pmutt Documentation

Release 1.2.19

Vlachos Research Group

Apr 08, 2020

Getting Started

CHAPTER 1

Python Multiscale Thermochemistry Toolbox (pMuTT)

The **P**ython **Mu**ltiscale **T**hermochemistry **T**oolbox (pMuTT) is a Python library for Thermochemistry developed by the Vlachos Research Group at the University of Delaware. This code was originally developed to convert *ab-initio* data from DFT to observable thermodynamic properties such as heat capacity, enthalpy, entropy, and Gibbs energy. These properties can be fit to empirical equations and written to different formats.

1.1 Documentation

See our documentation page for examples, equations used, and docstrings.

1.2 Developers

- Jonathan Lym (jlym@udel.edu)
- Gerhard Wittreich, P.E. (wittregr@udel.edu)

1.3 Dependencies

- Python3
- Atomic Simulation Environment: Used for I/O operations and to calculate some thermodynamic properties
- Numpy: Used for vector and matrix operations
- Pandas: Used to import data from Excel files

- xlrd: Used by Pandas to import Excel files
- SciPy: Used for fitting heat capacities and generating smooth curves for reaction coordinate diagram
- Matplotlib: Used for plotting thermodynamic data
- pyGal: Similar to Matplotlib. Used for plotting interactive graphs
- PyMongo: Used to read/write to databases
- dnspython: Used to connect to databases
- NetworkX: Used to plot reaction networks
- More Itertools: Used for writing ranges for OpenMKM output.
- PyYAML: Used to write YAML input files for OpenMKM.

1.4 Getting Started

1. Install using pip (see documentation for more thorough instructions):

```
pip install pmutt
```

2. Look at examples using the code
3. Run the unit tests.

1.5 License

This project is licensed under the MIT License - see the LICENSE.md file for details.

1.6 Publications

- J. Lym, G.R. Wittreich and D.G. Vlachos, A Python Multiscale Thermochemistry Toolbox (pMuTT) for thermochemical and kinetic parameter estimation, Computer Physics Communications (2019) 106864, https://doi.org/10.1016/j.cpc.2019.106864.

1.7 Contributing

If you have a suggestion or find a bug, please post to our Issues page with the enhancement or bug tag respectively.

Finally, if you would like to add to the body of code, please:

- fork the development branch
- make the desired changes
- write the appropriate unit tests
- submit a pull request.

1.8 Questions

If you are having issues, please post to our Issues page with the help wanted or question tag. We will do our best to assist.

1.9 Funding

This material is based upon work supported by the Department of Energy's Office of Energy Efficient and Renewable Energy's Advanced Manufacturing Office under Award Number DE-EE0007888-9.5.

1.10 Special Thanks

- Dr. Jeffrey Frey (pip and conda compatibility)
- Jaynell Keely (Logo design)

CHAPTER 2

Installation

2.1 Installing Python

Anaconda is the recommended method to install Python for scientific applications. It is supported on Linux, Windows and Mac OS X. Download Anaconda here. Note that pMuTT runs on Python 3.X.

2.2 Installing pMuTT using pip

Using pip is the most straightforward way to install pMuTT.

1. Open a command prompt with access to Python (if Python is installed via Anaconda on Windows, open the Anaconda Prompt from the start menu).

2. Install pMuTT by typing the following in the command prompt:

```
pip install pmutt
```

The output towards the end should state "Successfully built pMuTT" if the installation was successful.

Receiving installation errors? Post the error to our Issues page.

2.2.1 Common installation errors

PyYAML Uninstallation Error

Error:

```
ERROR: Cannot uninstall 'PyYAML'. It is a distutils installed project and
thus we cannot accurately determine which files belong to it which would
lead to only a partial uninstall.
```

Solution

Append `--ignore-installed PyYAML` to pip command.:

```
pip install pmutt --ignore-installed PyYAML
```

See issue regarding the PyYAML error here.

Pip Permission Error

Error:

```
Could not install packages due to an EnvironmentError: [Errno 13] Permission
denied: '/usr/local/bin/pmutt'
```

Solution

Append `--user` to pip command.:

```
pip install pmutt --user
```

See explanation why this permission error occurs here.

2.3 Installing pMuTT from source

If you would prefer to install from source or you are interested in development, follow the instructions below.

```
pip install git+https://github.com/VlachosGroup/pMuTT.git
```

2.4 Installing the developer branch

pMuTT has a release roughly once a month. Changes that will be in the next release will be located in the Developer branch but may have more bugs than the master branch. You can install using the following:

```
pip install --upgrade git+https://github.com/VlachosGroup/pMuTT.git@development
```

2.5 Upgrading pMuTT using pip

To upgrade to a newer release, use the –upgrade flag:

```
pip install --upgrade pmutt
```

2.6 Running unit tests

pMuTT has a suite of unit tests that should be run before committing any code. To run the tests, run the following commands in a Python terminal.

```
import pmutt
pmutt.run_tests()
```

The expected output is shown below. The number of tests will not necessarily be the same.

```
.........................
----------------------------------------------------------------------
Ran 25 tests in 0.020s

OK
```

CHAPTER 3

Examples

This page and its subpages show some examples using the pMuTT code. There are several formats available.

To run examples on Binder, simply click the logo and wait for the session to be initialized.

To run examples on HTML, Jupyter, or Python, it is recommended to download the ZIP folder. This will contain other files (like spreadsheets) that may be required for the code to run.

If a link leads to a page of text, right-click any empty space on that page and choose "Save Page As". It will download to your computer in the appropriate format.

The pros and cons of each format are listed below.

