OpenCalphad Python

Release 0.0.1

Chunhui Luo

Oct 09, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

Open Calphad (OC) is an open-source computational tool of performing thermodynamic calculations for all kinds of applications. OC is written in module structure using modern Fortran (F90) language. It can be operated using two modes: Console and TQ interface.

TQ interface is an application program interface (API) which is developed to FORTRAN and C language users. If you use FORTRAN or C++ , you could find some examples from OpenCalphad GitHub (https://github.com/sundmanbo/opencalphad/tree/master/examples/TQ4lib).

OC-Python is a Python interface, which allows the Python users within the thermodynamical computing field access to OpenCalphad functionality without the requirement of learning Fortran. It is prepared for those who would like to develop own python application codes by coupling OpenCalphad. Of course, it might be a help tool for Integrated Computational Materials Engineering (ICME).

There are some solutions available from GitHub (Shengyen Li, Teslos, Clément Introïni) to use OpenCalphad in a Python code.

OC-Python is built by using F2PY and F90wrap tools for the application module "liboctq.f90". It can be used under Windows / Linux.

F2PY is a tool that provides an easy connection between Python and Fortran languages. F2PY is part of NumPy. F2PY can build the generated extension modules as a library which be imported and used directly in python code.

f90wrap is a tool to automatically generate Python extension modules which interface to Fortran code that makes use of derived types.

Figure 1 show the application structure diagram for Opencalphad. Only subroutines in the TQ library (Opencalphad API) are used to access the Opencalphad Engine (main code), which are invoked by different application programs (Fortran, C++, Python).

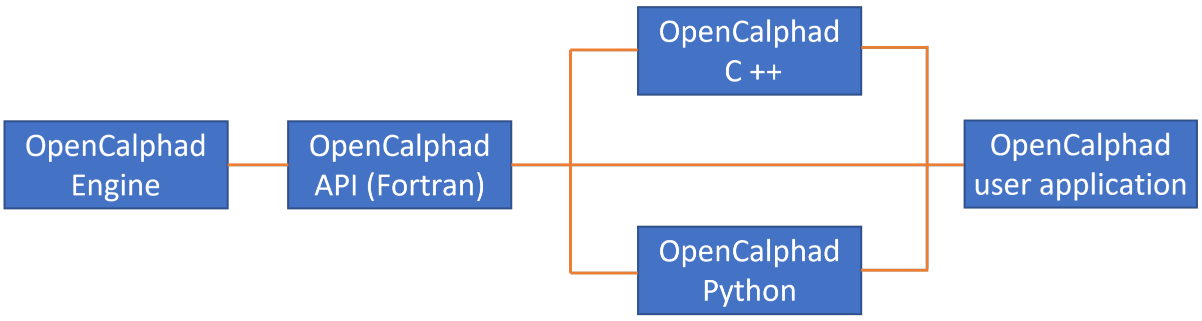

Fig. 1: Figure 1 Application structure diagram for Opencalphad

GPL license is used for OC-Python.

CHAPTER TWO

INSTALLATION

2.1 Prerequisite

For Windows 10:

python 3.7, 3.8, 3.9

For Linux (only tested for Ubuntu 20.04):

python 3.8

numpy >= 1.18

2.2 Installation of OC-Python (using python wheels)

Windows (open the Command Prompt as Administrator):

```
pip install ocpython-0.0.1-py37-none-win_amd64.whl

pip install ocpython-0.0.1-py38-none-win_amd64.whl

pip install ocpython-0.0.1-py39-none-win_amd64.whl
```

Linux:

```
pip install ocpython-0.0.1-py38-none-linux_x86_64.whl
```

2.3 Uninstall of OC-Python

Windows / Linux:

```
pip uninstall ocpython
```

2.4 Check current version of ocpython installed on your computer:

```
pip show ocpython
```

CHAPTER

THREE

BUILD AND USE OF OC-PYTHON

3.1 Build of OC-Python library

OC-Python library is built with following steps:

1. Building OC TQ

Download OC package and run “linkmake” (adding extension ‘.cmd’ first)

2. Compile liboctq.f90 using gfortan compiler

Create liboctq.mod and liboctq.o

3. Wrap f90 files and produce wrappers suitable for input to f2py use

4. Build python library using f2py

liboctq_f90wrap.cp37-win_amd64.pyd for Windows and Python 3.7

liboctq_f90wrap.cp38-win_amd64.pyd for Windows and Python 3.8

liboctq_f90wrap.cp39-win_amd64.pyd for Windows and Python 3.9

liboctq_f90wrap.cpython-38-x86_64-linux-gnu.so for Linux and Python 3.8

5. Create a PyPi ready pack

ocpython-0.0.1-py37-none-win_amd64.whl for Windows and Python 3.7

ocpython-0.0.1-py38-none-win_amd64.whl for Windows and Python 3.8

ocpython-0.0.1-py39-none-win_amd64.whl for Windows and Python 3.9

ocpython-0.0.1-py38-none-linux_x86_64.whl for Linux (only tested for Ubuntu 20.04)

Subroutines or functions in the wrapped liboctq.f90:

minimal:

eq = f90wrap_pytqini(n)
f90wrap_pytqrfil(filename,eq)
f90wrap_pytqrpfil(filename,nsel,selel,eq)
f90wrap_pytqce(target,n1,n2,value,eq)
f90wrap_pytqgetv(stavar,n1,n2,n3,values,eq)

useful:

errorcode = f90wrap_pygeterr()
f90wrap_pyseterr(errorcode)
f90wrap_pytqcecompact(filename,nsel,massunit,selel,tpn,xi,xf,phnames1,elref,phref,eq)
f90wrap_pytqcompbatch(nsel,nxfrac,xi,xfrac,temp,stavar,values,eq)
f90wrap_pytqtempbatch(nsel,ntemp,xi,xfrac,temp,stavar,values,eq)

```
n,comp = f90wrap_pytqgcom(eq)
n = f90wrap_pytqgnp(eq)
phasename = f90wrap_pytqgpn(phtupx,eq)
phtupx = f90wrap_pytqgpi(phasename,eq)
iph,ics = f90wrap_pytqgpi2(phasename,eq)
phases = f90wrap_pytqgpsm(n,status1,amdgm,eq)
csname = f90wrap_pytqgpcn2(n,c)
nspel,smass,qsp = f90wrap_pytqgpcs(c,ielno,stoi)
f90wrap_pytqphsts(phtupx,newstat,val,eq)
f90wrap_pytqphsts2(phnames,newstat,val,eq)
cnum = f90wrap_pytqsetc(stavar,n1,n2,value,eq)
f90wrap_pytqtgsw(i)
nsub = f90wrap_pytqgphc1(iph,cinsub,spix,yfrac,sites,extra,eq)
f90wrap_pytqsphc1(n1,yfra,extra,eq)
f90wrap_pytqcph1(n1,n2,n3,gtp,dgdy,d2gdydt,d2gdydp,d2gdy2,eq)
f90wrap_pytqcph2(n1,n2,n3,n4,eq)
f90wrap_pytqcph3(n1,n2,g,eq)
f90wrap_pytqdceq(name)
n1,neweq = f90wrap_pytqcceq(name,eq)
eq = f90wrap_pytqselceq(name)
f90wrap_pytqcref(ciel,phase,tpref,eq)
f90wrap_pytqlr(lut,eq)
f90wrap_pytqlr1(lut,eq)
f90wrap_pytqlc(lut,eq)
f90wrap_pytqltdb()
f90wrap_pytqquiet(yes)
```

3.2 State variables in Opencalphad

Thermodynamics variables describe the state of the system at equilibrium. These variables can be divided into two types: intensive variables and extensive variables. Pressure and temperature are two important intensive variables. Please refer to Section 2.6 in Opencalphad's help manual for details. In this user guide (OC-Python), two kinds of tables are given based on the need due to programming.

The state variables in Table 1 are associated with component. All state variables in this table are used for setting condition and getting result.

Table 1 State variables associated with component

Name	Symbol	Note
Amount of moles of element	**N**	For element
Amount of mass of element	**B**	For element
Mole fraction of element	**X**	For element
Mass fraction of element	**W**	For element
Chemical potential	**MU**	For element
Activity	**AC**	For element
Log of activity	**LNAC**	For element

The state variables in Table 2 are associated with phase. All state variables in this table are used for getting result and some of them are used for setting condition.

Table 2 State variables associated with phase

Name (symbol)	For system	For phases	For phases (per mole)	For phases (per mass)	For phases (per m^3)	Note
Gibbs energy (G)	**G**	**G**	**GM**	**GW**	**GV**	
Internal energy (U)	**U**	**U**	**UM**	**UW**	**UV**	
Entropy (S)	**S**	**S**	**SM**	**SW**	**SV**	
Enthalpy (H)	**H**	**H**	**HM**	**HW**	**HV**	
Helmholtz energy (A)	**A**	**A**	**AM**	**AW**	**AV**	
Volume (V)	**V**	**V**	**VM**	**VW**	**VV**	
Stability of phase (Q)		**Q**				
Driving force (DG)		**DG**	**DGM**	**DGW**		
Moles of phase (NP)		**NP**	**NPM**			
Mass of phase (BP)		**BP**		**BPW**		

3.3 Use of OC-Python

You can create a Python script to construct your application using directly the functions listed in the Section 3.1. It is also possible to extend the capabilities of the wrapped Fortran subroutines / functions through generating intermediate Python modules.

