# unary-calculations

February 18, 2020

# 1 MatSE 410 pycalphad examples

**pycalphad** is a free and open source software written in Python for performing thermodynamic calculations within the CALculation of PHAse Diagrams (CALPHAD) method.

This guide was contributed to by

- Brandon Bocklund (2019, 2020)
- Matthew Feurer (2019)

## 1.1 Outline

## 1.2 CALPHAD

- Semi-empirical
- Gibbs energies are fit for each phase (fcc, bcc, liquid, gas)
- Piecewise polynomials (in temperature) of the same Gibbs energy we've discussed in class

## 1.3 Installation

### 1.3.1 Installing Anaconda

Anaconda is a distribution of the Python programming language and common packages that gives you the tools you need to run and write Python code. This course will only cover the basics of Python needed to do calculations in pycalphad. There are many sources online for learning to write code in python and I encourage you to look at these, and fimiliarize yourself with the language.

Download Anaconda here. Python 2 is not supported by packages as of January 1, 2020. We **strongly** reccommend downloading Python 3.7.

**Make sure that you select the option to add anaconda to your path variable when installing**

To test that everything installed correctly (Windows): 1. Go to your start menu. You should see several new applications installed 2. Open up the application “Anaconda Prompt”, which should open a command window. 3. Try typing “conda -h” which should print to the screen a help menu of commands for conda.

### 1.3.2 Installing pycalphad

If conda is already installed or you have just completed the installation of Anaconda it is time to install pycalphad. Doing so is easy and fast!

To install (Windows): 1. Go to your start menu and open up the application titled, “Anaconda Prompt”. This should open up a command prompt window. 2. Copy and paste to run the following command (Right click to paste into the Anaconda Prompt):

```
conda install -c pycalphad -c conda-forge pycalphad jupyter
```

3. pycalphad is now installed

See the pycalphad installation instructions for the most up to date instructions.

## 1.4 Introduction to pycalphad

This tutorial is being run from the Jupyter Lab program in the browser. You can run Jupyter Lab by running `jupyter lab` in the Anaconda Prompt.

In Python you need to import the tools that we use to preform our code. They act as additions to the basic Python language. In general you can copy them as you see here without any modification. At the begining of any new session you must run these import statements to get all the tools we need. if the command below exicutes without any errors, your installation of pycalphad is succesfull.

### 1.4.1 Databases

Any calculation in pycalphad requires you to define:

- A Database
- The components to consider
- The phases to consider
- The conditions of the calculation

CALPHAD database files are made by researchers who model the Gibbs energy functions. The parameters (Gibbs energy functions) in the databases are stored in the Thermo-Calc DataBase (TDB) format. Files are usually saved with the file extension `.tdb`. These files are read by pycalphad by importing the `Database` class and running the following

```
from pycalphad import Database
db = Database("path/to/database.tdb")
```

Many database files have been indexed at the TDBDB, which can be used to search for TDB files. Many binary and ternary systems have been evaluated, some more than once as new data or calculations are published.

Some databases are included here (`Zn-ssub5.tdb`) and (`sgte-unary.tdb`), which contain parameters for Zn solid/liquid/gas and parameters for many pure elements in the liquid and various solid phases (e.g. fcc, bcc, hcp; even if that phase is not stable), respectively.

## 1.5 One component systems

Here we first import some functions that will be used in these examples, then load the SGTE unary TDB and calculate some properties of Al vs. temperature.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from pycalphad import Database, calculate, equilibrium, variables as v
from PT import PT_phase_diagram
```

[2]:
```
# Load the database
db_sgte = Database('sgte-unary.tdb')
```

The SGTE database has many elements and phases, which can be listed.

**Elements** Note there are two special “elements”: `/-` and `VA`. `/-` is used in databases to represent a unit of negative charge, while `VA` is a dummy species that means vacancy.

**Note that many databases require `VA` are included in the sublattice models, so you should get in the habit of included them in the lists of components manually**

[3]:
```
print(sorted(db_sgte.elements))
```