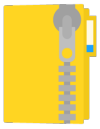
Criteria	Binder	HTML	Jupyter	Python
File in ZIP Folder	N	Y	Y	Y
pMuTT installation req.	N	N	Y	Y
Jupyter installation req.	N	N	Y	N
Other files (e.g. spreadsheets) req.	N	N	Y	Y
Non-code elements easy to read	Y	Y	Y	N
Code elements easily editable	Y	N	Y	Y

3.1 Overview

3.1.1 Topics Covered

- Using constants and converting units using the `constants` module
- Initializing `StatMech` objects by specifying all modes and by using *Presets*
- Initializing empirical objects such as `Nasa` objects using a `StatMech` object or from a previously generated Nasa polynomial
- Initializing `Reference` and `References` objects to adjust DFT's reference to more traditional references
- Input (via Excel) and output `Nasa` polynomials to thermdat format
- Initializing `Reaction` objects from strings

3.2 Experimental to Empirical



3.2.1 Topics Covered

- Using pmutt's constants for unit conversions
- Create a `Shomate` object from experimental data
- Calculate thermodynamic properties using the `Shomate` object
- Plot the shape of the `Shomate` curve
- Save the `Shomate` object as a JSON file

3.3 Excel to Empirical Data

3.3.1 Topics Covered

- Reading *ab-initio* data from an Excel file
- Initialize `Reference` objects and a `References` object
- Generate a `Nasa` object using `StatMech` models
- Write `Nasa` object to a `thermdat` file

3.4 Reaction

3.4.1 Topics Covered

- Read a thermdat file and convert it to a dictionary of Nasa objects
- Initialize a `Reaction` object manually and from strings
- Add a BEP relationship to a `Reaction` object
- Calculate thermodynamic and kinetic properties using the `Reaction` object
- Save the `Reaction` object as a `JSON` file

3.5 Chemkin_IO

3.5.1 Topics Covered

- Read species *ab-initio* data, reactions, and catalyst sites from a spreadsheet
- Write the thermdat, gas.inp, surf.inp, T_flow.inp, EAg.inp, EAs.inp, tube_mole.inp files

---

3.6 OpenMKM_IO

3.6.1 Topics Covered

- Read species *ab-initio* data, reactions, lateral interactions and phases from a spreadsheet
- Write the CTI input file

3.7 Phase Diagram

3.7.1 Topics Covered

- Create `Nasa` and `StatMech` objects
- Initialize `Reaction` objects to describe the formation reaction of FeOx species
- Generate a 1D phase diagram by varying T
- Generate a 2D phase diagram by varying T and P
- Save the `PhaseDiagram` object as a `JSON` file

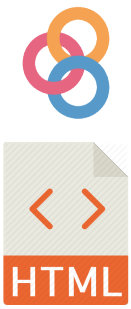
3.8 Workshops

3.9 NAM26 Workshop

3.9.1 Topics Covered

- Using constants and converting units using the `constants` module
- Initializing `StatMech` objects by specifying all modes and by using *Presets*
- Initializing empirical objects such as `Nasa` objects using a `StatMech` object or directly using the polynomial
- Input (via Excel) and output `Nasa` polynomials to thermdat format
- Initializing `Reaction` objects from strings

3.10 AIChE 2019 Workshop




jupyter

CHAPTER 4

Workshops

Links to workshops involving pMuTT can be found here.

4.1 NAM 2019

Instructions and materials for the "Theory, Applications, and Tools for Kinetic Modeling" workshop can be found here.

4.1.1 Setup instructions

If you encounter a problem while setting up, check our pMuTT issues page for help. When creating a new issue, make sure to tag it as NAM2019 to help others! We will do our best to assist you.

pMuTT instructions

1. Install Anaconda with Python 3.7. Use default settings.
2. Install pMuTT using the installation instructions.
3. Download `pmutt_nam2019.zip` from the workshop materials.
4. Unzip the folder, find the `SampleJupyterNotebook.ipynb` notebook and open it. You may have to install Jupyter by opening the Anaconda Navigator from the start menu.
5. Run the cell with the Python code to ensure the most updated version of pMuTT is installed.

OpenMKM

This workshop had a microkinetic modeling section but the license was only available during the session. We will be releasing an open-source microkinetic modeling software (OpenMKM) soon. If interested, please e-mail Jon Lym (jlym@udel.edu) to sign up for the mailing list.

Files

- Slides and the Jupyter Notebook can be found here.
- GitHub view of Jupyter notebook. Note this will not render pictures used in the presentation.

Mailing List

If you are interested in software developments, please sign up for our mailing list by e-mailing Jon Lym (jlym@udel.edu) to sign up!

4.2 AICHE 2019

Materials used in the workshop are available here. <https://www.dropbox.com/sh/lt9iavcyifyi646/AABXSVkiSg1z_YI8WNmiISnPa?dl=0

CHAPTER 5

Publications

If you use pMuTT for your research, please cite the following reference.

```
J. Lym, G.R. Wittreich and D.G. Vlachos, A Python Multiscale Thermochemistry
Toolbox (pMuTT) for thermochemical and kinetic parameter estimation, Computer
Physics Communications (2019) 106864,
https://doi.org/10.1016/j.cpc.2019.106864.
```

Download citation as BibTex file.

5.1 Logos

When citing our software in a presentation or other reproducible format, you may use the high-resolution logos below.

Default

Inverse

Black

White

5.2 Cited By

1. Wang, C., Wittreich, G.R., Lin, C. et al. Catal Lett (2019). https://doi.org/10.1007/s10562-019-03027-8
2. Fu, J.; Lym, J.; Zheng, W.; Alexopoulos, K.; Mironenko, A. V.; Li, N.; Boscoboinik, J. A.; Su, D.; Weber, R. T.; Vlachos, D. G. Nat. Catal. 2020. https://doi.org/10.1038/s41929-020-0445-x

If you have published work using pMuTT and would like to be featured here, please post an issue with the publication tag.

CHAPTER 6

Referencing

Enthalpies calculated using VASP (and some other computational methods) have different references than standard references (i.e. the enthalpy of formation of pure substances, like O_2 or Pt, is not necessarily zero). This difference makes it difficult to ensure thermodynamic consistency for our mechanisms since we may be mixing experimental gas thermodynamics with computational surface thermodynamics. In order to make the references consistent, we find a correction factor for each element by solving the equation:

$$\underline{\underline{H}}^{expt}_{[M\times N]} = \underline{\underline{H}}^{DFT}_{[M\times N]} + \underline{\underline{x}}_{[M\times N]}\underline{\mathbf{h}}_{[N]}$$

where M is the number of reference species, N is the number of elements, H^{expt} is the experimental standard enthalpies, H^{DFT} is the standard enthalpies calculated using DFT, **x** is a matrix that describes the composition of the references (each row represents a specie, each column represents an element), and **h** is the correction for each element.

The equation can be solved using a Least Squares approach. The correction factor can then be added to subsequent species calculated through DFT to ensure consistent references. Referencing is handled by the `References`.

CHAPTER 7

Constants

pMuTT will soon migrate its unit conversions and constants to VUnits!