3.3.1 Use OC-Python via directly accessing to OC's wrapped subroutines

The user can directly access to OC's subroutines which are defined in liboctq.f90 (https://github.com/sundmanbo/opencalphad/blob/master/examples/TQ4lib/F90/liboctq.F90). In OC-Python package, these subroutines are re-defined using "f90wrap_py" prefix.

Table 1: *Table 3: Redefinition of subroutine names*

Code Name	liboctq.f90	f90wrap_pytq.f90
Subroutine example 1	tqini	f90wrap_pytqini
Subroutine example 2	tqce	90wrap_pytqce

Invoke subroutine "tqini" in TQ fortran code:

call tqini(n,ceq)

Invoke wrapped subroutine "tqini" in Python code:

oc.f90wrap_pytqini(n)

This kind of invoke is called as low-level use of OC-Python. The user has to know the definitions of input and output for each Fortran subroutine in liboctq.f90. The advantage is that you can program your code freely, which is similar to the programming in OC TQ.

Table 2: *Table 4: Minimal calls of wrapped subroutines in liboctq.f90 for a simple equilibrium calculation*

function	Python script
Import library	from ocpython import liboctq_f90wrap as oc
Initiation	eq = oc.f90wrap_pytqini(n)
Read tdb	oc.f90wrap_pytqrpfil(tdbFile,n_elem,element_str,eq)
Set temperature condition	oc.f90wrap_pytqsetc('T',0,0,temperature,eq)
Set pressure condition	oc.f90wrap_pytqsetc('P',0,0,pressure,eq)
Set molar number	oc.f90wrap_pytqsetc('N',0,0,pressure,eq)
Set composition condition	oc.f90wrap_pytqsetc('X',i+1,0,xcomp,eq)
Calculate equilibrium	oc.f90wrap_pytqce('',gridMinimizerStatus,0,0.0,eq)
Get result	oc.f90wrap_pytqgetv('G',0,0,1,value,self.eq)

3.3.2 Use OC-Python via intermediate Python modules

The low-level use of wrapped subroutines in OC-Python is not so efficient in programming. A pre-programmed python code, which invokes several low-level subroutines from wrapped Fortran module, can be used instead, leading to a so-called intermediate-level use of wrapped subroutines in OC-Python. For example, "OCPython.py" is a collection of intermediate python codes. It is extensible and users can further add more functions.

Two python scripts are available in the OCPython package: 1) "OCPython.py", is main part of OC-Python 2) "OCPython_utility.py", is the auxiliary part of OC-Python

Table 3: *Table 5: Minimal calls of modules in OCPython.py for a simple equilibrium calculation*

function	Python script
Import library	from ocpython.OCPython import SingleEquilibriumCalculation
Initiation	oc = SingleEquilibriumCalculation(vs)
Read tdb	oc.readtdb(tdbFile,elements)
Set temperature condition	oc.setTemperature(temperature)
Set pressure condition	oc.setPressure(1E5)
Set molar number	oc.setTotalMolarAmount(1)
Set composition condition	oc.setElementMassFraction(elementMassFractions)
Calculate equilibrium	oc.calculateEquilibrium(gmStat.On)
Get result	oc.getGibbsEnergy()

State variables in Tables 1 and 2 can be obtained using functions in OC-Python:

oc.getValuePhase(‘G’): Gibbs energy values for all phases at the current equilibrium

oc.getScalarResult(‘G’): Gibbs energy of the system at the current equilibrium

oc.getValueComponent(‘W’): mass fraction values for all components at the current equilibrium

oc.getValueComponent(‘MU’): chemical potential values for all components at the current equilibrium

In the next Sections, more examples are shown to demonstrate how build your own application using Python.

3.4 Tips for programming

To be added later.

If you have comments and suggestions on OC-Python, please send an email to MatProComp@gmail.com.

CHAPTER

FOUR

GETTING STARTED

4.1 Prerequisite

It is assumed that you have installed OC-Python library on your python environment (For Windows: Python 3.7 or 3.8 or 3.9; For Linux Ubuntu: Python 3.8).

4.2 Developing environment

It is recommended to debug/run your project (using OC-Python) under an IDE. You may choose Visual Studio Code or PyCharm.

4.3 Run example code (Jupyter: OC_get_started.ipynb)

This example helps you to get started in using OC-Python

Single equilibrium is calculated and some results are shown.

```
import json
from ocpython.OCPython import SingleEquilibriumCalculation
from ocpython.OCPython import Verbosity
from ocpython.OCPython import GridMinimizerStatus as gmStat
from ocpython.OCPython_utility import OCPython_utility
```

Initiate module and set verbosity level

```
vs = Verbosity(newlogfile=True,process_name=
↪'SingleEquilibriumCalculation')
isVerbosity = False
vs.setVerbosity(isVerbosity)

# Initiate SingleEquilibriumCalculation
oc = SingleEquilibriumCalculation(vs)
```

Read tdb database with elements

```
# reading database (.tdb file)
tdbFile=r'steel1.TDB'
elements = ['FE','C','CR']
oc.readtdb(tdbFile,elements)

# get all phases in the system
nPhase = oc.getNumberPhase()
```

(continues on next page)

(continued from previous page)

```
phases = []
for index in range(nPhase):
    phases.append(oc.getPhaseName(index))
print('list phases:',phases)
comp = oc.getComponentNames()
print('components list from database:', comp)
```

list phases: [‘LIQUID’, ‘BCC_A2’, ‘CBCC_A12’, ‘CEMENTITE’, ‘CHI_A12’, ‘CR3SI’, ‘CRSI2’, ‘CUB_A13’, ‘DIAMOND_A4’, ‘FCC_A1’, ‘FE4N’, ‘FECN_CHI’, ‘GRAPHITE’, ‘HCP_A3’, ‘KSI_CARBIDE’, ‘M23C6’, ‘M3C2’, ‘M5C2’, ‘M7C3’, ‘SIGMA’, ‘V3C2’]

components list from database: [‘C’, ‘CR’, ‘FE’]

Set conditions

```
# set pressure and temperature
oc.setPressure(1E5)
oc.setTemperature(800)
oc.setTotalMolarAmount(1)

# set element molar fraction
elementMassFractions = {
      'C' : 0.009,
      'CR' : 0.045,
      'FE': -1
      }
oc.setElementMassFraction(elementMassFractions)
```

Calculate equilibrium and list results

```
# calculate equilibrium with grid-minimizer
oc.calculateEquilibrium(gmStat.On)
print('Gibbs energy [J]: ',  "{:.2f}".format(oc.getGibbsEnergy()))

phaseElementComposition = oc.getPhaseElementComposition()
if not isVerbosity: print('Phase element composition:\n'+json.
↪dumps(json.loads(json.dumps(phaseElementComposition), parse_
↪float=lambda x: round(float(x), 6)),indent=4))

phasefraction = OCPython_utility.getPhaseFraction(oc)
if not isVerbosity: print('Phase molar fractions:\n'+json.
↪dumps(json.loads(json.dumps(phasefraction), parse_float=lambda x:␣
↪round(float(x), 6)),indent=4))
```

```
Gibbs energy [J]:  -30129.41
Phase element composition:
{
    "BCC_A2": {
        "C": 3.1e-05,
        "CR": 0.002029,
        "FE": 0.99794
    },
    "GRAPHITE": {
        "C": 1.0,
        "CR": 0.0,
        "FE": 0.0
    },
    "M7C3": {
        "C": 0.3,
        "CR": 0.435324,
        "FE": 0.264676
    }
}
Phase molar fractions:
{
    "BCC_A2": 0.887532,
    "GRAPHITE": 0.009454,
    "M7C3": 0.103014
}
```

Note: You must copy tdb database “steel1.tdb” to your location for the python code.

CHAPTER

FIVE

APPLICATION EXAMPLES

These examples are prepared to demonstrate the use of the main features and functions of OC-Python.

Only example 5.1 is related to direct access to subroutines in liboctq modules and other examples uses intermediate module OCPython.py. After equilibrium calculation, one can get more results using the state variables in Table 1 and Table 2.

5.1 Direct access to subroutines in wrapped Fortran liboctq module

Code name : OC_ex1_direct_access_liboctq.py

This example imports f90wrap_liboctq library and directly access to the subroutines in liboctq modules.

Single equilibrium calculation at temperature 1173 K is done for HSS steel with database steel1.tdb. Some basic subroutines are invoked.

5.2 Single equilibrium calculation

Code name : OC_ex2_singleEq.py

This example shows how to carry out single equilibrium using intermediate module "OCPython.py".

Some features are tested:

1) Set phase status (suspended, entered)
2) Calculate equilibrium with or without using grid minimizer
3) Compare the Gibbs energies with or without using grid minimizer
4) Get more results (chemical potential, element composition in phases, sites in phases, constituent composition)

5.3 Single equilibrium calculation with compact mode

Code name: OC_ex3_singleEqCompact_stepCalc.py

This example shows how to use intermediate module for single equilibrium calculation with compact mode and temperature step calculation.

