------------------------
TEA_work_dir/
         ./TEA.cfg      ./Jupiter100layers/             ./WASP43b/
                                        ./atm_inputs/           ./inputs/
                                        ./inputs/              ./headers/
                                        ./headers/             ./outputs/
                                        ./outputs/             ./results/
                                        ./results/
```

8.2 Pre-pipeline

The pre-pipeline contains three modules to produce thermochemical libraries and stoichiometric information (`../TEA/lib/gdata/` and `../TEA/lib/stoich.txt`). Both of these data are provided fully populated with the TEA package, so they need not be produced upon TEA installation. However, should new or modified JANAF tables become available, the user can re-run the code to populate these files with the new information. Note that the format of these new JANAF tables must mirror that of the ones available in the `../TEA/janaf/` directory, else TEA's execution will fail.

The modules contained within the pre-pipeline itself are `prepipe.py`, `makestoich.py`, and `readJANAF.py`. The `prepipe.py` module contains the common setup procedures for both `makestoich.py` and `readJANAF.py` and will execute both of these modules together. As stand-alone modules, `makestoich.py` produces stoichiometric information that will be located in the `../TEA/lib/stoich.txt` file, while `readJANAF.py` populates the `../TEA/lib/gdata/` folder folder with the thermochemical libraries.

The modules executed from any location will be automatically produce the outputs in the `../TEA/lib/` directory. There are two ways the user can run the code:
1. Run `prepipe.py` as: `../TEA/prepipe/prepipe.py`
2. Run `makestoich.py` and `readJANAF.py` separately as:
`../TEA/prepipe/makestoich.py`
`../TEA/prepipe/readJANAF.py`

8.3 makeatm.py

`makeatm.py` is a supporting module that allows the user to make a multiple T, P pre-atmosphere file in the format TEA can read. To execute `makeatm.py`, the user must make a TEA working directory outside of the TEA main package and to copy the TEA.cfg file (located in `../TEA/lib/`) in that directory (see Section 8.1). The `makeatm.py` module uses both TEA.cfg sections to run, thus they need to be edited with correct information (see Section 6.1). The input files used are the abundances file that carries elemental abundances and a PT-profile file with pressure and temperature information. The user can provide its own abundances file, granted the correct format given in the abundances file provided is obeyed. The basic solar elemental abundances file (described in Section 6.5) is placed in `../TEA/lib/abundances.txt`, while the PT-profile file needs to be created by the user in the format that `makeatm.py` can read. The

format of the PT-profile file is described in Section 6.3 and examples of some PT-profile files are given in `../TEA/doc/examples/PT/`.

The `makeatm.py` module is run with one argument:

```
../TEA/tea/makeatm.py <DIRECTORY_NAME>
```

The structure of the input-output directories is explained in the Section 8.1. An example of a `makeatm.py` execution is provided in the `../TEA/doc/examples/multiTP/atm_inputs/` folder.

8.4 Single T, P

The `runsingle.py` module executes using a single T, P input file containing only one temperature and pressure point. Again, before running the code, the user needs to edit the TEA.cfg file copied to the TEA working directory.

`runsingle.py` is executed with two arguments:

```
../TEA/tea/runsingle.py <SINGLETP_INPUT_FILE_PATH> <DIRECTORY_NAME>
```

`<SINGLETP_INPUT_FILE_PATH>` defines the path to the single T, P file that was made by the user. The format of the file is given in Section 6.2. `<DIRECTORY_NAME>`, as previously explained in Section 8.1, is the directory in which this particular run will be contained. An example with the full directory inputs/outputs structure is given in `../TEA/doc/examples/singleTP/`.

8.5 Multiple T, P

The `runatm.py` module executes using a multiple T, P pre-atmosphere input file containing a list of pressure and temperature points. Again, before running the code, the user needs to edit the TEA.cfg file copied in the TEA working directory.

`runatm.py` is executed with two arguments:

```
../TEA/tea/runatm.py <MULTITP_INPUT_FILE_PATH> <DIRECTORY_NAME>
```

`<MULTITP_INPUT_FILE_PATH>` defines the path to a multiple T, P file. This file, as previously explained in Section 6.3, can either be produced using `makeatm.py` or made manually by the user if the proper formatting rules are obeyed. `<DIRECTORY_NAME>` is the directory in which this particular run will be contained. The directory structure made upon running `runatm.py` is described in the Section 8.1. An example of the multiple T, P run with the full directory inputs/outputs structure is given in `../TEA/doc/examples/multiTP/`.

8.6 Plot TEA

For user to visualize and plot the results of TEA, we provide the plotting routine `plotTEA.py`. The input for this module is the final output of the multiple T, P run ('atmosphere file'), that contains all species final abundances (mixing fractions) for each layer in the atmosphere, i.e., each temperature and pressure pair (Section 6.3). The routine cannot plot the results of the single T, P run.

`plotTEA.py` module is executed with two arguments:

```
../TEA/tea/plotTEA.py <RESULT_ATM_FILE_PATH> <SPECIES>
```

The `<RESULT_ATM_FILE_PATH>` argument is the path to the final atmosphere file, while `<SPECIES>` contains the names of species the user wants to plot, listed with no breaks between species and without the state indicators (e.g. CH4,H2O,N2). Plots produced by any run will be

saved in the `plots/` directory placed in the TEA working directory (thus above the directories given by the user in each run, `<DIRECTORY_NAME>`). To differentiate between plots, the plot name will contain the name of the output (atmosphere file) used. Additionally, each plots' title will also carry the current output filename. An example of the `plotTEA.py` output is given in the `../TEA/doc/examples/plots/` directory.

9 Potential User Errors

The following are some possible errors the user may encounter when making a TEA input file. Any of the following scenarios will cause the file in question to conflict with TEA's normal run-time scenarios and will result in erroneous abundances and/or crashes:

1. The user should have unique T, P values for each line in the PT-profile file before running `makeatm.py`, otherwise each repeated layer will be overwritten and the final atmosphere file will end up with fewer lines.
2. `input_elem` must have names as they appear in the periodic table.
3. `out_species` must include all input elements with their states as `readJANAF.py` produces them. See the `../TEA/lib/gdata/` folder and the `conversion_record_sorted.txt` file for the correct names of the species.
4. Should the code stall on the first iteration of the first temperature, check if all elements that appear in the species list are included with their correct names.
5. Elemental hydrogen (H) and helium (He) must be included in `in_elem` for hot-Jupiter atmosphere calculations. Similarly, the H_g, He_ref and H2_ref species must also appear in `out_spec` for these calculations.

10 Examples

Four directories are listed in `../TEA/doc/examples/`: `singleTP`, `multiTP`, `PT`, and `plots`. The `singleTP` and `multiTP` folders each contain the full TEA run with all inputs/outputs folders. The `PT` directory contains several PT-profile file examples. The `plots` directory contains an example plot produced by `plotTEA.py` using the final atmosphere file from the aforementioned multiple T, P run.

For the above single T, P run, the `inputs/` folder contains an example of the single T, P input file. The user should refer to this file when manually producing their own single T, P input file. For the above multiple T, P run the input pre-atmosphere file is contained in both the `atm_inputs/` and `inputs/` folder. If the pre-atmosphere file is produced using `makeatm.py`, the user should follow the PT-profile file format provided in the `PT` directory.

The `inputs/` directory for all runs contain all input files used for that run. As such, the user can easily replicate any run by ensuring the same inputs are used. The commands how to run the examples with the total expected execution time are provided in the `install_guide.txt` file, placed in the `../TEA/doc/`.

11 Current Limitations

At the time of writing, the TEA code works only for systems containing purely gaseous species. It works with high numerical precision, without adjustments, for the temperatures above $\sim$ 600 K. For temperatures below 600 K and mixing fractions below $10e^{-14}$, the code produces results with low precision. The output can be improved with fine adjustments to the lambda exploration variables in lambdacorr.py module (see TEA Code Description document by Blecic $ Bowman).

References

Asplund, M., Grevesse, N., Sauval, A. J., & Scott, P. 2009, ARA&A, 47, 481, ADS, 0909.0948
Blecic, J., Harrington, J., & Bowman, M. O. 2015, in prep for ApJSupp
Eriksson, G. 1971, Acta Chem.Scand.
White, W. B., Johnson, S. M., & Dantzig, G. B. 1958, J. Chem. Phys., 28, 751, ADS