7.1 Constants

R
kb
h
c
Na
P0
T0
V0
e
m_e
m_p

7.2 Unit Conversions

convert_unit
debye_to_einstein
einstein_to_debye
energy_to_freq
energy_to_temp
energy_to_wavenumber
freq_to_energy

Continued on next page

Table 2 – continued from previous page

freq_to_temp
freq_to_wavenumber
inertia_to_temp
temp_to_energy
temp_to_freq
temp_to_wavenumber
wavenumber_to_energy
wavenumber_to_freq
wavenumber_to_inertia
wavenumber_to_temp

7.3 Misc.

atomic_weight
prefixes
symmetry_dict
type_dict

CHAPTER 8

Input and Output

Input and output to different forms is an active area of development for pMuTT.

8.1 Excel

excel.read_excel

8.1.1 Special Rules

Special rules can be defined in the `read_excel()` function to process inputs differently. Currently supported special rules are listed below.

excel.set_element
excel.set_formula
excel.set_atoms
excel.set_statmech_model
excel.set_trans_model
excel.set_vib_model
excel.set_rot_model
excel.set_elec_model
excel.set_nucl_model
excel.set_vib_wavenumbers
excel.set_rot_temperatures
excel.set_nasa_a_low
excel.set_nasa_a_high
excel.set_list_value
excel.set_dict_value

8.2 Thermdat

This is the output format used for Chemkin. A list of NASA objects can be written to a thermdat file.

`thermdat.read_thermdat`
`thermdat.write_thermdat`

8.3 JSON

JavaScript Object Notation (JSON) is a format that is easily read and written by both humans and machines. All pmutt objects with the methods `to_dict` and `from_dict` are JSON compatible.

`json.pmuttEncoder`
`json.json_to_pmutt`

8.3.1 Examples

Saving pmutt objects can be done by using `pmuttEncoder()`.

```
import json
from pmutt.io.json import pmuttEncoder

with open(json_path, 'w') as f_ptr:
    json.dump(pmutt_obj, f_ptr, cls=pmuttEncoder, indent=True)
```

Loading pmutt objects can be done by using the object hook: `json_to_pmutt()`.

```
import json
from pmutt.io.json import json_to_pmutt

with open(json_path, 'r') as f_ptr:
    pmutt_obj = json.load(f_ptr, object_hook=json_to_pmutt)
```

8.3.2 Sample JSON File

JSON writes in a human-readable syntax. An example showing a H2 `Shomate` object in JSON format is shown below.

```
{
 "class": "<class 'pmutt.empirical.shomate.Shomate'>",
 "type": "shomate",
 "name": "H2",
 "phase": "G",
 "elements": {
  "H": 2
 },
 "notes": null,
 "smiles": null,
```

(continues on next page)

(continued from previous page)

```
 "model": null,
 "misc_models": [
  {
   "class": "<class 'pmutt.empirical.GasPressureAdj'>"
  }
 ],
 "a": [
  33.066178,
  -11.363417,
  11.432816,
  -2.772874,
  -0.158558,
  -9.980797,
  172.707974,
  0.0
 ],
 "T_low": 298.0,
 "T_high": 1000.0,
 "units": "J/mol/K"
}
```

8.3.3 Creating New pmutt Classes

Encoding

To ensure your new class can be encoded using the `pmuttEncoder`, the `to_dict()` method should be implemented. One of the entries of the dictionary should be `'class': str(self.__class__)` so that it can be decoded later. The other elements should be the attributes that can be used to reinitialize the object and must be JSON-supported objects. A simple example using `FreeTrans` is shown below.

```
def to_dict(self):
    return {'class': str(self.__class__),
            'n_degrees': self.n_degrees,
            'molecular_weight': self.molecular_weight}
```

If the attributes are not supported by JSON (such as other pmutt objects), use their `to_dict()` methods to convert to JSON-supported objects. An example using `StatMech` is shown below.

```
def to_dict(self):
    return {'class': str(self.__class__),
            'trans_model': self.trans_model.to_dict(),
            'vib_model': self.vib_model.to_dict(),
            'rot_model': self.rot_model.to_dict(),
            'elec_model': self.elec_model.to_dict(),
            'nucl_model': self.nucl_model.to_dict()}
```

Decoding

To ensure your object can be decoded using the `json_to_pmutt` object hook, add an entry to the dictionary in the `pmutt.io.json.type_to_class` method. The key should be the type of your object in string format (i.e. the result of `str(self.__class__)`). Your class should also have the `from_dict()` class method to reinitialize your object. A simple example using `FreeTrans` is shown below.

```
from pmutt.io.json import remove_class

@classmethod
def from_dict(cls, json_obj):
    json_obj = remove_class(json_obj)
    return cls(**json_obj)
```

Similarly to encoding, sometimes your object contains pmutt objects. You can use the `json_to_pmutt` object hook to remake these objects. An example using `StatMech` is shown below.

```
from pmutt.io import json as json_pmutt

@classmethod
def from_dict(cls, json_obj):
    json_obj = remove_class(json_obj)
    trans_model = json_pmutt.json_to_pmutt(json_obj['trans_model'])
    vib_model = json_pmutt.json_to_pmutt(json_obj['vib_model'])
    rot_model = json_pmutt.json_to_pmutt(json_obj['rot_model'])
    elec_model = json_pmutt.json_to_pmutt(json_obj['elec_model'])
    nucl_model = json_pmutt.json_to_pmutt(json_obj['nucl_model'])

    return cls(trans_model=trans_model,
               vib_model=vib_model,
               rot_model=rot_model,
               elec_model=elec_model,
               nucl_model=nucl_model)
```

8.4 YAML

YAML Ain't Markup Language (YAML) is a human friendly data serialization standard for all programming languages. All pmutt objects are natively supported by YAML.

8.4.1 Examples

Saving pmutt objects can be done by using `pmuttEncoder()`.

```
import yaml

with open(yaml_path, 'w') as f_ptr:
    yaml.dump(pmutt_obj, f_ptr)
```

Loading pmutt objects can be done by using the object hook: `json_to_pmutt()`.

```
import yaml

with open(yaml_path, 'r') as f_ptr:
    pmutt_obj = yaml.load(f_ptr)
```

8.4.2 Sample YAML File

YAML writes in a human-readable syntax. An example showing a H2 `Shomate` object in YAML format is shown below.

```
!!python/object:pmutt.empirical.shomate.Shomate
T_high: 1000.0
T_low: 298.0
_units: J/mol/K
a: !!python/object/apply:numpy.core.multiarray._reconstruct
  args:
  - !!python/name:numpy.ndarray ''
  - !!python/tuple [0]
  - !!binary |
    Yg==
  state: !!python/tuple
  - 1
  - !!python/tuple [8]
  - !!python/object/apply:numpy.dtype
    args: [f8, 0, 1]
    state: !!python/tuple [3, <, null, null, null, -1, -1, 0]
  - false
  - !!binary |
    e9tMhXiIQEDxngPLEbomwP9eCg+a3SZA/G1PkNguBsAZdELooEvEv6MHPgYr9iPAYw0XuaeWZUAA
    AAAAAAAAAA==
elements: {H: 2}
misc_models:
- !!python/object:pmutt.empirical.GasPressureAdj {}
model: null
n_sites: null
name: H2
notes: null
phase: G
smiles: null
```