HSS steel is used as an example.

A module "singleEquilibriumCalculation_Compact" inside class "SingleEquilibriumCalculation" is designed to integrate more function, resulting in a compact mode for the serial calculations. The corresponding subroutine "f90wrap_pytqcecompact" in liboctq.f90 is also available, including reading tdb, setting phase status, setting reference phase of element, setting conditions, performing equilibrium calculation.

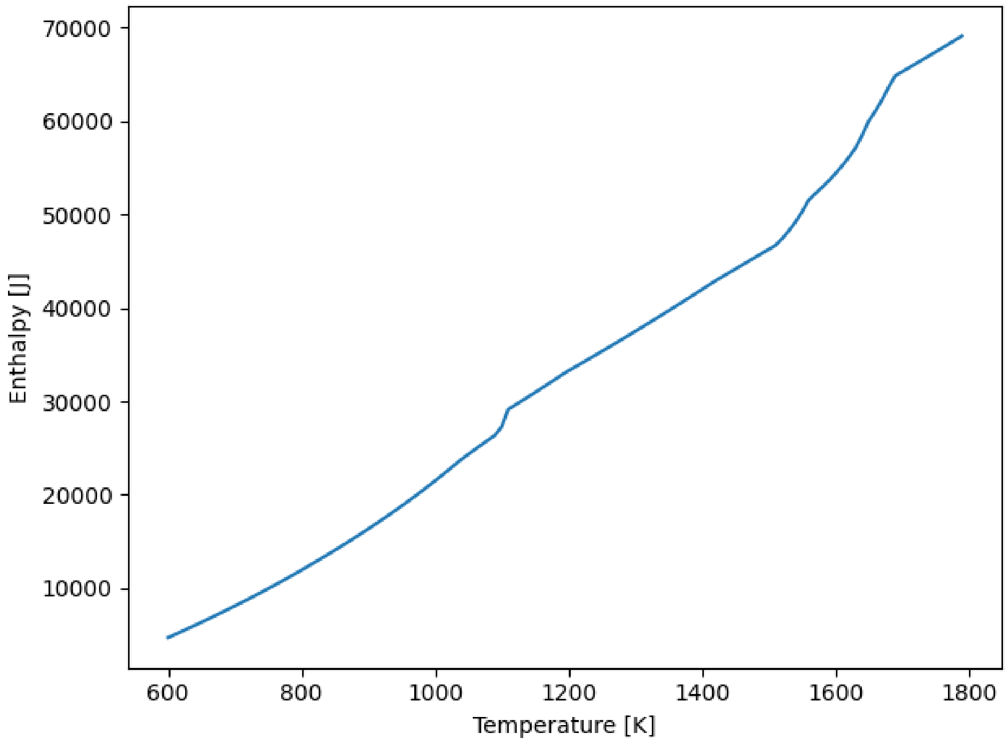

Fig. 1: Figure 2 Plot of enthalpy as a function of temperature for HSS steel

5.4 Single equilibrium calculation with compact mode plus property diagram

Code name : OC_ex4_singleEqCompact_propertyDiagram_HSS.py

This example shows how to create a property (step) diagram for HSS steel.

The step variable is temperature.

A special module is used in OCPython_utility.py to treat the possible new phase (FCC_A1#2) based on FCC_A1 matrix phase.

Code name : OC_ex4_singleEqCompact_propertyDiagram_ALCuMgZn.py

This example shows how to create a property (step) diagram for Al-Cu-Mg-Zn alloy.

The step variable is temperature.

5.5 Batch equilibria calculations with two composition variables

Code name : OC_ex5_batchEq_composition_loop.py

This example shows how to perform batch equilibria calculations and create a contour / 3D diagrams.

Composition step variable is used for batch equlibria computations.

Two composition axes are defined.

Results plotted for HSS steel:

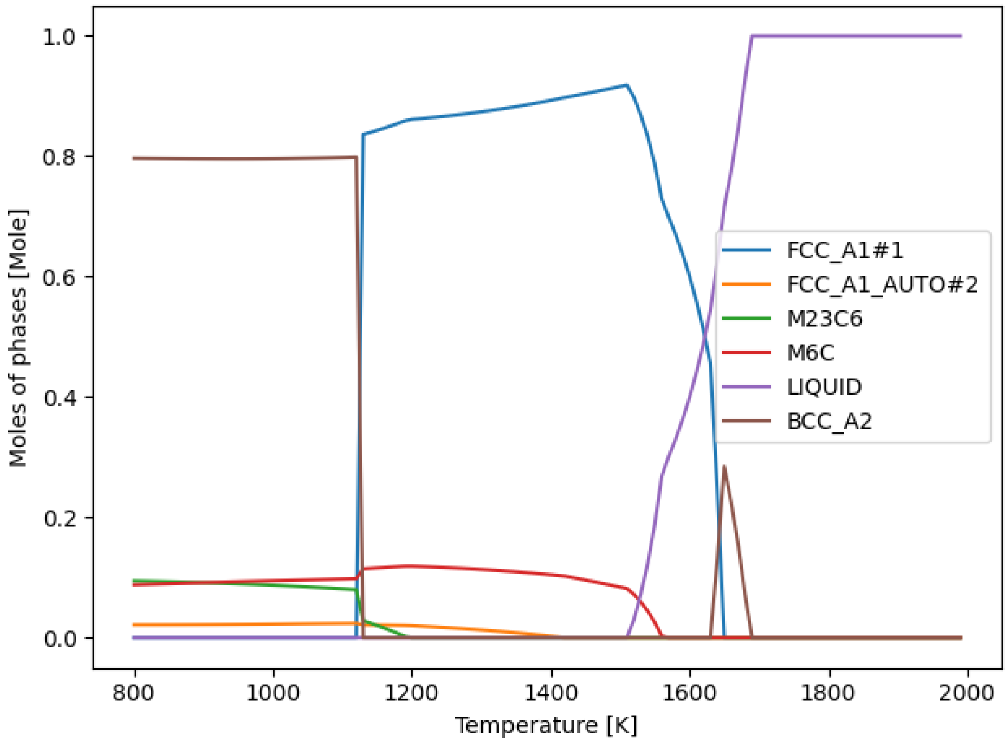

Fig. 2: Figure 3 Plot of moles of phases as a function of temperature for HSS steel

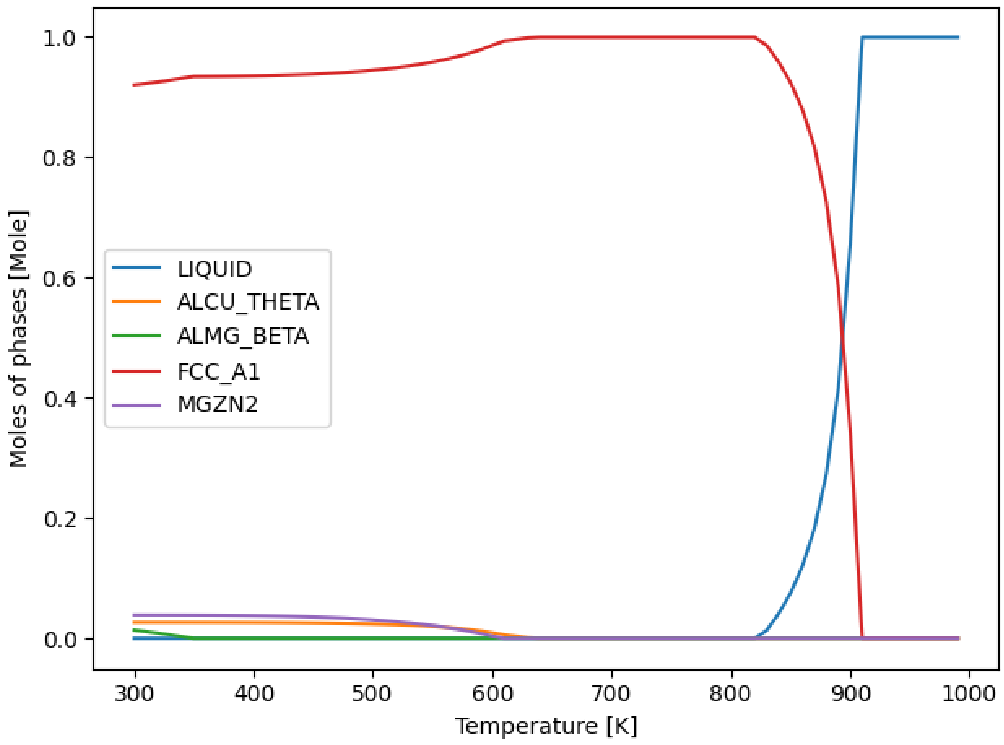

Fig. 3: Figure 4 Plot of moles of phases as a function of temperature for AlCUMgZn alloy