```
['/-', 'AG', 'AL', 'AM', 'AS', 'AU', 'B', 'BA', 'BE', 'BI', 'C', 'CA', 'CD',
'CE', 'CO', 'CR', 'CS', 'CU', 'DY', 'ER', 'EU', 'FE', 'GA', 'GD', 'GE', 'HF',
'HG', 'HO', 'IN', 'IR', 'K', 'LA', 'LI', 'LU', 'MG', 'MN', 'MO', 'N', 'NA',
'NB', 'ND', 'NI', 'NP', 'O', 'OS', 'P', 'PA', 'PB', 'PD', 'PR', 'PT', 'PU',
'RB', 'RE', 'RH', 'RU', 'S', 'SB', 'SC', 'SE', 'SI', 'SM', 'SN', 'SR', 'TA',
'TB', 'TC', 'TE', 'TH', 'TI', 'TL', 'TM', 'U', 'V', 'VA', 'W', 'Y', 'YB', 'ZN',
'ZR']
```

**Phases** Many of the phases you know have some symbols after the underscore in the phase name, for example `FCC_A1` has `A1`. These are called *Strukturbericht* symbols. The letter sometimes has meaning (e.g. `A` for pure elements), but the number has almost no meaning and usually the numbers roughly correspond to the order in which they were discovered/named.

For CALPHAD calculations, it’s just a convention.

A list of many phases and Strukturbericht symbols can be found here.

[4]:
```
print(sorted(db_sgte.phases))
```

```
['ALPHA_PU', 'ALPHA_RHOMBO_B', 'BCC_A2', 'BCT_A5', 'BCT_AA', 'BETA_PU',
'BETA_RHOMBO_B', 'CBCC_A12', 'CUB_A13', 'DHCP', 'DIAMOND_A4', 'FCC_A1',
'GAMMA_PU', 'GAS', 'GRAPHITE', 'HCP_A3', 'HCP_ZN', 'HEXAGONAL_A8', 'LIQUID',
'MONOCLINIC', 'OMEGA', 'ORTHORHOMBIC_A20', 'ORTHORHOMBIC_GA', 'ORTHORHOMBIC_S',
'ORTHO_AC', 'RED_P', 'RHOMBOHEDRAL_A7', 'RHOMBO_A10', 'RHOMB_C19',
'TETRAGONAL_A6', 'TETRAGONAL_U', 'TETRAG_AD', 'TET_ALPHA1', 'WHITE_P']
```

### 1.5.1 `calculate()` and `equilibrium()`

The main two functions for computing things with pycalphad are `calculate()` and `equilibrium()`. There is a subtle difference between these that may not be very clear initially.

- `calculate()` computes a property of a phase for given natural variables for Gibbs energy ($P$, $T$, and $N$)
- `equilibrium()` finds the phases and phase fractions that give the global minimum Gibbs energy constrained under conditions of $P$, $T$, and $x_i$. It uses pycalphad to numerically sample the Gibbs energies of phases that could form under the given conditions.

Whenever we want to see a property of a phase, regardless of whether or not it's stable, we use `calculate()`.

Whenever we want to know the experimental/equilibrium properties, i.e. which phases are stable, we use `equilibrium()`.

The inputs of each are slightly different, in that `calculate()` takes state variables as keyword arguments, while `equilibrium()` takes the conditions as a dictionary. If you don't know what this means, the examples will help clarify.

### 1.5.2 State variables

In any thermodynamic calculation, $c+2$ conditions must be defined for $c$ independent components. Note that `VA` is not an independent component.

The different state variables are accessed by prepending `v.<StateVariable>`:

- **Pressure:** `v.P` in Pascal. Standard pressure is 101325 Pa.
- **Moles:** `v.N` only use `N=1`.
- **Temperature:** `v.T` in Kelvin. Most databases are only defined between 298.15 K and 6000 K, so you should keep calculations in this range.
- **Composition:** `v.X('NI')` as a mole fraction (for Ni).

### 1.5.3 Gibbs energy of Al

First, calculate the Gibbs energy of Al for all phases over all temperatures. Since we want to see the energies for phases even if they are not stable, we'll use `calculate()`.

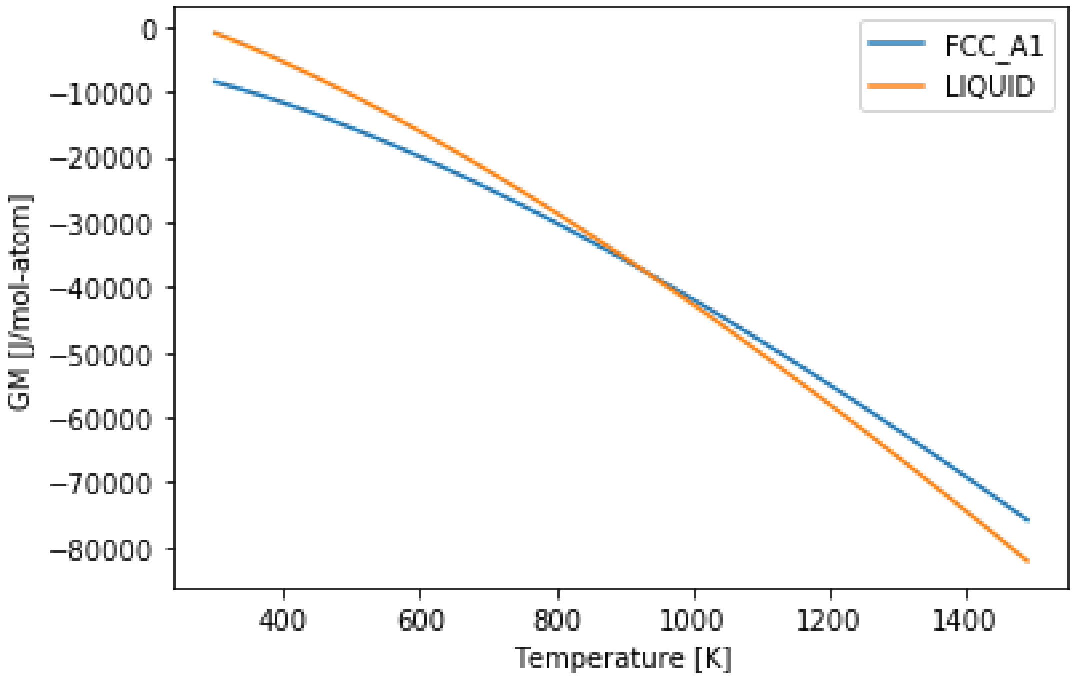
Ranges of state variables can be defined using a `(start, stop, step)` format, e.g. `T=(300, 1500, 10)` goes from 300 K to 1500 K by intervals of 10 K.

**Remember to include `VA` as a component, since it is used by the FCC_A1 phase.**