8.5 VASP

vasp.set_vib_wavenumbers_from_outcar

8.6 Gaussian

gaussian.read_pattern
gaussian.read_zpe
gaussian.read_electronic_and_zpe
gaussian.read_frequencies
gaussian.read_rotational_temperatures
gaussian.read_molecular_mass
gaussian.read_rot_symmetry_num

8.7 Chemkin

chemkin.read_reactions
chemkin.write_EA
chemkin.write_gas
chemkin.write_surf
chemkin.write_T_flow
chemkin.write_tube_mole

8.8 OpenMKM

omkm.write_cti
omkm.write_yaml
omkm.get_species_phases
omkm.get_reactions_phases
omkm.get_interactions_phases
omkm.organize_phases

CHAPTER 9

Empirical Models

Empirical models predict thermodynamic properties of species using simple polynomials. These can be evaluated more quickly than statistical mechanical equations and tend to be used by kinetic modeling software.

When fitting an empirical relationship, be mindful that the resulting polynomial will have the same reference state as the original data. Databases like the NIST Chemistry WebBook and Burcat tend to use the standard reference (i.e. pure elements in their natural state at 1 atm have a standard enthalpy of formation of 0). However, data from DFT software (and DFT-based references) may use different references. See *Referencing* for more information.

9.1 Parent Class

`EmpiricalBase`

9.2 Nasa

`nasa.Nasa`
`nasa.SingleNasa9`
`nasa.Nasa9`

9.3 Shomate

`shomate.Shomate`

9.4 Referencing

Referencing is used to bridge the gap between *ab-initio* code reference states and the standard state reference. See *Referencing* for more information.

`references.Reference`
`references.References`

9.5 Misc.

Miscellaneous models associated with the empirical models are located here.

`GasPressureAdj`

CHAPTER 10

Statistical Thermodynamic Models

10.1 StatMech Base Class

The `StatMech` class stores the classes associated with each mode. Each mode has methods that specifies how to calculate thermodynamic quantities, such as partition functions, heat capacities, internal energy, enthalpy, entropy, Helmholtz energy and Gibbs energy.

`StatMech`

The `StatMech` object can be initialized in two ways:

1. by each passing each mode as objects or
2. passing each mode as a class and the associated parameters.

The examples below give an equivalent result:

10.1.1 Example of initialization using objects

```
import numpy as np
from ase.build import molecule
from pmutt.statmech import StatMech, trans, vib, rot, elec

atoms = molecule('H2O')
H2O_trans = trans.FreeTrans(n_degrees=3, atoms=atoms)
H2O_vib = vib.HarmonicVib(vib_wavenumbers=[3825.434, 3710.2642, 1582.432])
H2O_rot = rot.RigidRotor(symmetrynumber=2, atoms=atoms)
H2O_elec = elec.GroundStateElec(potentialenergy=-14.2209, spin=0)
H2O_statmech = StatMech(trans_model=H2O_trans,
                        vib_model=H2O_vib,
                        rot_model=H2O_rot,
                        elec_model=H2O_elec)
```

10.1.2 Example of initialization using classes and parameters

```
import numpy as np
from ase.build import molecule
from pmutt.statmech import StatMech, trans, vib, rot, elec


H2O_statmech = StatMech(trans_model=trans.FreeTrans,
                        n_degrees=3,
                        vib_model=vib.HarmonicVib,
                        vib_wavenumbers=[3825.434, 3710.2642, 1582.432],
                        rot_model=rot.RigidRotor,
                        symmetrynumber=2,
                        atoms=molecule('H2O'),
                        elec_model=elec.GroundStateElec,
                        potentialenergy=-14.2209,
                        spin=0)
```

10.2 Presets

If you are using a common model (e.g. the ideal gas model), then you can get the default parameters from the dictionary, `pmutt.statmech.presets`. The same H2O StatMech object can be specified without the need to pass all the types of modes:

```
from ase.build import molecule
from pmutt.statmech import StatMech, presets

idealgas_defaults = presets['idealgas']
H2O_new = StatMech(vib_wavenumbers=[3825.434, 3710.2642, 1582.432],
                   potentialenergy=-14.2209,
                   atoms=molecule('H2O'),
                   spin=0,
                   symmetrynumber=2,
                   **idealgas_defaults)
```

Currently supported presets are described below. The first table shows the attributes already specified, and the second table shows the attributes that are still required, and the third table shows the required and optional parameters to calculate a thermodynamic property (where the value in parentheses is the default value).

10.2.1 Ideal Gas (idealgas)

Useful for modeling ideal gases. Assumes the system has:

- 3 degrees of translational freedom and no interactions with other molecules
- harmonic vibrations
- rigid rotor rotations
- electronic ground state
- nuclear ground state

Set Attributes	Default Value
trans_model	`FreeTrans`
n_degrees	3
vib_model	`HarmonicVib`
elec_model	`GroundStateElec`
rot_model	`RigidRotor`

Required Attributes	Description
molecular_weight	(float) Molecular weight in g/mol
vib_wavenumbers	(list of float) Vibrational wavenumbers in 1/cm
potentialenergy	(float) Electronic potential energy in eV
spin	(float) Electron spin. 0 if all electrons are paired (e.g. N2), 0.5 if the specie is a radical (e.g. CH3.), 1 if the specie exists as a triplet (e.g. O2).
geometry	(str) Geometry of molecule
rot_temperatures	(list of float) Rotational temperatures in K
symmetrynumber	(int) Symmetry number
atoms	(ase.Atoms object) Optional. If this parameter is specified, `molecular_weight`, `geometry`, `rot_temperatures`, and `potentialenergy` do not have to be specified.