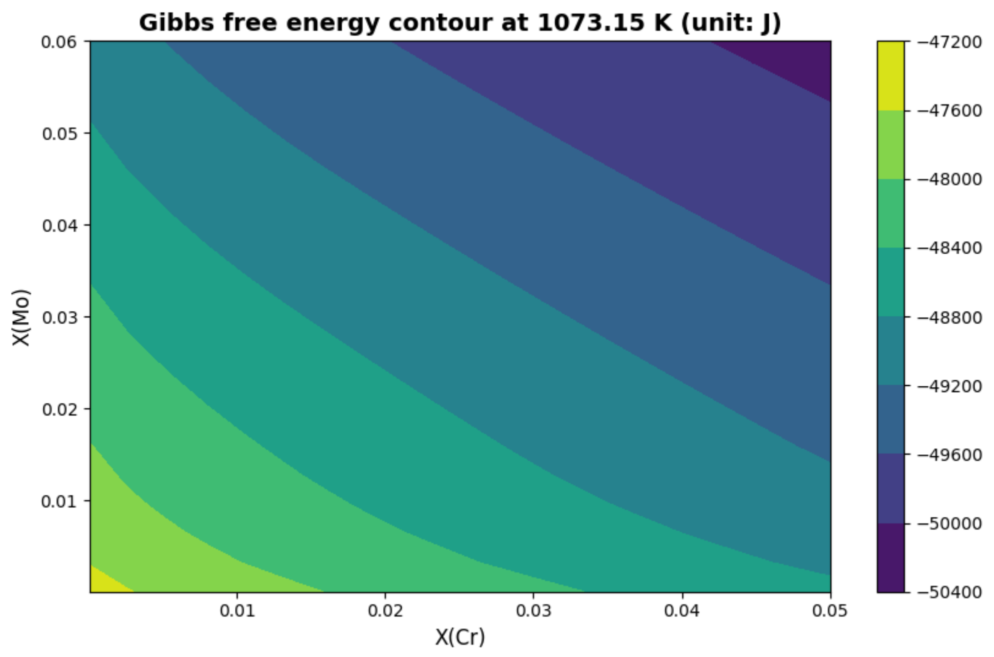

Fig. 4: Figure 5 Contour diagram of Gibbs energy vs two composition variations for HSS steel

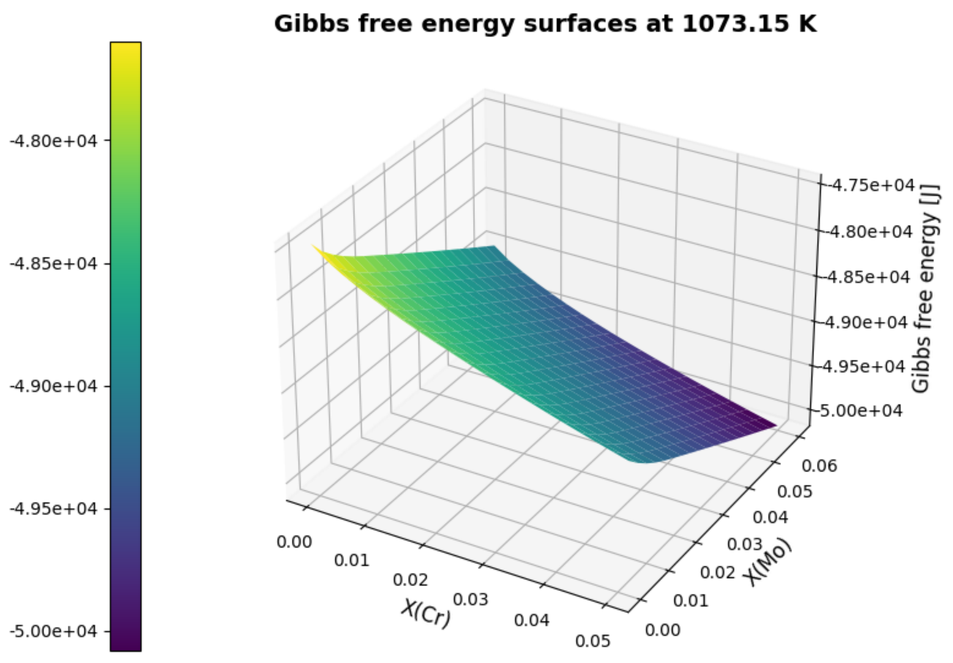

Fig. 5: Figure 6 3D Plot of Gibbs energy vs two composition variations for HSS steel

5.6 Batch equilibria calculations with composition and temperature variables

Code name : OC_ex6_batchEq_comp_temp_loop.py

This example shows how to perform batch equilibria calculations and create a contour / 3D diagrams.

One step variable is temperature, another step variable is composition.

Results plotted for Al-Mg-Zn alloy:

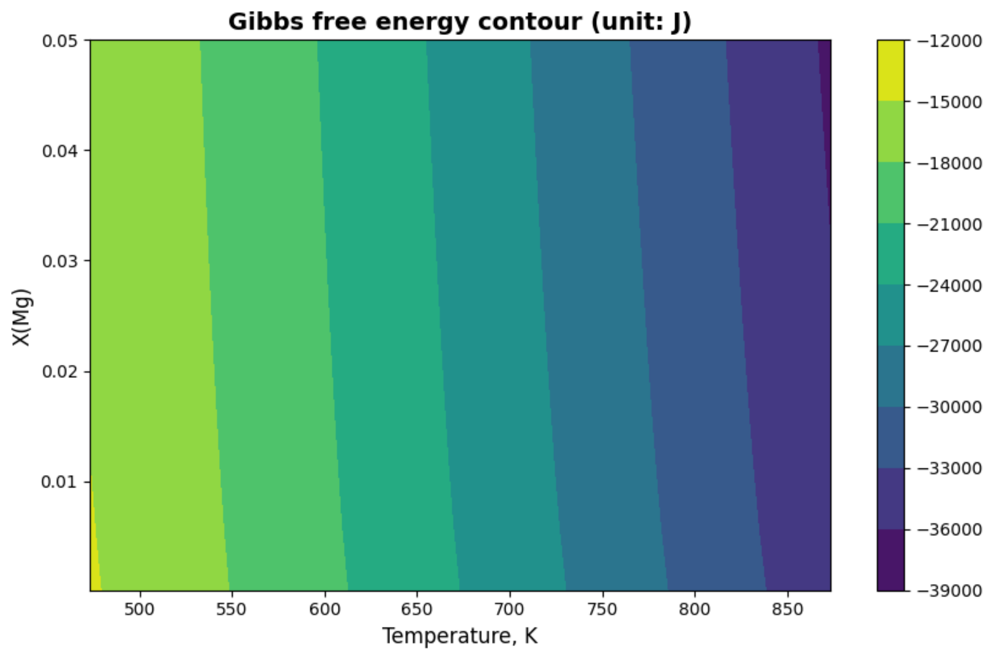

Fig. 6: Figure 7 Contour diagram of Gibbs energy vs two composition variations for Al-Mg-Zn alloy

5.7 Simple diffusion model for a ternary Al-Ni-Pt system

Code name : OC_ex7_diffusion_AlNiPt.py

This example is same as fortran example (https://github.com/sundmanbo/opencalphad/tree/master/examples/TQ4lib/F90/parallel-alnipt).

It simulates diffusion in 1D using OC Ternary system Al-Ni-Pt coating of superallys.

This python code can dynamically illustrate the evolution of compositions and chemical potentials with elapsed time.

Results plotted at time step 0:

Results plotted at time step 10000:

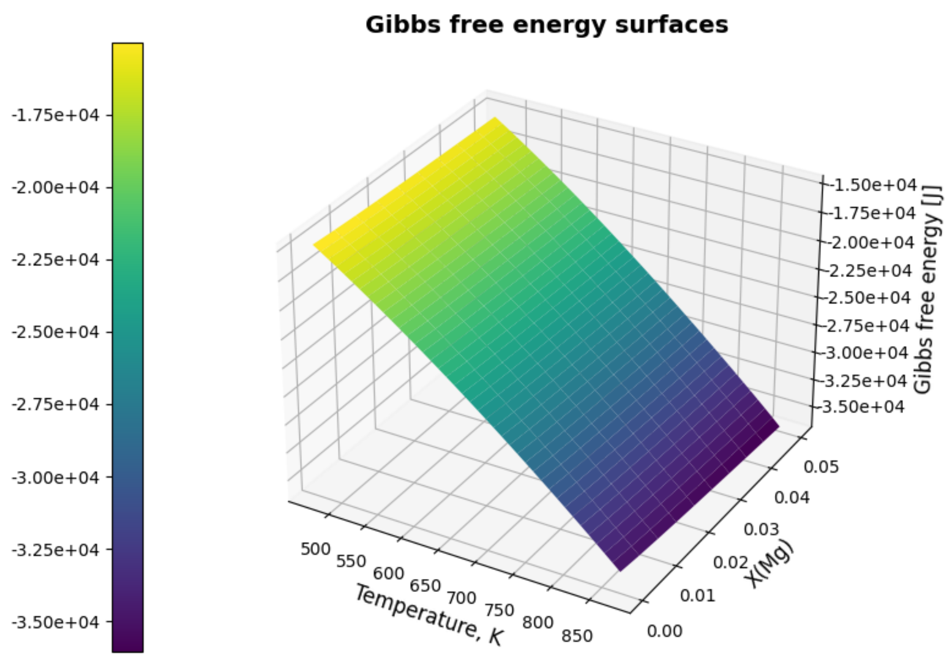

Fig. 7: Figure 8 3D Plot of Gibbs energy vs two composition variations for Al-Mg-Zn alloy

5.8 Single equilibrium calculation plus solidus and liquidus temperature

Code name : OC_ex8_singleEq_liquidus_solidus_temperature.py

This example shows how to use intermediate module for single equilibrium calculation plus liquidus and solidus temperatures. To calculate the liquidus and solidus temperatures fixed phase conditions for the liquid phase is utilized.