```
[5]: fcc_GM = calculate(db_sgte, ['AL', 'VA'], 'FCC_A1', P=101325, T=(300, 1500,␣
      ↪10), N=1)
     liquid_GM = calculate(db_sgte, ['AL', 'VA'], 'LIQUID', P=101325, T=(300, 1500,␣
      ↪10), N=1)

     # Squeeze is required because the array corresponding to GM is multidimensional␣
      ↪for each T, P, and N where we've calculated the energy
     plt.plot(fcc_GM.T, fcc_GM.GM.squeeze(), label='FCC_A1')
     plt.plot(liquid_GM.T, liquid_GM.GM.squeeze(), label='LIQUID')
     plt.xlabel("Temperature [K]")
     plt.ylabel("GM [J/mol-atom]")
     plt.legend()
```

```
[5]: <matplotlib.legend.Legend at 0x1243acb70>
```



### 1.5.4 Equilibrium enthalpy of Zn

Now we want to see the enthalpy change as a function of temperature, and see that the enthalpy is discontinuous at the two phase equilibrium transformation temperature, according to Gibbs phase rule.

To be able to see the enthalpy, we have to provide an `output` argument to equilibrium to ask for the `HM` (molar enthalpy). The list of quantities you can ask for are:

```
Molar enthalpy (HM)        'HM'
```

```
Mixing enthalpy            'HM_MIX'
Molar Gibbs energy         'GM'
Gibbs energy of mixing     'GM_MIX'
Molar Entropy              'SM'
Molar Heat capacity        'CPM'
Degree of Ordering         'DOO'
```

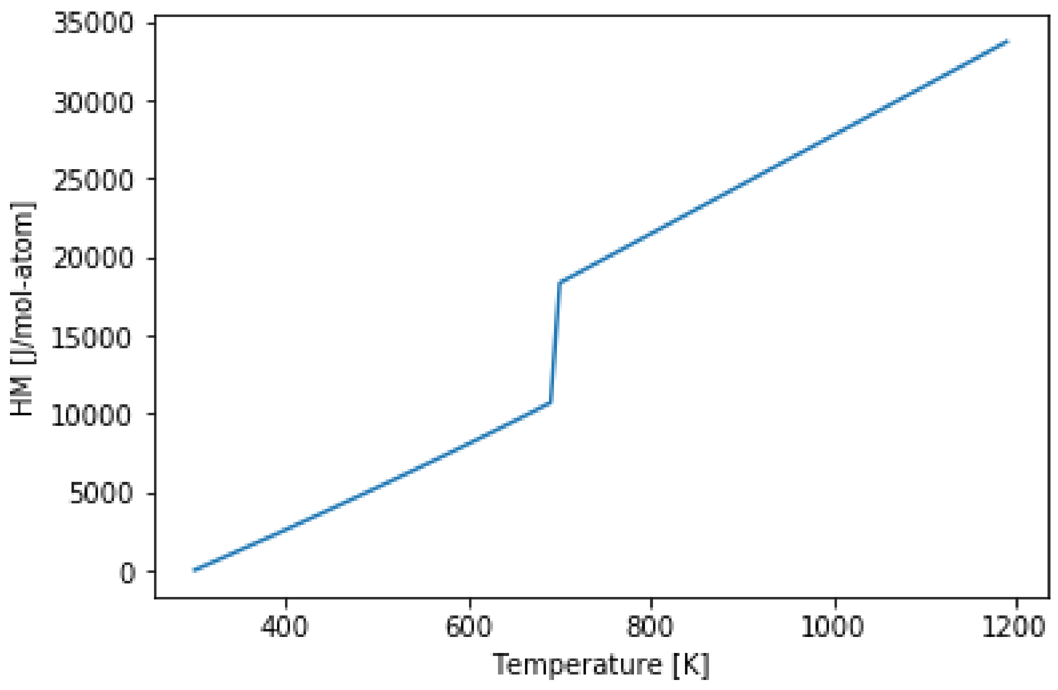
For now, only consider the solid and liquid phases.

[6]:
```
db_zn = Database('Zn-ssub5.tdb')
```

[7]:
```
conditions = {  # mapping of state variable to values
    v.N: 1,
    v.P: 101325,
    # for now, only temperature has a range. If multiple variables have ranges,␣
 ↪all combinations are calculated
    v.T: (300, 1200, 10),
}
equil_result = equilibrium(db_zn, ['ZN'], ['SOLID', 'LIQUID'], conditions,␣
 ↪output='HM')