Thermodynamic Quantity	Expected Parameters	Optional Parameters
q	T	• ignore_q_elec (False) • P (1.01325 bar)
$\frac{C_V}{R}$	T	
$\frac{C_P}{R}$	T	
$\frac{U}{RT}$	T	
$\frac{H}{RT}$	T	
$\frac{S}{R}$	T	P (1.01325 bar)
$\frac{A}{RT}$	T	P (1.01325 bar)
$\frac{G}{RT}$	T	P (1.01325 bar)

10.2.2 Harmonic Approximation (harmonic)

Typically used to model adsorbates. Assumes the system has:

- harmonic vibrations
- electronic ground state

Parameter	Default Value
vib_model	`HarmonicVib`
elec_model	`GroundStateElec`

Required Parameters	Description
vib_wavenumbers	(list of float) Vibrational wavenumbers in 1/cm
potentialenergy	(float) Electronic potential energy in eV
spin	(float) Electron spin
atoms	(ase.Atoms object) Optional. If this parameter is specified, `potentialenergy` does not have to be specified.

Thermodynamic Quantity	Expected Parameters	Optional Parameters
q	T	ignore_q_elec (True)
$\frac{C_V}{R}$	T	
$\frac{C_P}{R}$	T	
$\frac{U}{RT}$	T	
$\frac{H}{RT}$	T	
$\frac{S}{R}$	T	
$\frac{A}{RT}$	T	
$\frac{G}{RT}$	T	

10.2.3 Electronic (electronic)

Typically used to model systems where the electronic modes dominate and changes due to temperature are not important. Assumes the system has:

- electronic ground state

Parameter	Default Value
elec_model	`GroundStateElec`

Required Parameters	Description
potentialenergy	(float) Electronic potential energy in eV
spin	(float) Electron spin
atoms	(ase.Atoms object) Optional. If this parameter is specified, `potentialenergy` does not have to be specified

Thermodynamic Quantity	Expected Parameters	Optional Parameters
q	T	ignore_q_elec (True)
$\frac{C_V}{R}$		
$\frac{C_P}{R}$		
$\frac{U}{RT}$	T	
$\frac{H}{RT}$	T	
$\frac{S}{R}$		
$\frac{A}{RT}$	T	
$\frac{G}{RT}$	T	

10.2.4 Placeholder (placeholder)

Typically used to model species that have no contribution. The partition function is set to 1 and all other thermodynamic quantities evaluate to 0.

Parameter	Default Value
trans_model	`EmptyMode`
vib_model	`EmptyMode`
rot_model	`EmptyMode`
elec_model	`EmptyMode`
nucl_model	`EmptyMode`

Required Parameters	Description
N/A	

Thermodynamic Quantity	Expected Parameters	Optional Parameters
q		
$\frac{C_V}{R}$		
$\frac{C_P}{R}$		
$\frac{U}{RT}$		
$\frac{H}{RT}$		
$\frac{S}{R}$		
$\frac{A}{RT}$		
$\frac{G}{RT}$		

10.2.5 Constant (constant)

Arbitrarily specify a constant for each quantity. This is primarily used for testing so be careful as you can disobey some fundamental thermodynamic quantities. For example,

$G = H - TS$

may not be obeyed.

Parameter	Default Value
elec_model	`ConstantMode`

Required Parameters	Description
q	(float) Optional. Partition function. Default is 1
Cv	(float) Optional. Heat capacity at constant volume in eV/K. Default is 0
Cp	(float) Optional. Heat capacity at constant pressure in eV/K. Default is 0
U	(float) Optional. Internal energy in eV. Default is 0
H	(float) Optional. Enthalpy in eV. Default is 0
S	(float) Optional. Entropy in eV/K. Default is 0
F	(float) Optional. Helmholtz energy in eV. Default is 0
G	(float) Optional. Gibbs energy in eV. Default is 0

Thermodynamic Quantity	Expected Parameters	Optional Parameters
q		
$\frac{C_V}{R}$		
$\frac{C_P}{R}$		
$\frac{U}{RT}$		
$\frac{H}{RT}$		
$\frac{S}{R}$		
$\frac{A}{RT}$		
$\frac{G}{RT}$		

The `pmutt.statmech.presets` dictionary is flexible where you can create a new entry if you will use a model often.

10.3 Translational Models

`trans.FreeTrans`

10.4 Vibrational Models

`vib.HarmonicVib`
`vib.QRRHOVib`
`vib.EinsteinVib`
`vib.DebyeVib`

10.5 Rotational Models

`rot.RigidRotor`

10.6 Electronic Models

`elec.GroundStateElec`
`lsr.LSR`

10.7 Nuclear Models

Typically these are unimportant for chemical reactions, but the module is present in case nuclear modes become important in the future.

`nucl.EmptyNucl`

10.8 Misc.

`EmptyMode`
`ConstantMode`

10.9 Creating New StatMech Models

New models should inherit from `pmutt._ModelBase`. This class already has the dimensional methods (e.g. `get_Cv`, `get_U`, `get_S`), and some routine methods (e.g. `get_FoRT`, `get_GoRT`, `from_dict`, `to_dict`) implemented.

For full accessibility, the following methods should be implemented:

- `get_q`
- `get_CvoR`
- `get_CpoR`
- `get_UoRT`
- `get_HoRT`
- `get_SoR`

CHAPTER 11

Mixture Models

These models allow the user to excess thermodynamic properties that arise from species mixing. Mixture models are supported by `Nasa`, `Shomate`, `Nasa9` and `StatMech` classes.

11.1 Coverage Effects

`cov.PiecewiseCovEffect`

CHAPTER 12

Kinetic Models

12.1 Chemkin

Classes and functions related to writing Chemkin files. Please note these modules were written specifically for the Vlachos group's in-house version. Features may or may not be supported in the commercial version.

12.1.1 Classes

chemkin.CatSite
reaction.ChemkinReaction

12.1.2 Input and Output

io.chemkin.read_reactions
io.chemkin.write_EA
io.chemkin.write_gas
io.chemkin.write_surf
io.chemkin.write_T_flow
io.chemkin.write_tube_mole

12.1.3 Examples

- *Chemkin_IO*

12.2 Cantera

Classes and functions related to Cantera. This functionality is still in its early stages.

12.2.1 Phases

cantera.phase.Phase
cantera.phase.IdealGas
cantera.phase.StoichSolid

12.2.2 Units

cantera.units.Units

12.2.3 Input and Output

io.cantera.obj_to_CTI

12.3 OpenMKM

Classes and functions related to OpenMKM. This functionality is still in its early stages.