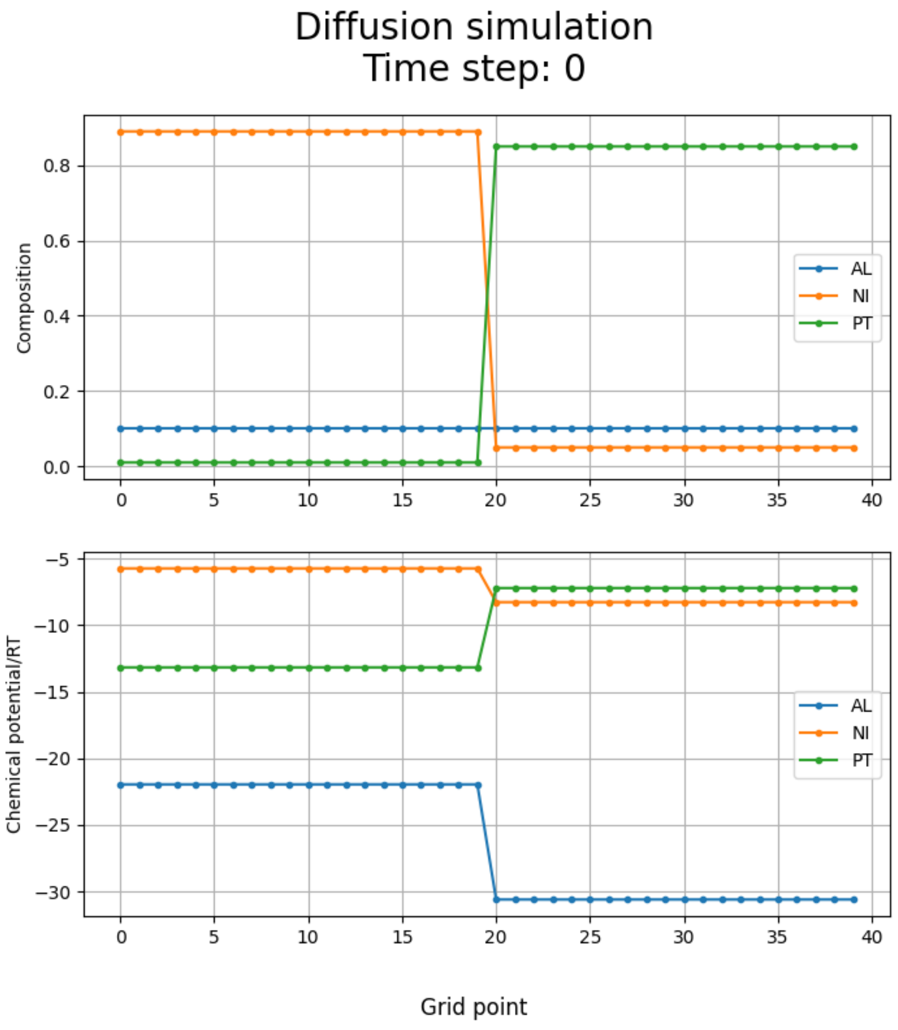

Fig. 8: Figure 9 Plot of initial composition profile and chemical potential

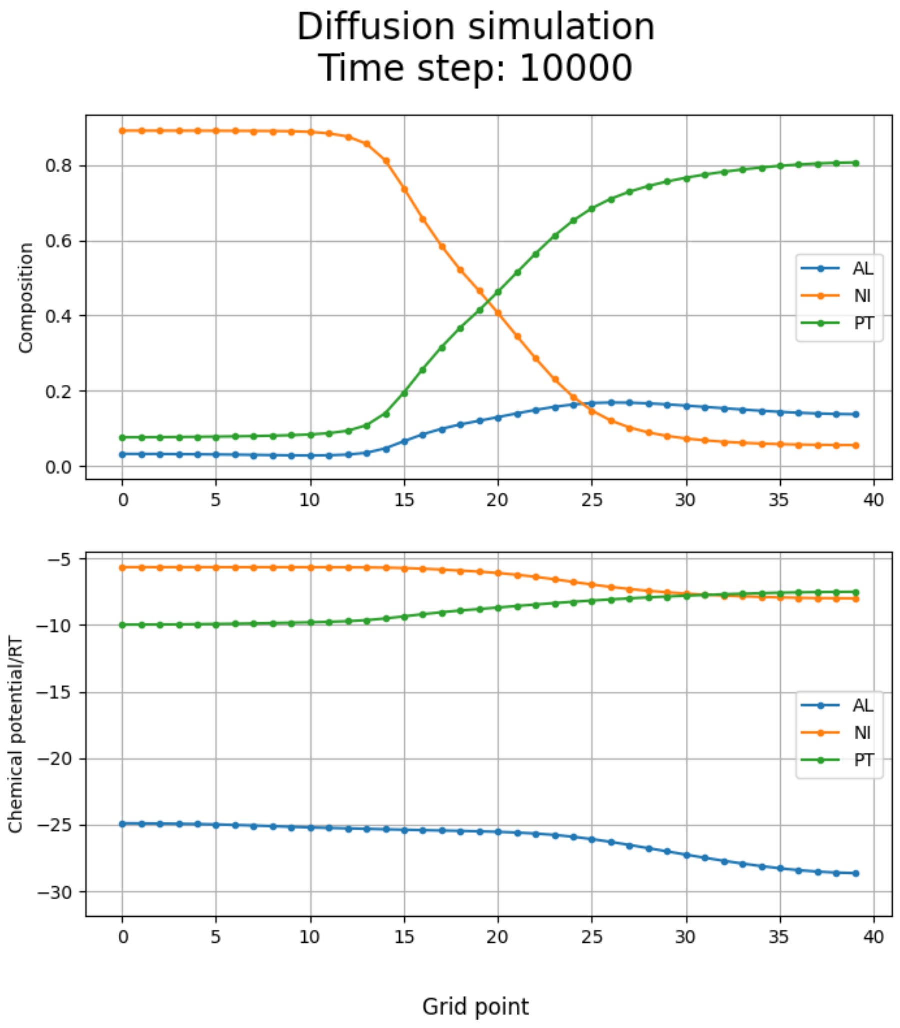

Fig. 9: Figure 10 Plot of composition profile and chemical potential at 10000 time steps

CHAPTER

SIX

API REFERENCE

6.1 Python

6.1.1 OCPython module

This is the main part of OC-Python (under development)

Notes: 1) It must be used together with the library (liboctq_f90wrap.cp38-win_amd64.pyd or liboctq_f90wrap.cp39-win_amd64.pyd) 2) The partial codes are based on the code from OpenCalphad GitHub (https://github.com/sundmanbo/opencalphad/tree/master/OCisoCbinding)

author: Chunhui Luo, 2022

class `OCPython.`**`GetResults`**(*self_*)

Bases: `object`

class: get results

`getComponentAssociatedResult`(*symbol: str*)

get component associated result

Parameters
`param1` – symbol

Returns
calculated property

Return type
dict

`getPhaseAssociatedResult`(*symbol: str*)

get phase associated result

Parameters
`param1` – symbol

Returns
calculated property

Return type
dict

`getPhaseConstituentComposition`()

get phase constituent composition

Returns
phase constituent composition

Return type
dict

`getPhaseElementComposition()`

get phase element composition

Returns
phase element composition

Return type
dict

`getPhaseSites()`

get phase sites

Returns
phase sites

Return type
dict

`getScalarResult(`*symbol: str*`)`

get scalar result

Parameters
`param1` – symbol

Returns
calculated property

Return type
value

`class OCPython.GridMinimizerStatus(`*value*`)`

Bases: `IntEnum`

class: set GridMinimizerStatus (On or Off)

`Off = -1`

`On = 0`

`class OCPython.MassUnit(`*value*`)`

Bases: `IntEnum`

class: set mass unit

`MassFraction = 2`

`MoleFraction = 1`

`class OCPython.PhaseStatus(`*value*`)`

Bases: `IntEnum`

class: Set phase status (Suspended, Dormant, Entered, Fixed)

`Dormant = -2`

`Entered = 0`

`Fixed = 2`

`Suspended = -3`

`class OCPython.SingleEquilibriumCalculation(`*vs*`)`

Bases: `object`

Single Equilibrium Calculation

`batchEquilibriaComp`(*n_xfrac*, *elementMoleFractions*, *xfrac_matrix*, *temp*, *stavar*)

Batch equilibrium calculations with composition loops

Parameters

- **`param1`** – n_xfrac:
- **`param2`** – elementMoleFractions
- **`param3`** – xfrac_matrix:
- **`param4`** – temp
- **`param5`** – stavar

Returns
calculated properties

Return type
list

`batchEquilibriaTemp`(*elementMoleFractions*, *xfrac_matrix*, *temp_list*, *stavar*)

Batch equilibrium calculations with temperature loops

Parameters

- **`param1`** – elementMoleFractions
- **`param2`** – xfrac_matrix:
- **`param3`** – temp_list
- **`param4`** – stavar

Returns
calculated properties

Return type
list

`calculateEquilibrium`(*gridMinimizerStatus=GridMinimizerStatus.On*)

calculate equilibrium

Parameters
`param1` – gridMinimizerStatus

`changeEquilibriumRecord`(*eqName=None*, *copiedEqName=None*)

change equilibrium record

`deleteEquilibrium`(*eqName=None*)

delete equilibrium with name

`eq`()

return self.eq

`getChemicalPotentials`()

get chemical potentials

Returns
chemical potential

Return type
dict

`getComponentNames`() → List[str]

Returns the ordered name list of the imported components after reading database.