# Squeeze is required because the array corresponding to HM is multidimensional␣
 ↪for each T, P, and N where we've calculated the energy
plt.plot(equil_result.T, equil_result.HM.squeeze())
plt.xlabel("Temperature [K]")
plt.ylabel("HM [J/mol-atom]")
```

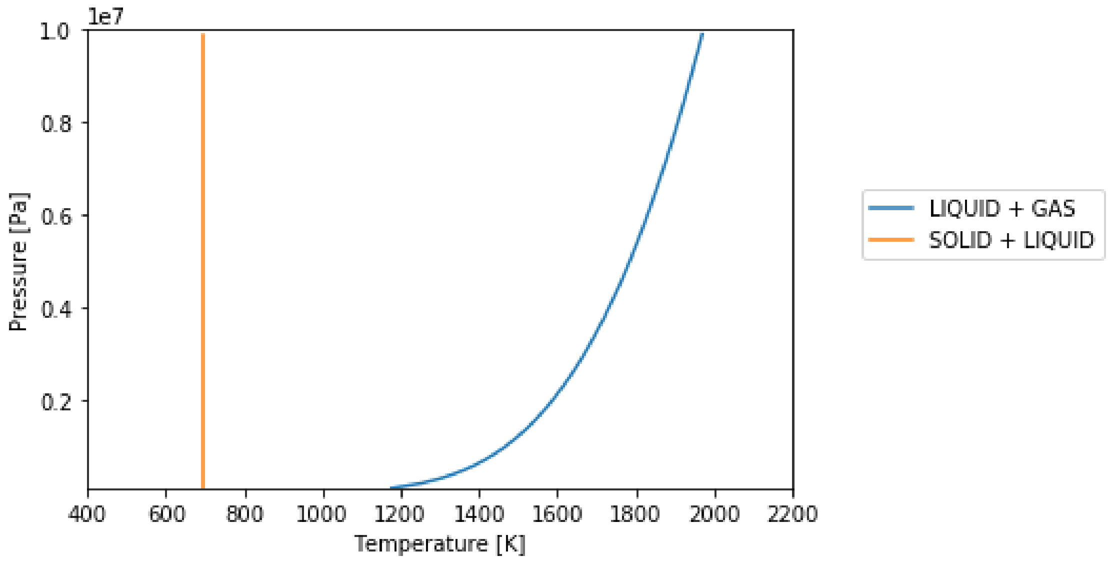
[7]:
```
Text(0, 0.5, 'HM [J/mol-atom]')
```



## 1.6 Pressure Temperature phase diagram

The `PT_phase_diagram` function is written for you to use `calculate()` to determine the PT diagram. here's one for ZN:

```
[8]: PT_phase_diagram(db_zn, ['ZN'], ['GAS', 'SOLID', 'LIQUID'], {v.N: 1, v.P: (1e5,␣
      ↪1e7, 1e5), v.T: (400, 2200, 1)})
```


1e7
Pressure [Pa]
Temperature [K]
LIQUID + GAS
SOLID + LIQUID