12.3.1 Phases

omkm.phase.IdealGas
omkm.phase.StoichSolid
omkm.phase.InteractingInterface

12.3.2 Reactions

omkm.reaction.SurfaceReaction
omkm.reaction.BEP

12.3.3 Units

omkm.units.Units

12.3.4 Input and Output

io.omkm.write_cti
io.omkm.write_yaml
io.omkm.get_species_phases
io.omkm.get_reactions_phases
io.omkm.get_interactions_phases
io.omkm.organize_phases

12.3.5 Examples

- *OpenMKM_IO*

12.4 Zacros

Classes related to Zacros.

12.4.1 Classes

empirical.zacros.Zacros

CHAPTER 13

Reactions

13.1 Reaction Objects

These classes are used to model a single chemical reaction. `ChemkinReaction` and `SurfaceReaction` have additional formatting options for the appropriate modeling software.

`reaction.Reaction`
`reaction.ChemkinReaction`
`omkm.reaction.SurfaceReaction`

13.2 Reactions Objects

These classes model multiple reactions. Multiple reactions can be used together for phase diagrams, reaction coordinate diagrams, and other purposes.

`reaction.Reactions`
`reaction.ChemkinReaction`
`omkm.reaction.SurfaceReaction`

13.3 BEP

Bronsted Evans Polyani relationships can be added to `Reaction` objects so the activation energy can be estimated using the change in enthalpy.

reaction.bep.BEP
omkm.reaction.BEP

CHAPTER 14

Phase Diagrams

Phase diagrams can be made by creating a list of `Reaction` objects. The reactions should be balanced and have the same reactant references to give meaningful data.

`PhaseDiagram`

14.1 Examples

- *Phase Diagram*

CHAPTER 15

Equations of State

Use equations of state to calculate P, V, T, or n.

`IdealGasEOS`
`vanDerWaalsEOS`

CHAPTER 16

Visualization

This page contains various functions to visualize functions.

16.1 Plot_1D

16.1.1 Example

Below, we plot the enthalpy, entropy, and Gibbs energy of a water molecule as a function of temperature at 1 bar and 10 bar.

```
import numpy as np
from pmutt import plot_1D
from pmutt.examples import H2O_statmech
from matplotlib import pyplot as plt

T = np.linspace(300., 500.) # K
fig1, ax1 = plot_1D(H2O_statmech, x_name='T', x_values=T,
                    methods=('get_H', 'get_S', 'get_G'),
                    get_H_kwargs={'units':'kcal/mol'},
                    get_S_kwargs={'units':'cal/mol/K'},
                    get_G_kwargs={'units': 'kcal/mol'})

# Passing the figure and ax arguments superimpose the plots
fig1, ax1 = plot_1D(H2O_statmech, x_name='T', x_values=T,
                    P=10., methods=('get_H', 'get_S', 'get_G'),
                    get_H_kwargs={'units':'kcal/mol'},
                    get_S_kwargs={'units':'cal/mol/K'},
                    get_G_kwargs={'units': 'kcal/mol'},
                    figure=fig1, ax=ax1)

# Add legend to matplotlib axes
ax1[2].legend(['1 bar', '10 bar'])
```

(continues on next page)

(continued from previous page)

```
ax1[0].set_ylabel('H (kcal/mol)')
ax1[1].set_ylabel('S (cal/mol/K)')
ax1[2].set_ylabel('G (kcal/mol)')
plt.show()
```

16.2 Plot_2D

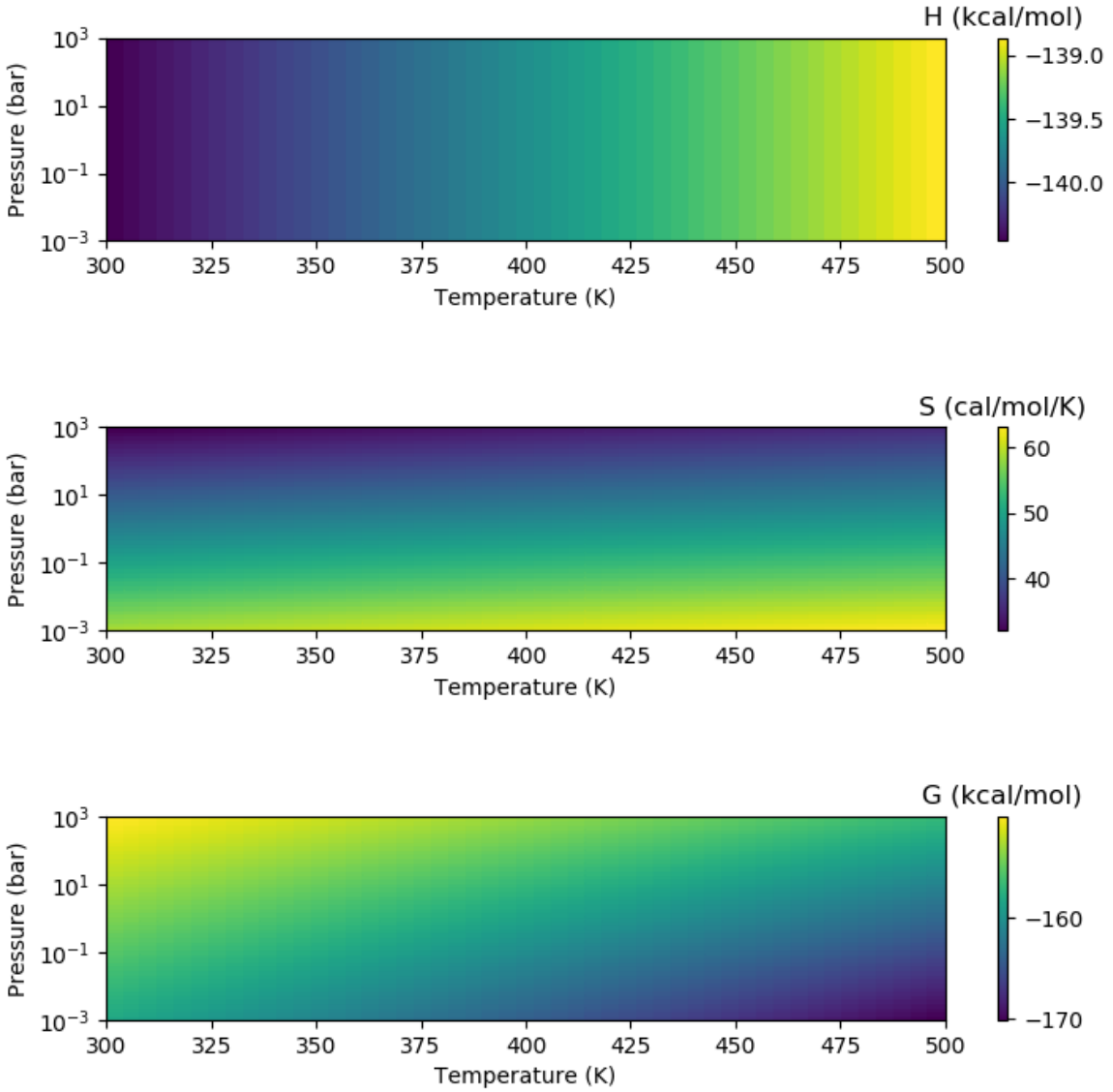
Below, we plot the enthalpy, entropy, and Gibbs energy of a water molecule as a function of temperature and pressure.

16.2.1 Example

```
import numpy as np
from pmutt import plot_2D
from pmutt.examples import H2O_statmech
from matplotlib import pyplot as plt

T = np.linspace(300., 500.) # K
P = np.logspace(-3, 3) # bar

fig1, ax1, c1, cbar1 = plot_2D(H2O_statmech,
                               x1_name='T', x1_values=T,
                               x2_name='P', x2_values=P,
                               methods=('get_H', 'get_S', 'get_G'),
                               get_H_kwargs={'units':'kcal/mol'},
                               get_S_kwargs={'units':'cal/mol/K'},
                               get_G_kwargs={'units': 'kcal/mol'})
plt.show()
```


H (kcal/mol)
−139.0
−139.5
−140.0
Pressure (bar)
10^3
10^1
10^{-1}
10^{-3}
300
325
350
375
400
425
450
475
500
Temperature (K)
S (cal/mol/K)
60
50
40
Pressure (bar)
Temperature (K)
G (kcal/mol)
−160
−170
Pressure (bar)
Temperature (K)

CHAPTER 17

Release Notes

17.1 Development Branch

Development Branch

17.2 Version 1.2.19

Apr. 8, 2020

- Fixed bug where `LSR` could not be imported in `read_excel()`.
- Updated `io` sections to incorporate Pathlib library
- Added helper functions in `omkm` to organize phases
- Fixed bug where slopes and y intercepts were switched for `PiecewiseCovEffect` when writing CTI files.
- Fixed bug in `write_surf()` where `n_sites` could be written as a float
- Updated OpenMKM IO example

17.3 Version 1.2.18

Jan. 31, 2020

- Hotfix to correct broken links in documentation.

17.4 Version 1.2.17

Jan. 31, 2020

- Added more descriptive warning messages when incorrect temperature values are passed to `Nasa`, `Nasa9`, and `Shomate`.
- Fixed bug where the conversion factor for Hartrees was incorrect.
- Added extra parameters for OpenMKM IO.
- Added helper functions for OpenMKM IO to assign phases easily.
- Added a helper method in `IdealGas` and `StoichSolid` to only assign a reaction to the phase if all the species belong to that phase.
- Fixed outdated code in Chemkin example and OpenMKM example.
- Reorganized documentation to use stubs. Shorter pages should hopefully make the documentation easier to navigate.

17.5 Version 1.2.16

Dec. 9, 2019

- Hotfix to correct a typo for PyYAML version required.

17.6 Version 1.2.15

Dec. 5, 2019

- Added `write_yaml()` to write YAML files for OpenMKM.
- Added warning for `read_excel()` if the header is blank but the cells are occupied.
- Fixed bug in `read_excel()` where `model` was not correctly initialized with `StatMech()`.
- Added the generic method, `set_dict_value()` to specify dictionaries in `read_excel()`
- Removed redundant statements involving returning dictionaries in functions to process Excel data.
- Fixed warning raised whenever CpoR = 0 when fitting empirical polynomials.

17.7 Version 1.2.14

Oct. 25, 2019

- Added functionality to write files (such as `write_thermdat()`) can return a string containing the file if `filename` is not specified.
- Bug fix where `from_model` for `Nasa` and `Shomate` returned errors. The fix was related to incorrect datatyping for `misc_models`.
- Improved `Shomate` to allow users to specify the units for the polynomial coefficients.
- Energies from Gaussian input functions (`gaussian`) was originally in Hartrees. Changed to allow users to specify what unit they desire (default in eV).
- Added functionality to write BEP relationships to OpenMKM CTI files.
- Restructured OpenMKM CTI writer to be more robust when specifying custom IDs

- Added functionality to remove leading and trailing spaces when reading from Excel sheets since users found this error hard to pick up.

17.7.1 Contributors

- Qiang Li (lqcata)

17.8 Version 1.2.13

Oct. 2, 2019

- Fixed bug where small non-zero rotational inertia modes were chosen preferentially over larger contributing modes.
- Fixed bug where presets had to be specified before statistical mechanical arguments. Now, the preset will not overwrite any previously set values.
- Updated `read_thermdat()` to allow the user to return the `Nasa` objects as a list, tuple, or dictionary.
- Updated `write_thermdat()` to accept a list or a dictionary of `Nasa` objects
- Implemented *from_model* method in `Nasa` and `Shomate` classes so empirical objects can be created from `StatMech` objects as well as other empirical objects. The `from_statmech` method is deprecated.
- Added more descriptive warnings and errors.
- Created `GasPressureAdj` so entropy and Gibbs energy of gas-phase empirical objects (like `Shomate` and `Nasa`) are dependent on pressure. This object is assigned automatically to `misc_models` if `phase` is 'g' or 'gas' and the `add_gas_P_adj` can be set to False if users do not wish to assign this object automatically.
- Thermodynamic quantities of individual species can also be calculated on a per mass basis (i.e. users can calculate quantities in J/g, cal/kg, etc.). The object must contain a dictionary of its composition in `elements` for this functionality.
- Fixed broken hyperlinks.

17.8.1 Contributors

- Geun Ho Gu (googhgoo)

17.9 Version 1.2.12

Aug. 22, 2019

- Refactored `write_thermdat` so that it is simpler to understand
- Implemented `Nasa9` and `SingleNasa9` polynomials
- Added preliminary CTI file writer for Cantera and OpenMKM
- Added Binder notebooks to Examples page so users can try pMuTT before installing
- Fixed bug where `StatMech` was not passed when modes were specified indivudally in spreadsheets.

17.9.1 Contributors

Xenhua Zhang (xenhua)

17.10 Version 1.2.11

Jun. 18, 2019

- Added xlrd dependency so spreadsheets can be read using pandas
- Updated documentation page with NAM 2019 instructions.

17.11 Version 1.2.10

Jun. 13, 2019

- Another hotfix to fix a bug where the version was not incremented correctly

17.12 Version 1.2.9

Jun. 13, 2019

- Hotfix where pypi created the folder in the old case (pMuTT) instead of lower case (pmutt)

17.13 Version 1.2.8

Jun. 13, 2019

- Importing from pMuTT is now all in lowercase. (i.e. `import pmutt` instead of `import pMuTT`)

17.14 Version 1.2.7

Jun. 11, 2019

- Added documentation page for more verbose installation instructions.
- Updated `network` to use graph theory approach using states as nodes
- Bug fix for `LSR` to handle inputs that are not pmutt model objects
- Added ability to create interactive plots with Pygal
- Updated `GroundStateElec` to read `potentialenergy` from inputted `Atoms` object.

17.15 Version 1.2.6

Apr. 26, 2019

- Moved `references` attribute from empirical classes to `StatMech`
- Changed `mix_models` attribute to `misc_models` in indicating any model object can be used
- Implemented `DebyeVib` and `ConstantMode` classes
- Restructured `BEP` object to act as a transition state species in `Reaction` objects
- Implemented `LSR` object
- Added option to calculate pre-exponential factor using ratio of partition functions or entropy of activation
- Added option to use electronic energy as descriptor for `BEP` object
- Added some imperial unit functionality to `pmutt.constants` module
- Renamed `from_` parameter and `to` parameter in `pmutt.constants.convert_unit()` to `initial` and `final`
- Added ability to import individual translational, rotational, vibrational, electronic and nuclear modes to Excel
- Renamed `pmutt.statmech.trans.IdealTrans` to `FreeTrans`
- Renamed `pmutt.statmech.elec.IdealElec` to `GroundStateElec`
- Renamed `pmutt.statmech.nucl.IdealNucl` to `EmptyNucl`

17.16 Version 1.2.5

Mar. 21, 2019

- Renamed `pmutt.io_` module to `pmutt.io`
- Renamed `pmutt.io_.jsonio` module to `pmutt.io.json`
- Added preliminary IO support for MongoDB in module: `pmutt.io.db`
- Bug fixes for Chemkin IO behavior

17.17 Version 1.2.4

Mar. 11, 2019

- Hotfix to correct Chemkin IO behavior

17.18 Version 1.2.3

Feb. 25, 2019

- Added `smiles` attribute to `StatMech` and `EmpiricalBase` classes
- Added functions to write Chemkin surf.inp, gas.inp, and EAs.inp files
- Added `CovEffect` class to model coverage effects and integrated it with `StatMech` and `EmpiricalBase` classes

- Added `include_ZPE` parameter to `get_EoRT`, `get_E`, `get_delta_EoRT` and `get_delta_E` for the `StatMech` class and `Reaction` class to add zero-point energy in calculations
- Renamed private methods `_get_delta_quantity` and `_get_state_quantity` to public methods `get_delta_quantity` and `get_state_quantity` in `Reaction` class
- Added generic method `get_quantity` to `StatMech` class so any method can be evaluated. It takes the parameters `raise_error` and `raise_warning` so the user has the ability to ignore modes if they do not have the desired properties
- Added `plot_coordinate_diagram` method to the `Reactions` class to plot coordinate diagrams.
- Added `get_EoRT` and `get_E` methods to `StatMech` class to calculate electronic contribution to thermodynamic properties
- Added `get_EoRT_state` and `get_delta_EoRT` methods to `Reaction` to calculate electronic contribution to reaction properties
- Added an optional parameter, `activation`, to `get_delta_X` methods to specify the difference between the reactants/products and the transition state.
- Added `pmutt.constants.symmetry_dict` to allow easy look up of common symmetry numbers
- Fixed bug where specie-specific arguments were not passed correctly for `Reaction` class

17.19 Version 1.2.2

Jan. 18, 2019

- Added option to extract imaginary frequencies from VASP's OUTCAR files
- Added support for imaginary frequencies for `HarmonicVib` and `QRRHOVib` classes
- Restructured `HarmonicVib` and `QRRHOVib` classes to calculate vibrational temperatures, scaled wavenumbers and scaled inertia when methods are called (rather than at initialization) to prevent incorrect calculations due to changes in the vibrational wavenumbers.
- Fixed unit test names
- Added `get_species` to `Reaction` and `Reactions`
- Fixed bug related to `References` and `Reference` objects not JSON-write compatible.
- Fixed bug related to referencing in `Shomate` class

17.20 Version 1.2.1

Dec. 17, 2018

- Added `vib_outcar` special rule for `read_excel()` and `set_vib_wavenumbers_from_outcar()` to get vibrational frequencies directly from VASP's OUTCAR file.
- Added `get_X` methods to `Nasa`, `Shomate`, `StatMech` and `Reaction` to directly calculate thermodynamic properties (such as H, S, F, G) with the appropriate units
- Changed symbol for Hemlholtz energy from A to F

17.20.1 Contributors

- Himaghna Bhattacharjee (himaghna)

17.21 Version 1.2.0

Dec. 12, 2018

- Restructured code to exclude `model` module

17.22 Version 1.1.3

Dec. 11, 2018

- Added `BEP` class
- Restructured `Reaction` class so reaction states (i.e. reactants, products, transition states) can be calculated separately
- Updated `References` class to be able reference any attribute
- Added `placeholder` entry to `presets` dictionary to represent an empty species
- Added correction factor to calculate partition coefficient, q, in `IdealElec` class

17.23 Version 1.1.2

Nov. 27, 2018

- Fixed bugs in `Reaction` class for calculating pre-exponential factors
- Added methods in `Reaction` class to calculate rate constants and activation energy (currently, this just calculates the difference in enthalpy between the reactant/product and the transition state)
- Quality of life improvements such as allowing `Reaction` class inputs to be a single pmutt object instead of expecting a list

17.24 Version 1.1.1

Nov. 7, 2018

- Fixed bugs in `Shomate` class for `get_HoRT` and `get_SoR` where one temperature would return a 1x1 vector instead of a float
- Fixed bug in `Zacros` class where it expected vibrational energies instead of wavenumbers.

17.25 Version 1.1.0

Oct. 26, 2018

- Updated `Reaction` class to parse strings

- New `Shomate` class
- New equation of state classes: `IdealGasEOS`, `vanDerWaalsEOS`
- New `PhaseDiagram` class
- New `EinsteinVib` class
- New `read_reactions()` function to read species and reactions from Chemkin surf.inp and gas.inp files

CHAPTER 18

Indices and tables

- genindex
- modindex
- search