Returns
names of components

Return type
list

`getConstituentsDescription()`
get constituents description

Returns
constituents description

Return type
dict

`getErrorCode()`
get error code

`getGibbsEnergy()`
get Gibbs energy

Returns
Gibbs energy

Return type
value

`getNumberPhase()`
get total number of phases

Returns
number of phases

Return type
int

`getPhaseConstituentComposition()`
get phase constituent composition

Returns
phase constituent composition

Return type
dict

`getPhaseElementComposition()`
get phase element composition

Returns
phase element composition

Return type
dict

`getPhaseName`(*index: int*)
get name of phase from index

Parameters
`param1` – index

Returns
phase name

Return type
string

`getPhaseSites()`
get phase sites

Returns
phase sites

Return type
dict

`getPhasesStatus`(*nPhase*)
set phase status

Parameters

- **param1** – phaseNames
- **param2** – phaseStatus
- **param3** – phaseAmount

`getScalarResult`(*symbol: str*)
get scalar result

Parameters
param1 – symbol

Returns
calculated property

Return type
Value

`getValueComponent`(*symbol: str*)
get component associated result

Parameters
param1 – symbol

Returns
calculated property

Return type
dict

`getValuePhase`(*symbol: str*)
get phase associated result

Parameters
param1 – symbol

Returns
calculated property

Return type
dict

`listConditions`()
show conditions for equilibrium calculation

`listEqResults`()
show result for equilibrium calculation

`listEqResults_lr1`()
show result for equilibrium calculation

`readtdb`(*tdbFilePath: str*, *elements=None*)
read database

Parameters

- **param1** – tdbFilePath

- **param2** – element

resetErrorCode()

reset error code

setElementMassFraction(*elementMassFractions*)

set element mass fraction

Parameters
param1 – elementMoleFractions

setElementMolarFraction(*elementMoleFractions: dict*)

set element molar fraction

Parameters
param1 – elementMoleFractions

setPhasesStatus(*phaseNames*, *phaseStatus*, *phaseAmount=0.0*)

set phase status

Parameters

- **param1** – phaseNames
- **param2** – phaseStatus
- **param3** – phaseAmount

setPressure(*pressure: float*)

set pressure

Parameters
param1 – pressure

setSingleElementMassFraction(*index*, *wfrac*)

set single element mass fraction

Parameters

- **param1** – index
- **param2** – wfrac

setSingleElementMolarFraction(*index*, *xfrac*)

set single element molar fraction

Parameters

- **param1** – index
- **param2** – xfrac

setTemperature(*temperature: Optional[float] = None*)

set temperature

Parameters
param1 – temperature

setTotalMolarAmount(*n: float*)

set total molar amount

Parameters
param1 – n

setquiet()

if argument TRUE spurious output should be suppressed `singleEquilibriumCalculation_Compact`(*tdbFilePath*, *elements*, *massunit*, *tpn*, *elementFractions*, *phaseNames=None*, *elementReferencePhase=None*)

Single Equilibrium Calculation with compact mode

Parameters

- **`param1`** – tdbFilePath
- **`param2`** – elements
- **`param3`** – tpn
- **`param4`** – elementMoleFractions
- **`param5`** – phaseNames
- **`param6`** – elementReferencePhase

class `OCPython.`**`Verbosity`**(*newlogfile*, *process_name*)

Bases: `object`

class: Verbosity

`setVerbosity`(*isVerbose*)

set Verbosity: True or False.

If Verbosity is True, logging level is set as DEBUG and more detailed information is shown.

If Verbosity is False, logging level is set as INFO and less information is shown.

6.1.2 OCPython utility module

This is the auxiliary part of OC-Python (under development)

author: Chunhui Luo, 2022

class `OCPython_utility.`**`OCPython_utility`**

Bases: `object`

static **`calc_phasefrac_temploop_newphase`**(*oc*, *gmStat*, *T_list*, *stateVar*)

Calculate phase fraction with temperature loop (allow new phase creation from FCC_A1)

Parameters

- **`param1`** – oc
- **`param1`** – gmStat
- **`param1`** – T_list

Returns

- *dict* – values_dict
- *list* – T_K,C_comp_in_FCC_A1

static **`calc_phasefrac_temploop_nonewphase`**(*oc*, *gmStat*, *T_list*, *stateVar*)

Calculate phase fraction with temperature loop (don’t allow new phase creation)

Parameters

- **`param1`** – oc
- **`param1`** – gmStat
- **`param1`** – T_list

Returns

- *dict* – values_dict

- *list* – T_K

`static comp_new_order(`*elements*`)`

get component associated result

To pass array of strings to Fortran in Python, one must create an array of chars with shape (<number of strings>, <string length>), fill its content, and then pass the char array to f2py generated function.

Here xstring is used for this purpose. The input is element list, the output is numpy string array, which is dedicated for use in reading tdb withe element names.

Parameters
`param1` – elements

Returns
names of components

Return type
numpy string array

`static getPhaseFraction(`*oc*`)`

get phase fraction (only non-zero values)

Parameters
`param1` – oc

Returns
phase fraction

Return type
list

`static getStablePhase(`*oc*`)`

get stable phases and its fractions ((i.e. the phases present in the current equilibrium))

`static getcomp(`*n*, *comp*`)`

get component list

Convert string array from Fortran -> component string list in Python

Parameters

- `param1` – n
- `param2` – comp

Returns
names of components

Return type
list

`static phase_parsing_bytes_string(`*nPhase*, *phasename_bytes*`)`

get component associated result

To pass array of strings to Fortran in Python, one must create an array of chars with shape (<number of strings>, <string length>), fill its content, and then pass the char array to f2py generated function.

Here xstring is used for this purpose. The input is element list, the output is numpy string array, which is dedicated for use in reading tdb withe element names.

Parameters
`param1` – elements

Returns
names of components

Return type
numpy string array

static plot3D(*x_list*, *y_list*, *result_list*, *xlabel*, *ylabel*, *zlabel*, *title*)

Plot a 3D surface.

Parameters

- **param1** – x_list: data for the x-axis
- **param2** – y_list: data for the y-axis
- **param3** – result_list: data for the z-axis
- **param4** – xlabel: label for the x-axis
- **param5** – ylabel: label for the y-axis
- **param6** – zlabel: label for the z-axis
- **param7** – title: title of the figure

static plotContour(*x_list*, *y_list*, *result_list*, *xlabel*, *ylabel*, *title*)

Plot a contour.

Parameters

- **param1** – x_list: data for the x-axis
- **param2** – y_list: data for the y-axis
- **param3** – result_list: data for the z-axis
- **param4** – xlabel: label for the x-axis
- **param5** – ylabel: label for the y-axis
- **param6** – title: title of the figure

6.2 Fortran

6.2.1 Subroutines in Liboctq.f90

subroutine tqini(n,ceq)

! initiate workspace

implicit none

integer n ! Not nused, could be used for some initial allocation

type(gtp_equilibrium_data), pointer :: ceq ! EXIT: current equilibrium

subroutine tqrfil(filename,ceq)

! read all elements from a TDB file

implicit none

character*(*) filename ! IN: database filename

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqrpfil(filename,nsel,selel,ceq)

! read TDB file with selection of elements

implicit none

character*(*) filename ! IN: database filename

integer nsel ! IN: number of elements

character selel(*)*2 ! IN: elements to be read from the database

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqcecompact(filename,nsel,massunit,selel,tpn,xi,xf,phnames1,elref,phref, ceq)

! single equilibrium calculation with compact mode

! It combines reading tdb, setting phase status, setting reference phase of element, setting conditions, performing equilibrium calculation.

implicit none

character*(*) filename ! IN: database filename

integer nsel ! IN: number of elements

integer massunit ! IN: unit of mass

character target*60,selel(*)*2 ! IN: element names

double precision tpn(3) ! IN: values of temperature, pressure and moles

integer :: xi(maxel) ! IN: index of element

double precision xf(maxel) ! IN: mole fraction of element

character phnames1*60 ! IN: phase names

character elref*100 ! IN: reference element

character phref*100 ! IN: reference phase

subroutine tqcompbatch(nsel,nxfrac,xi,xfrac,temp,stavar,values,ceq)

! batch calculation with composition loop

! composition matrix is used.

! each row in composition matrix: a set of compositions for an alloy

! number of rows stands for number of composition variations.

implicit none

integer nsel ! IN: number of element

integer nxfrac ! IN: number of fraction vector

integer :: xi(nsel) ! IN: index of element

double precision xfrac(nxfrac,nsel) ! IN: mole fraction of element

double precision temp ! IN: temperature

character stavar*(*) ! IN: name of state variable

double precision values(*) ! EXIT: calculated state variable

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqtempbatch(nsel,ntemp,xi,xfrac,temp,stavar,values,ceq)

! batch calculation with temperature loop

implicit none

integer nsel ! IN: number of element

integer ntemp ! IN: number of temperature

integer :: xi(nsel) ! IN: index of element

double precision xfrac(nsel) ! IN: mole fraction of element

double precision temp(ntemp) ! IN: temperature

character stavar*(*) ! IN: name of state variable

double precision values(*) ! EXIT: calculated state variable

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqgcom_(n,compnames,ceq)

! get system component names. At present the elements

implicit none

integer n ! EXIT: number of components

character*24, dimension(*) :: compnames ! EXIT: names of components

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqgcom(n,compnames,ceq)

! get system component names. At present the elements

implicit none

integer, intent(out) :: n ! EXIT: number of components

character*2, dimension(10), intent(out) :: compnames ! EXIT: names of components

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqgnp(n,ceq)

! get total number of phase tuples (phases and composition sets)

! A second composition set of a phase is normally placed after all other

! phases with one composition set

implicit none

integer n !EXIT: n is number of phases

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqgpn(phtupx,phasename,ceq)

! get name of phase tuple with index phtupx (ceq redundant)

implicit none

integer phtupx !IN: index in phase tuple array

character phasename*(*) !EXIT: phase name, max 24+8 for pre/suffix

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqgpi(phtupx,phasename,ceq)

! get phasetuple index of phase phasename (including comp.set (ceq redundant)

implicit none

integer phtupx !EXIT: phase tuple index

character phasename*(*) !IN: phase name

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqgpi2(iph,ics,phasename,ceq)

! get indices of phase phasename (ceq redundant)

implicit none

integer iph, ics !EXIT: phase indices

character phasename*(*) !IN: phase name

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

integer phtupx

`subroutine tqgpcn2(n,c,csname)`

! get name of constituent with index c in phase with index n

! NOTE An identical routine with different constituent index is tqgpcn

implicit none

integer n !IN: phase number (not phase tuple)

integer c !IN: constituent index sequentially over all sublattices

character csname*(*) !EXIT: constituent name

`subroutine tqgpci(n,c,constituentname,ceq)`

! get index of constituent with name in phase n

implicit none

integer n !IN: phase index

integer c !IN: sequential constituent index over all sublattices

character constituentname*(*)

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqgpcs(c,nspel,ielno,stoi,smass,qsp)`

! get description of constituent c (stoichiometry, mass, charge)

implicit none

integer c !IN: sequential constituent index over all sublattices

integer nspel !EXIT: number of elements in species

integer ielno(*) !EXIT: element indices

double precision stoi(*) !EXIT: stoichiometry of elements

double precision smass !EXIT: mass

double precision qsp !EXIT: charge of the species

`subroutine tqgccf(n1,n2,elnames,stoi,mass,ceq)`

! get stoichiometry of component n1

! n2 is number of elements (dimension of elnames and stoi)

implicit none

integer n1 !IN: component number

integer n2 !EXIT: number of elements in component

character elnames(*)*(2) ! EXIT: element symbols

double precision stoi(*) ! EXIT: element stoichiometry

double precision mass ! EXIT: component mass (sum of element mass)

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqgnpc(n,c,ceq)`

! get number of constituents of phase n

implicit none

integer n !IN: Phase number

integer c !EXIT: number of constituents

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqphsts(phtupx,newstat,val,ceq)

! set status of phase tuple: SUSPEND: newstat=-3;DORMANT: newstat=-2; ENTERED: newstat=-1/0/1; FIX: newstat=2

integer phtupx ! IN: index in phase tuple array

integer newstat ! IN: phase status

double precision val ! EXIT:

type(gtp_equilibrium_data), pointer :: ceq ! IN: current equilibrium

subroutine tqphsts2(phnames,newstat,val,ceq)

! set status of many phases: SUSPEND: newstat=-3;DORMANT: newstat=-2; ENTERED: newstat=-1/0/1; FIX: newstat=2

! 1) all phases: phnames = '*', or 2) several phases: phnames = 'Phase1; …; Phase n'

character phnames*(*) ! IN: phase names (character)

integer newstat ! IN: phase status

double precision val ! EXIT:

type(gtp_equilibrium_data), pointer :: ceq ! IN: current equilibrium

subroutine tqgpsm(nphase,phases,status,amdgm,ceq)

! get all phase names and their status

! status = 2 fix, 1,0,-1 entered, -2 dormant, -3 suspended

! if status 0 or less the phase is not stable, extract DGM

! if this phase is stable, extract amount

integer, intent(in) :: nphase !IN: phase number

character*20, dimension(*), intent(out) :: phases !IN: phase name

integer, dimension(nphase), intent(inout) :: status ! EXIT: phase status

double precision, intent(inout) :: amdgm(*) ! EXIT: DGM or amount

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqsetc(stavar,n1,n2,value,cnum,ceq)

! set condition

! stavar is state variable as text

! n1 and n2 are auxilliary indices

! value is the value of the condition

! cnum is returned as an index of the condition.

! to remove a condition the value sould be equial to RNONE ????

! when a phase indesx is needed it should be 10*nph + ics

! see TQGETV for doucumentation of stavar etc.

implicit none

integer n1 ! IN: 0 or phase tuple index or component number

integer n2 ! IN: 0 or component number

integer cnum ! EXIT: sequential number of this condition

character stavar*(*) ! IN: character with state variable symbol

double precision value ! IN: value of condition type(gtp_equilibrium_data), pointer :: ceq ! IN: current equilibrium

`subroutine tqtgsw(i)`

! toggle global status word of index i

implicit none

integer i !IN: global status word of index i

`subroutine tqce(target,n1,n2,value,ceq)`

! calculate equilibrium with possible target

! Target can be empty or a state variable with indices n1 and n2

! value is the calculated value of target

implicit none

character target*(*) ! IN:

integer n1 ! IN: n1 = 0 with grid minimizer; n1 = -1 without grid minimizer

integer n2 ! IN:

double precision value ! EXIT

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqgetv(stavar,n1,n2,n3,values,ceq)`

! get equilibrium results using state variables

implicit none

character stavar*(*) ! IN: the state variable IN CAPITAL LETTERS with indices n1 and n2

integer n1 ! IN: phase tuple index

integer n2 ! IN: component index

integer n3 ! IN: the dimension of the array values when be called, changed to number of values on exit

double precision values(*) ! EXIT: an array with the calculated value(s)

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

!==

! stavar must be a symbol listed below

! IMPORTANT: some terms explained after the table

! Symbol index1,index2 Meaning (unit)

!.... potentials

! T 0,0 Temperature (K)

! P 0,0 Pressure (Pa)

! MU component,0 or ext.phase.index*1,constituent*2 Chemical potential (J)

! AC component,0 or ext.phase.index,constituent Activity = EXP(MU/RT)

! LNAC component,0 or ext.phase.index,constituent LN(activity) = MU/RT

!...... extensive variables

! U 0,0 or ext.phase.index,0 Internal energy (J) whole system or phase

! UM 0,0 or ext.phase.index,0 same per mole components

! UW 0,0 or ext.phase.index,0 same per kg

! UV 0,0 or ext.phase.index,0 same per m3

! UF ext.phase.index,0 same per formula unit of phase

! S*3 0,0 or ext.phase.index,0 Entropy (J/K)

! V 0,0 or ext.phase.index,0 Volume (m3)

! H 0,0 or ext.phase.index,0 Enthalpy (J)

! A 0,0 or ext.phase.index,0 Helmholtz energy (J)

! G 0,0 or ext.phase.index,0 Gibbs energy (J)

! ….. some extra state variables

! NP ext.phase.index,0 Moles of phase

! BP ext.phase.index,0 Mass of moles (kg)

! Q ext.phase.index,0 Internal stability/RT (dimensionless)

! DG ext.phase.index,0 Driving force/RT (dimensionless)

!…… amounts of components

! N 0,0 or component,0 or ext.phase.index,component Moles of component

! X component,0 or ext.phase.index,component Mole fraction of component

! B 0,0 or component,0 or ext.phase.index,component Mass of component

! W component,0 or ext.phase.index,component Mass fraction of component

! Y ext.phase.index,constituent*1 Constituent fraction

!……. some parameter identifiers

! TC ext.phase.index,0 Magnetic ordering temperature

! BMAG ext.phase.index,0 Aver. Bohr magneton number

! MQ& ext.phase.index,constituent Mobility

! THET ext.phase.index,0 Debye temperature

! LNX ext.phase.index,0 Lattice parameter

! EC11 ext.phase.index,0 Elastic constant C11

! EC12 ext.phase.index,0 Elastic constant C12

! EC44 ext.phase.index,0 Elastic constant C44

!……. NOTES:

! 1 The ext.phase.index is 10*phase_number+comp.set_number

! 2 The constituent index is 10*species_number + sublattice_number

! 3 S, V, H, A, G, NP, BP, N, B and DG can have suffixes M, W, V, F also

!————————————————————————

! special addition for TQ interface: d2G/dyidyj

! D2G + phase tuple

!————————————————————————

`subroutine tqgdmat(phtupx,tpval,xknown,cpot,tyst,nend,mugrad,mobval,consnames,n1,ceq)`

! equilibrates the constituent fractions of a phase for mole fractions xknown

! and calculates the Darken matrix and unreduced diffusivities

implicit none

integer phtupx ! IN: index in phase tuple array double precision tpval(*) ! IN: T and P

double precision xknown(*) ! IN: phase composition

double precision cpot(*) ! EXIT: (calculated) chemical potentials

logical tyst ! IN: TRUE means no output

integer nend ! EXIT: number of values returned in mugrad (dG_A/dN_B)

double precision mugrad(*) ! EXIT: derivatives of the chemical potentials wrt mole fractions??

double precision mobval(*) ! EXIT: mobilities

character*24, dimension(*) :: consnames ! EXIT: names of constituents

integer n1 ! EXIT: number of constituents

TYPE(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqgphc1(n1,nsub,cinsub,spix,yfrac,sites,extra,ceq)

! get phase constitution

! This subroutine returns the sublattices and constitution of a phase

implicit none

integer n1 ! IN: phase tuple index

integer nsub ! IN: number of sublattices (1 if no sublattices)

integer cinsub(*) ! EXIT: array with the number of constituents in each sublattices

integer spix(*) ! EXIT: array with the species index of the constituents in all sublattices

double precision yfrac(*) ! EXIT: constituent fractions in same order as in spix

double precision sites(*) ! EXIT: array of the site ratios for all sublattices

double precision extra(*) ! EXIT: array with some extra values: extra(1) is the number of moles of components per formula unit; extra(2) is the net charge of the phase

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqsphc1(n1,yfra,extra,ceq)

! set constitution of a phase

! NOTE The constituents fractions are normalized to sum to unity for each

! sublattice and extra is calculated by tqsphc1

! T and P must be set as conditions.

implicit none

integer n1 ! IN: phase tuple index

double precision yfra(*) ! EXIT: array with the constituent fractions in all sublattices in the same order as obtained by tqgphc

double precision extra(*) ! EXIT: array with returned values with the same meaning as in tqgphc1

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

subroutine tqcph1(n1,n2,n3,gtp,dgdy,d2gdydt,d2gdydp,d2gdy2,ceq)

! calculate phase properties and return arrays

!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

! WARNIG: this is not a subroutine to calculate chemical potentials

! those can only be obtained by an equilibrium calculation.

! The values returned are partial derivatives of G for the phase at the

! current T, P and phase constitution. The phase constitution has been

! obtained by a previous equilibrium calculation or

! set by the subroutine tqsphc

! The subroutine is equivalent to the "calculate phase" command.

!

! NOTE that values are per formula unit divided by RT,

! divide also by extra(1) in subroutine tqsphc1 to get them per mole component

!

!^^

! calculate G and some or all derivatives for a phase at current composition

! They are returned in the order: 1,1; 1,2; 1,3; …

! 2,2; 2,3; …

! 3,3; …

! for indexing one can use the integer function ixsym(i1,i2)

implicit none

integer n1 ! IN: phase tuple index

integer n2 ! IN: = 0 if only G and derivatives wrt T and P

! = 1 also first derivatives wrt compositions

! = 2 if also 2nd derivatives

integer n3 ! EXIT: number of constituents (dimension of returned arrays)

double precision gtp(6) ! EXIT: array with G, G.T, G:P, G.T.T, G.T.P and G.P.P

double precision dgdy(*) ! EXIT: array with G.Yi

double precision d2gdydt(*) ! EXIT: array with G.T.Yi

double precision d2gdydp(*) ! EXIT: array with G.P.Yi

double precision d2gdy2(*) ! EXIT: array with the upper triangle of the symmetrix matrix G.Yi.Yj

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqcph2(n1,n2,n3,n4,ceq)`

! calculate phase properties and return index

!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

! WARNIG: this is not a subroutine to calculate chemical potentials

! those can only be made by an equilibrium calculation.

! The values returned are partial derivatives of G for the phase at the

! current T, P and phase constitution. The phase constitution has been

! obtained by a previous equilibrium calculation or

! set by the subroutine tqsphc

! It corresponds to the "calculate phase" command.

!

! NOTE that values are per formula unit divided by RT,

! divide also by extra(1) in subroutine tqsphc1 to get them per mole component

!

!^^^

! calculate G and some or all derivatives for a phase at current composition

! for indexing one can use the integer function ixsym(i1,i2)

implicit none

integer n1 ! IN: phase tuple index

integer n2 ! IN: type of calculation (0, 1 or 2)

integer n3 ! EXIT: returned as number of constituents

integer n4 ! EXIT: index to ceq%phase_varres(lokres)% with all results

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqcph3(n1,n2,g,ceq)`

! Calculate phase properties and return single array

!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

! WARNIG: this is not a subroutine to calculate chemical potentials

! those can only be made by an equilibrium calculation.

! The values returned are partial derivatives of G for the phase at the

! current T, P and phase constitution. The phase constitution has been

! obtained by a previous equilibrium calculation or

! set by the subroutine tqsphc

! It corresponds to the "calculate phase" command.

!

! NOTE that values are per formula unit divided by RT,

! divide also by extra(1) in subroutine tqsphc1 to get them per mole component

!

!^^^

! calculate G and some or all derivatives for a phase at current composition

! g is an array with G derivatives under the form:

! G_m^alpha = G_M^alpha/N^alpha, frac{partial G_m^alpha}{partial T}, frac{partial G_m^alpha}{partial P}, frac{partial^2 G_m^alpha}{partial T^2}

! 1/N^alpha * frac{partial G_M^alpha}{partial y_i} (if n2>=1)

! 1/N^alpha * frac{partial^2 G_M^alpha}{partial y_ipartial y_j} (if n2>=2)

implicit none

integer n1 ! IN: phase tuple index

integer n2 ! IN: = 0 if only G and derivatives wrt T and P

! = 1 also first derivatives wrt compositions

! = 2 if also 2nd derivatives

double precision g(*) ! EXIT: array with G derivatives under the form:

! G_m^alpha = G_M^alpha/N^alpha,

! frac{partial G_m^alpha}{partial T},

! frac{partial G_m^alpha}{partial P},

! frac{partial^2 G_m^alpha}{partial T^2}

! 1/N^alpha * frac{partial G_M^alpha}{partial y_i} (if n2>=1)

! 1/N^alpha * frac{partial^2 G_M^alpha}{partial y_ipartial y_j} (if n2>=2)

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqdceq(name)`

! delete equilibrium with name

implicit none

character name*24 ! IN: name of equilibrium

`subroutine tqcceq(name,n1,newceq,ceq)`

! copy current equilibrium to newceq

! creates a new equilibrium record with name with values same as ceq

! n1 is returned as index

implicit none

character name*24 ! IN: name of equilibrium

integer n1 ! EXIT: index for equilibrium

type(gtp_equilibrium_data), pointer :: newceq,ceq !IN: new and current equilibrium

`subroutine tqselceq(name,ceq)`

! select current equilibrium to be that with name.

! Note that equilibria can be deleted and change number but not name

implicit none

character name*24 ! IN: name of equilibrium

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine Change_Status_Phase(myname,nystat,myval,ceq)`

implicit none

character myname*24 ! IN: name of phase

integer nystat ! IN: phase status

double precision myval ! IN: amount to be FIX or use as start value

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqcref(ciel,phase,tpref,ceq)`

! set component reference state

integer ciel ! IN: element index

character phase*(*) ! IN: name of phase

double precision tpref(*) ! IN: T and P values

type(gtp_equilibrium_data), pointer :: ceq ! IN: current equilibrium `subroutine tqlr(lut,ceq)`

! list the equilibrium results like in OC

implicit none

integer lut ! IN: unit for listing, =6 screen

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqlr1(lut,ceq)`

! list the equilibrium results like in OC

implicit none

integer lut ! IN: unit for listing, =6 screen

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqlc(lut,ceq)`

! list conditions like in OC

implicit none

integer lut ! IN: unit for listing, =6 screen

type(gtp_equilibrium_data), pointer :: ceq !IN: current equilibrium

`subroutine tqltdb`

! list TDB file elements, phases and parameters on screen

implicit none

`subroutine tqquiet(yes)`

! if argument TRUE spurious output should be suppressed

implicit none

logical yes ! IN: .TRUE. (yes)

CHAPTER SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX