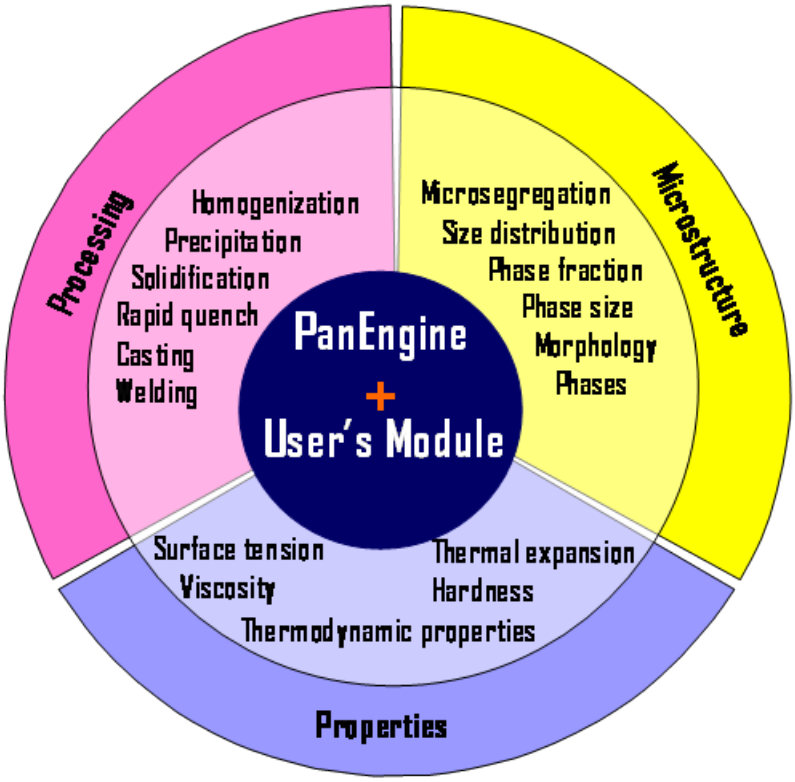
PanEngine^{TM}

User's Guide



CompuTherm, LLC

Copyright© 2012-2021

Getting Help

CompuTherm LLC is committed to providing you with the best possible technical support. Please contact us via the following approaches:

Web Site

www.computherm.com

E-Mail

info@computherm.com

Phone

+1 (608) 203 8843

Fax

+1 (608) 203 8045

Mail

CompuTherm LLC
8401 Greenway Blvd., Suite 248
Middleton, WI 53562
USA

Declaration

This document is furnished by CompuTherm LLC for information purposes only to licensed users of the `PanEngine` product and is furnished on an "AS IS" basis without any warranties expressed or implied. Information in this document is subject to change without notice and does not represent a commitment on the part of CompuTherm LLC.

Contents

1 Introduction to PanEngine **1**

1.1 What is PanEngine? . . . 1

1.2 Advantages of PanEngine . . . 1

1.3 API in PanEngine . . . 1

2 Getting Started with PanEngine **3**

2.1 Installation of PanEngine . . . 3

2.2 Getting Started with PanEngine . . . 4

3 Basic Concepts **9**

3.1 Gibbs Energy Models for Multi-Component Phases . . . 9

3.1.1 Stoichiometric compound . . . 9

3.1.2 Disordered solution phase . . . 9

3.1.3 Ordered intermetallic phase using the compound energy formalism . . . 10

3.2 PanEngine Classes . . . 10

3.2.1 class `P_Component` . . . 11

3.2.2 class `P_Species` . . . 11

3.2.3 class `P_Statespace` . . . 11

3.2.4 class `P_Phase_Point` . . . 12

3.2.5 class `P_Point` . . . 12

4 PanEngine API **13**

4.1 Functions for PanEngine Pointer . . . 15

4.1.1 Define a PanEngine pointer . . . 15

4.1.2 Delete a PanEngine pointer . . . 15

4.2 Functions for System . . . 15

4.2.1 Set system configuration . . . 15

4.2.2 Import a thermodynamic database . . . 16

4.2.3 Export a subsystem database into a file (`tdb`) . . . 16

4.2.4 Get component names in a database . . . 16

4.2.5 Deactivate a component in a database . . . 16

4.2.6 Get phase names in a database . . . 17

4.2.7 Get and set phase statuses . . . 17

4.3 Functions for Point Calculation . . . 17

4.3.1 Set calculation condition . . . 18

4.3.2 Calculate a global phase equilibrium . . . 18

4.3.3 Calculate a global phase equilibrium with initials . . . 18

4.3.4 Calculate a local phase equilibrium . . . 18

4.3.5 Find the liquidus surface . . . 19

4.3.6 Find the liquidus slopes . . . 19

4.4 Function for Solidification Simulation 20
4.5 Thermodynamic Factors 21
4.6 Hessian matrix of Gibbs free energy 21
4.7 Parallel Tangent Equilibrium 22

5 Examples **24**
5.1 Test Example 1 24
5.2 Test Example 2 25
5.3 Test Example 3 25
5.4 Test Example 4 25
5.5 Test Example 5 25
5.6 Test Example 6 25
5.7 Test Example 7 26
5.8 Test Example 8 26
5.9 Test Example 9 26

1 Introduction to PanEngine

1.1 What is PanEngine?

`PanEngine` is a dynamically linked library (DLL) in 64-bit for multi-component thermodynamics and phase equilibrium calculations. `PanEngine` is the calculation engine of Pandat. It has an application program interface (API) which allows a user's custom program to access the functions in `PanEngine`. It is implemented with C++TR1 standard in Microsoft Visual Studio. The examples in this manual were prepared under Microsoft Visual Studio 2015.

A library is a group of functions, classes, or other resources that can be made available to application programs that need previously implemented entities without the need to know how these functions, classes, or resources were created or how they function. A dynamic link library is a program that holds one or more functions or some functionality that other programs can use. Through `PanEngine`'s API, users can call the thermodynamic functions available in `PanEngine` and create custom software for their specific applications.

Custom Software Applications

`PanEngine` can be used by users to create custom software applications such as:

- Microscopic solidification simulations
  - Microstructure: e.g. the secondary dendrite arm spacing
  - Microsegregation: e.g. the concentration profile within a dendrite arm
- Macroscopic solidification simulations
  - Casting simulations: `PanEngine` provides enthalpy and the fraction-solid as a function of temperature as well as physical properties such as density, and thermal conductivity
- Heat treatment simulations
- Other applications where phase equilibrium information and thermodynamic properties are needed, such as the cellular automaton (CA) and phase field simulations

1.2 Advantages of PanEngine

`PanEngine` automatically finds the *correct*, *stable* phase equilibria without requiring the user to guess initial values. This is especially important when integrating with user's custom program for the following reasons:

- It is very difficult for a user to provide initial values and verify results when a custom software program needs stable phase equilibrium repeatedly for thousands of points.
- It is almost impossible for a user to guess the initial values in a multi-component phase equilibrium calculation.

1.3 API in PanEngine

`PanEngine`'s API has many commonly used functions. Some of them are listed below and more details will be given in the following sections.

- Import databases

- Set calculation conditions
- Calculate stable equilibria
- Calculate metastable (local) equilibria
- Calculate parallel tangent equilibria for phase field modeling
- Calculate driving force of a phase
- Find liquidus surface
- Calculate liquidus and solidus slopes
- Calculate partitioning coefficients
- Simulate a solidification process using Scheil or lever rule model
- Calculate physical properties such as molar volume and density
- Calculate kinetic properties such as mobility and diffusivity
- Calculate Hessian matrix of Gibbs free energy and its eigenvalues and eigenvectors
- Calculate user-defined properties of a phase or a system

2 Getting Started with PanEngine

2.1 Installation of PanEngine

PanEngine is available only from CompuTherm LLC. Once purchased, a hardware dongle will be provided with PanEngine. PanEngine will not run if the dongle is not attached to the user's computer's USB port. PanEngine consists of several different types of files. As shown in Table 1, the PanEngine thermodynamic calculation interface includes .lib file, .dll files, .h files, Visual Studio Solution and Project files, and some test example files (.cpp).

Table 1: List of PanEngine Files

File Name	Comment
PanEngineX.dll	PanEngine dynamically linked library
PanSolverX.dll haspms32.dll hasp_windows_57714.dll	Collection of dynamically linked libraries used by PanEngineX.dll
PanEngineX.lib	PanEngine library file
PanEngineX.h	PanEngine header file
std.h	Standard header file
stl.h	Standard Template Library (STL) header file
Pan_Global.h Pan_Global_Def.h solidification.h	Other header files
main.cpp	Main program
PanEngineTest_*.cpp	PanEngine test example files (*=1,2,$\cdots$,9)
AlMgZn.tdb, AlSiZn.tdb, FeNiCr.tdb, NiAlNb_Pseudo.tdb	Example database files in tdb format
PanEngineXTest.sln	Visual Studio Solution file
PanEngineXTest.vcxproj	VC++ Project file
PanEngineXTest.exe or PanEngineXTest_demo.exe	Compiled executable application file

The installation of PanEngine is rather straightforward. Simply copy all the files in the PanEngine

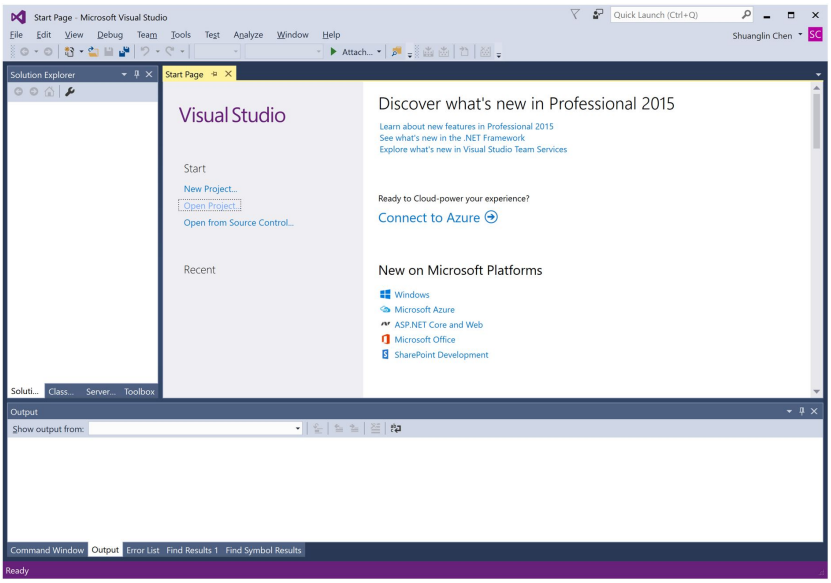
distribution CD or unzip the compressed files to any working directory where the user intends to place his/her own codes for applications. The library files can also locate in any other area, and can be accessed by specifying their appropriate paths in the application program codes. The current `PanEngine` can run on most of recent versions of Windows.

The recommended programming environment with `PanEngine` is Microsoft Visual Studio 2013 or 2015. Visual Studio 2013 or 2015 is the programming environment in which `PanEngine` was created. If a user uses a different C++ compiler, the `PanEngine` Visual Studio Solution file and corresponding project file may not work and then a completely new Solution and project files or make file need to be constructed by the user, or contact CompuTherm for solutions.

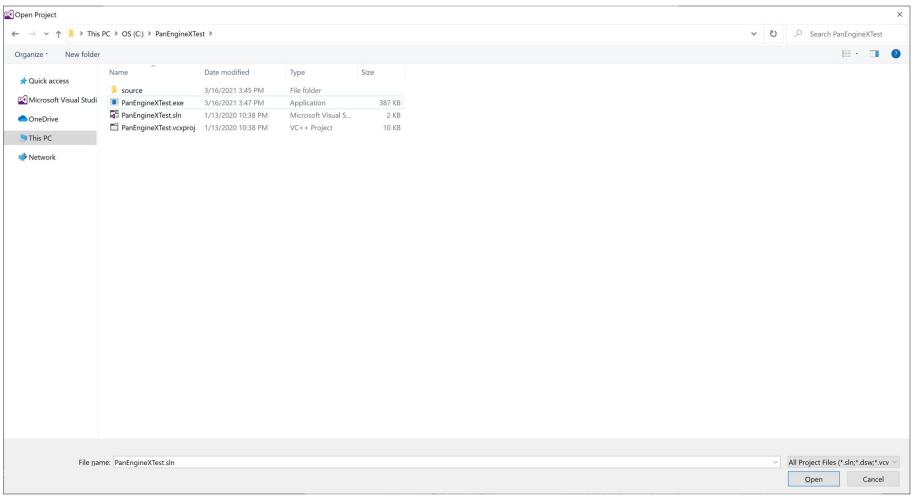
2.2 Getting Started with PanEngine

We assume that a full version of Microsoft Visual Studio 2013 or 2015 is installed on the user's computer. Follow the steps below to run the `PanEngine` test examples.

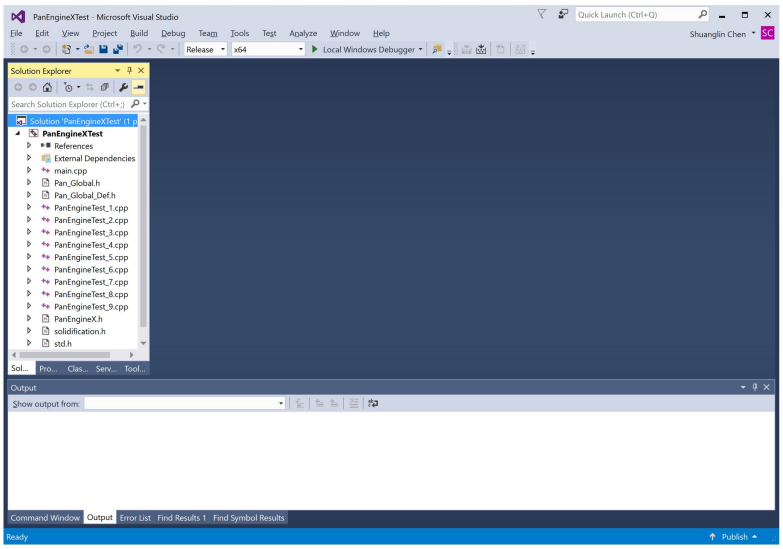
1. Attach the CompuTherm hardware dongle to the computer.
2. Start Microsoft Visual Studio 2013 or 2015.



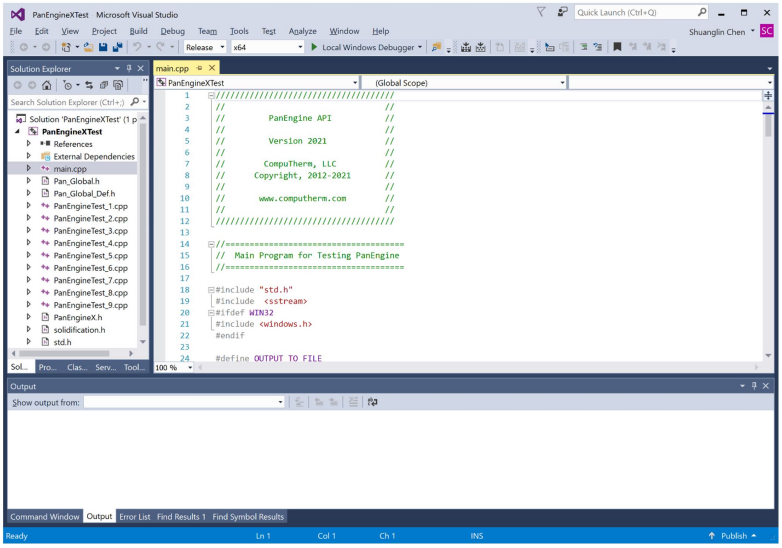
3. On the `Start` page of Visual Studio 2013 or 2015, click on `Open Project`. Go to the folder `/PanEngineXTest` (in the user's hard drive) and open `PanEngineXTest.sln`.



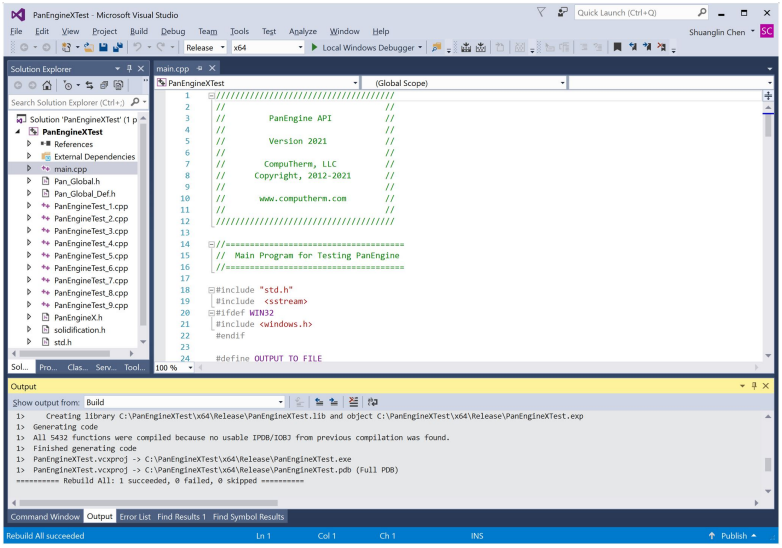
Here is what we will see in Visual Studio after expanding the file folders in the `Solution Explorer`:



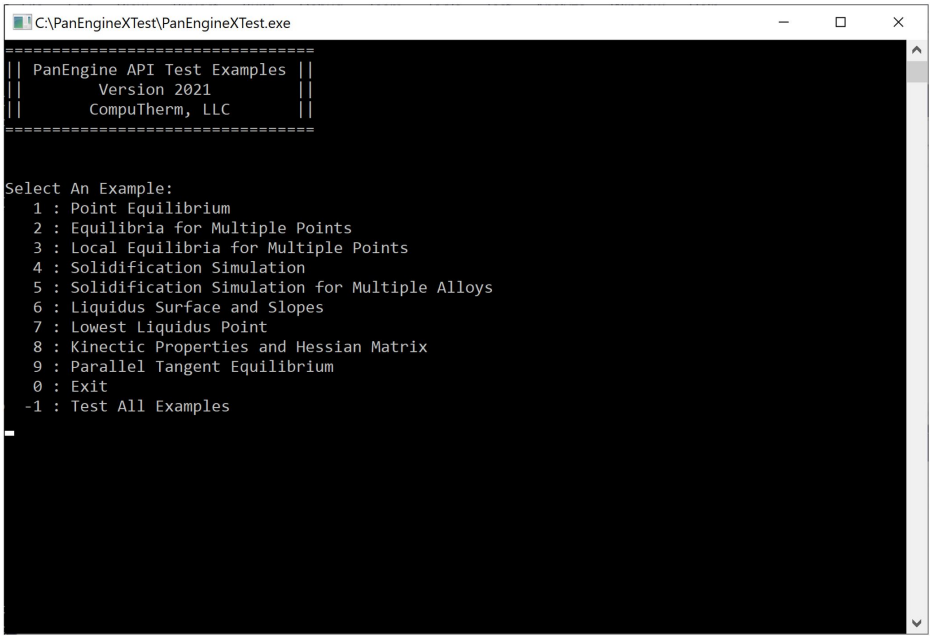
4. Double click on the file `main.cpp` in the `Solution Explorer` window.



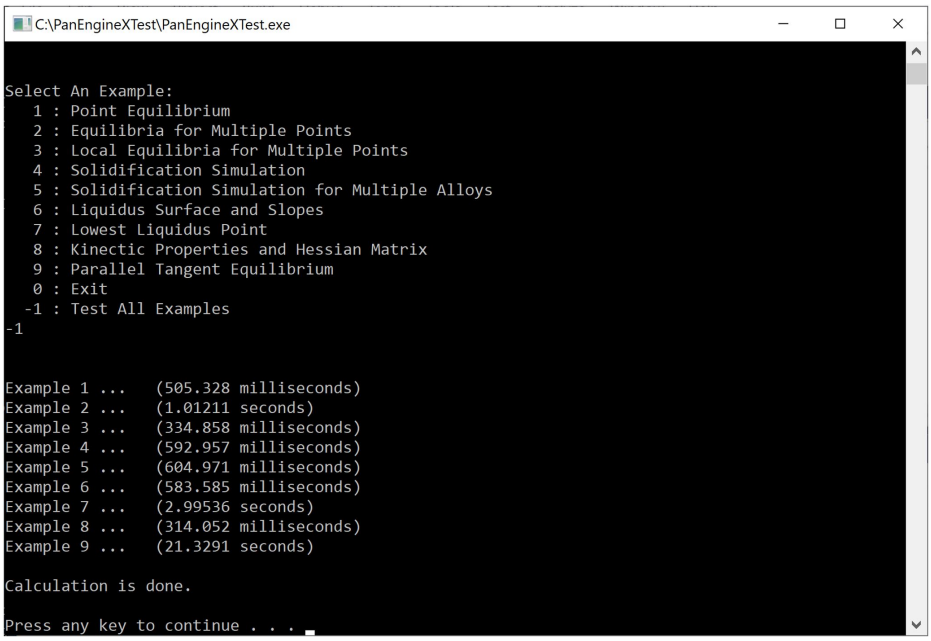
5. Rebuild `PanEngine` by clicking `Build` → `Rebuild Solution`.



6. Press F5 to run the test examples. A `Command` window will pop up as below.



There are nine test examples to select. To select a test example, enter the example ID `1` to `9`. Type "`0`" to exit, and type "`-1`" to run all the test examples together. The `Command` window will show the intermediate results of the calculations. The final status of the window looks like the following one (after selected "`-1`"), except that the path name on the top of the window will depend on the location of `PanEngine` on the user's computer.

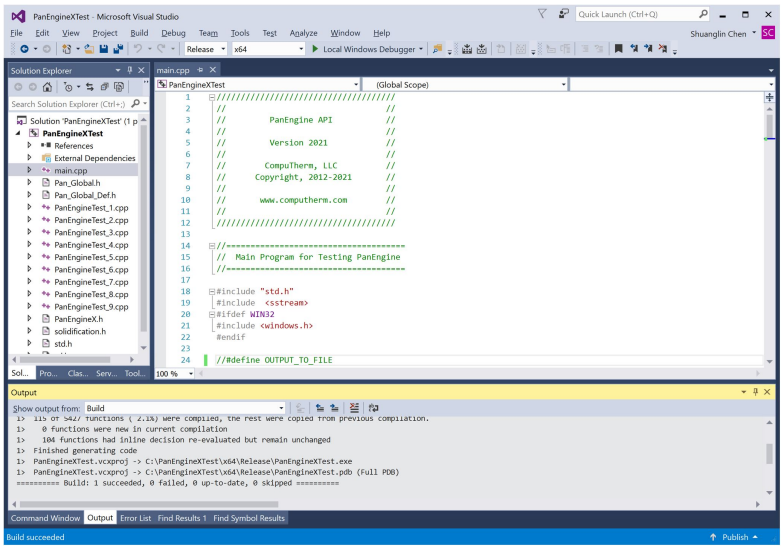


7. Press any key and return to the Visual Studio main window.

In the `main.cpp`, there is a line

```
#define OUTPUT_TO_FILE
```

as shown in the following image.



If this `#define` line is commented out, as in above image, the intermediate calculation results will be shown in the `Command` window. Otherwise, the results will be output into a file with a name of "`test_1.dat`", or "`test_2.dat`", etc.

In the following, we will introduce some basic concepts used in `PanEngine` and describe the details of the API and the test examples.

3 Basic Concepts

In the CALPHAD approach, the Gibbs energies of all the phases in an alloy system are described by thermodynamic models, such as the ones for stoichiometric phases, the regular-solution-type model for disordered phases, and the sublattice model for ordered phases with a range of homogeneity or an order/disorder transition. These types of models have been implemented in `PanEngine`.

`PanEngine` was developed with C++ language. It consists of many C++ classes. The `P_Point` C++ class refers to a system with a specified composition at a certain temperature and pressure. A `P_Point` object is directly interfaced with the user's code. The user can change the conditions (temperature or overall compositions) of the system through a `P_Point` object and get back the thermodynamic properties and phase equilibrium information under the newly specified conditions. The stable (or metastable) phase equilibrium information of the system (such as phase fractions, composition of each phase, and thermodynamic properties for each phase) are described by `P_Phase_Point` Class. General information about the system, such as alloying components, alloy overall composition and temperature, are stored in `P_Statespace` class. The details on these and other classes can be found in the C++ header files of `Pan_Global_Def.h` and `Pan_Global.h`.

In the following, we will first give a brief introduction of thermodynamic models and then describe the different classes used in `PanEngine`.

3.1 Gibbs Energy Models for Multi-Component Phases

3.1.1 Stoichiometric compound

The Gibbs energy of a stoichiometric phase is expressed as

$$G = \sum_{i=1}^{n} x_i G_i^{\circ,\phi} + G_f \tag{1}$$

where x_i is the mole fraction of component i, $G_i^{\circ,\phi}$ is the Gibbs energy of the pure component i with structure ϕ, and G_f is the Gibbs energy of formation of the stoichiometric phase referred to the structure ϕ for each component i.

3.1.2 Disordered solution phase

The Gibbs energy of a disordered solution phase is expressed as

$$G = \sum_{i=1}^{n} x_i G_i^{\circ,\phi} + RT \sum_{i=1}^{n} x_i \ln x_i + G^{ex,\phi} \tag{2}$$

where x_i is the mole fraction of component i, $G_i^{\circ,\phi}$ is the Gibbs energy of the pure component i with structure ϕ, R is the gas constant, and T is the absolute temperature. $G^{ex,\phi}$ is the excess Gibbs energy of the phase, defined as

$$G^{ex,\phi} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_i x_j \sum_{l=0}^{m} L_{ij}^{(l)} (x_i - x_j)^l + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} x_i x_j x_k \sum_{l=i,j,k} L_{ijk}^{(l)} V_{ijk}^{(l)} \tag{3}$$

where the first term represents the binary interaction terms, the second represents ternary interactions. The $L_{ij}^{(l)}$'s are binary interaction parameters for the i-j binary and the $L_{ijk}^{(l)}$'s are ternary interaction parameters. $V_{ijk}^{(l)}$ is defined as

$$V_{ijk}^{(l)} = x_l + \frac{1 - x_i - x_j - x_k}{n} \qquad (l = i, j, k) \tag{4}$$

For a ternary system,

$$V_{ijk}^{(l)} = x_l \qquad (l = i, j, k) \tag{5}$$

since $x_i + x_j + x_k = 1$.

3.1.3 Ordered intermetallic phase using the compound energy formalism

The Gibbs energy of an ordered intermetallic phase is described as

$$G = G^{ref} + G^{id} + G^{ex} \tag{6}$$

where G^{ref} is expressed in terms of compound energies (which are constant at constant temperature) and their associated sublattice species concentrations, y_p^i,

$$G^{ref} = \sum y_p^i y_q^j \cdots y_s^l G_{p:q:\cdots:s} \tag{7}$$

G^{id} is the ideal mixing term, which assumes the random mixing of species on each sublattice,

$$G^{id} = \sum_{i=1}^{l} f_i \sum_{p=1}^{m} y_p^i \ln y_p^i \tag{8}$$

G^{ex} is also expressed as a function of species concentrations with the sublattice L parameters being the numerical coefficients in the contributing terms,

$$G^{ex} = \sum y_p^i y_q^i y_r^j L_{p,q:r} \tag{9}$$

where

$$L_{p,q:r} = \sum_v L_{p,q:r}^v (y_p^i - y_q^i)^v \tag{10}$$

3.2 PanEngine Classes

`PanEngine` is a dynamically linked library of thermodynamic and phase equilibrium calculation functions. Most of the communications between the user's application code and `PanEngine` are through the objects of `PanEngine` classes as mentioned at the beginning of this chapter. The headers of `PanEngine` classes are included in the files `Pan_Global.h` and `Pan_Global_def.h`. The major `PanEngine` classes with frequently used functions and properties are described below. Please refer to the header files of `Pan_Global.h` and `Pan_Global_def.h`. for other classes.

3.2.1 class P_Component

Class Name	P_Component
Definition	a component is made up of one or more elements, for example: Al or FeO
Public Functions	P_Component() virtual ~P_Component() (and other copy constructors and operators)
Public Properties	string m_name // component name int m_id // component ID (see Pan_Global_Def.h for other member variables)
Comments	P_Component holds information for a component

3.2.2 class P_Species

Class Name	P_Species
Definition	species can be made up of one or more components, for example: O2
Public Functions	P_Species() virtual ~P_Species() (and other copy constructors and operators)
Public Properties	string m_name // species name vector<pair<string, double>>m_c // first:component name; second:amount of component (see Pan_Global_Def.h for other member variables)
Comments	associate model uses P_Species to define the species of the associates

3.2.3 class P_Statespace

Class Name	P_Statespace
Definition	Statespace describes temperature, pressure and composition of a system or a phase
Public Functions	P_Statespace() virtual ~P_Statespace() (and other copy constructors and operators)

Public Properties	double m_T; // in K, system temperature
	double m_P; // in pascal, system pressure
	map<string, shared_ptr<P_Component>>m_comp
	// collection of components
	(see Pan_Global_Def.h for other member variables)

Comments	most of calculation conditions are set through this class

3.2.4 class P_Phase_Point

Class Name	P_Phase_Point
Definition	information for a phase after a calculation: the state space, thermodynamic properties, species concentrations, etc.
Public Functions	P_Phase_Point() virtual ~P_Phase_Point() (and other copy constructors)
Public Properties	string m_phase_name // phase name int m_phase_id // phase ID (see Pan_Global_Def.h for other member variables)
Comments	the objects of this class will be created by PanEngine after calculation

3.2.5 class P_Point

Class Name	P_Point
Definition	an equilibrium state with one or more phase points (P_Phase_Point)
Public Functions	P_Point() virtual ~P_Point() P_Point(const P_Point&) (and other copy constructors and member functions)
Public Properties	shared_ptr<P_Statespace> m_st // statespace for a P_Point vector <shared_ptr<P_Phase_Point>>m_ppt; // phase point in this Point (see Pan_Global_Def.h for other member variables)
Comments	for solidification, a P_Point includes the fractions of solid and liquid

4 PanEngine API

The functions of the `PanEngine` application program interface (API) are defined as virtual functions in a class of `PanEngine` in `PanEngine.h`, except for the two global functions used for defining a `PanEngine` pointer and deleting an existing `PanEngine` pointer. These functions can be divided into four categories according to their purposes:

- `PanEngine` Pointer
  - define a `PanEngine` pointer and initialize it
  - delete an existing `PanEngine` pointer
- System
  - set system configuration
  - import a thermodynamic database
  - save a subsystem database
  - get system component names
  - get system phase names
  - get active phase names
  - set phase status
  - get phase status
  - activate a phase
  - deactivate a component
  - set calculation condition
- Point Calculation
  - find globally stable equilibrium
  - find globally stable equilibrium with initial
  - find metastable (local) equilibrium with initial
  - find liquidus surface
  - calculate liquidus slopes
- Solidification Simulation
  - lever rule model
  - Scheil model

Table 2 summarizes the functions of `PanEngine` API. These functions will be explained in detail in the following sections.

Table 2: List of PanEngine API Functions

Functions		Comments
User Pointer	extern "C" PANENGINE_API PanEngine* definePanEngineUser(char* msg)	define a PanEngine pointer
	extern "C" PANENGINE_API void deletePanEngineUser(PanEngine *pUser)	delete a PanEngine pointer
System	string pe_set_configuration(map<string, string>& config)	set system configuration
	string pe_import_database(pair<string, string>& db, bool append=false, const pair<string, string>& db_to=pair<string, string>())	import thermodynamic database
	string pe_export_subsystem_database(pair<string, string>& db, const string subsystem_name=string("sub.tdb"), const vector<string>& comp_name = vector<string>())	save a subsystem database
	string pe_get_component_names(pair<string, string>& db, vector<string>& comp_name)	get component names in a database
	string pe_deactivate_component(string& comp_name)	suspend a component
	string pe_get_phase_names(pair<string, string>& db, vector<string>& phase_name, vector<string> comp_name = vector<string>())	get phase names in a database with a given set of components
	string pe_get_phase_status(pair<string, string>& db, vector<pair<string, PAN_PHASE_STATUS>>& phase_name_status)	get phase status in a database
	string pe_set_phase_status(pair<string, string>& db, vector<pair<string, PAN_PHASE_STATUS>>& phase_name_status)	set phase status in a database
Point Calculation	string pe_set_calculation_condition(const Pan_Calculation& calc)	set a calculation condition
	string pe_calc_point_global(shared_ptr<P_Point> p_pt)	calculate a global phase equilibrium for a point
	string pe_calc_point_global_with_initial_point (shared_ptr<P_Point> p_pt)	calculate a global phase equilibrium for a point with initials
	string pe_calc_point_local_with_initial_point (shared_ptr<P_Point> p_pt, bool given_f=false)	calculate a local phase equilibrium or parallel tangent equilibrium for a point with initials
	string pe_find_liquidus_surface(string& liquid_phase_name, shared_ptr<P_Point> p_pt)	find a liquidus surface
	string pe_calc_liquidus_slope(string& liquid_phase_name, string& solvent_comp_name, shared_ptr<P_Point> p_pt)	calculate liquidus slopes
Solidification Simulation	String pe_solidification_simulation (Solidification_Parameter& s_param, Pan_Calculation& calc, vector<Solidification_Node>& solidification_result)	simulate solidification with lever rule or Scheil model

4.1 Functions for PanEngine Pointer

There are two functions associated with the `PanEngine` pointer: define a `PanEngine` pointer and delete a `PanEngine` pointer. These two functions are global functions.

4.1.1 Define a PanEngine pointer

Name	`extern "C" PANENGINE_API PanEngine* definePanEngineUser(char* msg)`
Purpose	define a `PanEngine` pointer and initialize it
Arguments	`msg` message returned from `PanEngine`

A `PanEngine` pointer must be successfully initialized before `PanEngine`'s other functions can be used. If the CompuTherm dongle is not attached to the computer, the initialization will fail.

4.1.2 Delete a PanEngine pointer

Name	`extern "C" PANENGINE_API void deletePanEngineUser(PanEngine *pUser)`
Purpose	delete a `PanEngine` pointer after all calculations are done
Arguments	`PanEngine *pUser` a defined and initialized `PanEngine` pointer

After a `PanEngine` pointer `pUser` is deleted, the system information inside `PanEngine` pointed to by `pUser` will be deleted and `pUser` will be a `null` pointer.

4.2 Functions for System

`PanEngine` API functions in the system level manage the system related information, such as importing a thermodynamic database and setting calculation conditions.

4.2.1 Set system configuration

Name	`string pe_set_configuration(map<string, string>& config)`
Purpose	set system configuration
Arguments	`config` a map of pair of strings to define a configuration

One of the configurations is case sensitive of component names and phase names in a database. This can be set with this function as:

```
map<string, string> config;
config["case_sensitive"] = "false";
s = user->pe_set_configuration(config);
```

which will convert all component and phase names into capital letters while reading the database.

4.2.2 Import a thermodynamic database

Name	`string pe_import_database(pair<string, string>& db, bool append=false, const pair<string, string>& db_to=pair<string, string>())`
Purpose	import a thermodynamic database file in `tdb` format
Arguments	`db` database file name; `append` append the database `db` to the database `db_to`

Thermodynamic parameters are stored in files. The tdb type of file is a text file which can be modified by the user using a text editor. The pdb type of file is an encrypted database. If `append=false`, import the database in the file of db. If `append=true`, append the database in the file of `db` to the already imported database from the file of `db_to`.

4.2.3 Export a subsystem database into a file (`tdb`)

Name	`string pe_export_subsystem_database (pair<string, string>& db, const string subsystem_name=string("sub.tdb"), const vector<string>& comp_name = vector<string>())`
Purpose	export a subsystem thermodynamic database in `tdb` format into a file
Arguments	`db` database file name; `subsystem_name` subsystem database file name; `comp_name` component names in the subsystem

After a thermodynamic database (`tdb`) is successfully loaded, a subsystem with selected components can be exported into a database file with `tdb` format.

4.2.4 Get component names in a database

Name	`string pe_export_subsystem_database (pair<string, string>& db, const string subsystem_name=string("sub.tdb"), const vector<string>& comp_name = vector<string>())`
Purpose	get all component names in a database
Arguments	`db` database file name; `comp_name` component names

This function gets all component names in a database with the file name of `db`.

4.2.5 Deactivate a component in a database

Name	`string pe_deactivate_component(string& comp_name)`
Purpose	Deactivate a component in a database
Arguments	`comp_name` component to be deactivated

This function deactivates a component with the name of `comp_name` in the current database.

4.2.6 Get phase names in a database

Name	`string pe_get_phase_names(pair<string, string>& db, vector<string>& phase_name, vector<string> comp_name = vector<string>())`
Purpose	get phase names in a database with a given set of components
Arguments	`db` database file name; `phase_name` phase names; `comp_name` selected component names

If `comp_name` is given, this function gets the phase names in the subsystem with the components of `comp_name` in a database with the file name of `db`. Otherwise, the function gets all phase names in a database with the file name of `db`.

4.2.7 Get and set phase statuses

Name	`string pe_get_phase_status(pair<string, string>& db, vector<pair<string, PAN_PHASE_STATUS>>& phase_name_status)`
Purpose	get phase status in a database
Arguments	`db` database file name; `phase_name_status` vector of phase's name and its status

Name	`string pe_set_phase_status(pair<string, string>& db, vector<pair<string, PAN_PHASE_STATUS>>& phase_name_status)`
Purpose	set phase status in a database
Arguments	`db` database file name; `phase_name_status` vector of phase's name and its status

These two functions get and set the phase statuses in a database. Phase status takes values of `P_PHASE_ENTERED`, `P_PHASE_SUSPENDED`, `P_PHASE_DORMANT`, `P_PHASE_FIXED`, `P_PHASE_STATUS_NOT_DEFINED`. See Pandat User's Guide for definition of the phase status.

4.3 Functions for Point Calculation

`PanEngine` uses a specially designed global optimization algorithm to find the most stable phase equilibrium automatically without guessing initial values. It also provides functions for performing locally metastable phase equilibrium calculations and other types of calculations. The point related calculations in `PanEngine` API are described below.

4.3.1 Set calculation condition

Name	`string pe_set_calculation_condition(const Pan_Calculation& calc)`
Purpose	set a calculation condition
Arguments	`calc` calculation condition object

Calculation condition defines database to be used, units, selected components and phases, state space `(T, P, xj)`. See test examples for detail.

4.3.2 Calculate a global phase equilibrium

Name	`string pe_calc_point_global(shared_ptr<P_Point> p_pt)`
Purpose	calculate a global phase equilibrium for a point
Arguments	`p_pt` a `shared_ptr` of `P_Point` to be calculated

This function calculates the global phase equilibrium according to the calculation condition. Information on the calculated phase equilibrium is stored in `p_pt`.

4.3.3 Calculate a global phase equilibrium with initials

Name	`string pe_calc_point_global_with_initial_point(shared_ptr<P_Point> p_pt)`
Purpose	calculate a global phase equilibrium for a point with initials
Arguments	`p_pt` a `shared_ptr` of `P_Point` to be calculated

This function calculates the global phase equilibrium with the initial conditions in `p_pt`. Information on the calculated phase equilibrium is also stored in `p_pt`. With the initial values in `p_pt`, the computational speed is usually faster.

4.3.4 Calculate a local phase equilibrium

Name	`string pe_calc_point_local_with_initial_point(shared_ptr<P_Point> p_pt, bool given_f=false)`
Purpose	calculate a local phase equilibrium or parallel tangent equilibrium for a point with initials
Arguments	`p_pt` a `shared_ptr` of `P_Point` to be calculated `given_f`: `false` for a local phase equilibrium and `true` for a parallel tangent equilibrium

This function calculates the local phase equilibrium with the initial condition in `p_pt`. Initial values in `p_pt` is required for this function. The calculated phase equilibrium is stored in `p_pt`.

4.3.5 Find the liquidus surface

Name	`string pe_find_liquidus_surface(string& liquid_phase_name, shared_ptr<P_Point> p_pt)`
Purpose	find a liquidus surface point for given composition
Arguments	`liquid_phase_name` phase name of `liquid`; `p_pt` a `shared_ptr` of `P_Point` for liquid and primary phases

This function calculates the liquidus surface temperature for a point of with fixed composition. `PanEngine` will find the stable liquidus surface.

4.3.6 Find the liquidus slopes

Name	`string pe_calc_liquidus_slope(string& liquid_phase_name, string& solvent_comp_name, shared_ptr<P_Point> p_pt)`
Purpose	calculate liquidus slopes on the liquidus surface
Arguments	`liquid_phase_name` phase name of liquid; `solvent_comp_name name of solvent component`; `p_pt` a `shared_ptr` of `P_Point` for liquid with primary phase

This function calculates the liquidus slope along the directions of components on the liquidus surface for a point with given composition. The slope along the direction of component j is defined as

$$s_j = \left(\frac{\partial T^{liq}}{\partial x_j}\right)_{x_i, i\neq j, j\neq solvent} \tag{11}$$

where T^{liq} is the liquidus surface temperature and x_j is the mole fraction for the specified component j. Since the molar fractions of components are dependent with each other by $\sum_k x_k = 1$, the solvent component must be specified. The slope along the direction of the solvent component will be treated as zero, $s_{solvent} = 0$.

For example, in a ternary A-B-C system, if the component A is selected as the solvent component, the slop of the liquidus surface along the direction of the component B is

$$s_B = \left(\frac{\partial T^{liq}}{\partial x_B}\right)_{x_C} \tag{12}$$

and the slope along the direction of the component C is

$$s_C = \left(\frac{\partial T^{liq}}{\partial x_C}\right)_{x_B} \tag{13}$$

The temperature change δT caused by the composition change of (δx_B, δx_C) will be calculated by

$$\delta T = s_B \delta x_B + s_C \delta x_C \tag{14}$$

The slopes in terms of weight fractions are also available, see test examples for detail.

Since PanEngine 2019, two new properties, `dxdT` and `dwdT`, have been added into `P_Component` for solidification simulation. `dxdT` and `dwdT` represent the change rates of the molar fraction and weight fraction for a component in a phase with temperature during a solidification, respectively. These two variables can be found in the definition of class `P_Component` in the head file of `Pan_Global_Def.h`.

Since PanEngine 2020, another set of properties, `dxdT_S` and `dxdT_L`, and the corresponding properties in weight fraction, `dwdT_S` and `dwdT_L`, have been added into `P_Component` for solidification simulation. `dxdT_S` and `dxdT_L` represent the change rates of the molar fraction for a component in a solid phase and the liquid phase with temperature during a solidification, respectively, assuming that the solid phase is the only phase solidified from the liquid. In other words, these properties are calculated for the (local) equilibrium between the liquid phase and the only solid phase, excluding other solid phases even though they exist. Therefore, when there is more than one solid phase solidified from liquid, `dxdT_S` and `dxdT_L` will have different values from `dxdT`. `dxdT_S`, `dxdT_L`, `dwdT_S` and `dwdT_L` are stored in the solid phase only. These variables can also be found in the definition of class `P_Component` in the head file of `Pan_Global_Def.h`.

When temperature decreases by δT during solidification, the composition changes of the solid phase and the liquid phase can be calculated by

$$\delta x_j^s = \texttt{dxdT_S}\, \delta T \qquad (j = 1, 2, \cdots, c) \tag{15}$$

$$\delta x_j^l = \texttt{dxdT_L}\, \delta T \qquad (j = 1, 2, \cdots, c) \tag{16}$$

Test Example 4 has the printout of those properties in the callback function, `progress_4`.

4.4 Function for Solidification Simulation

`PanEngine` has another API function for solidification simulations. There are two models available: lever rule and Scheil.

Name	`String pe_solidification_simulation (Solidification_Parameter& s_param, Pan_Calculation& calc, vector<Solidification_Node>& solidification_result)`
Purpose	simulate solidification with lever rule or Scheil model
Arguments	`s_param` parameters for solidification simulation; `calc` calculation condition; `solidification_result` solidification results

Details on how to use this function will be demonstrated in test Examples 4 and 5.

4.5 Thermodynamic Factors

In Pandat, thermodynamic factors are available with the format `ThF(*,*)`, which is defined by

$$\texttt{ThF}(\texttt{x}_\texttt{i},\texttt{x}_\texttt{j}) = \frac{\partial \mu_i}{\partial x_j} \qquad (i,j = 1,2,\cdots,c) \tag{17}$$

where $(x_1, x_2, \cdots, x_c)$ are treated as the independent compositional variables. If the component 1 is assumed to be the solvent component and its molar fraction x_1 is taken as the dependent variable, the independent compositional variables now are $(X_2, X_3, \cdots, X_c)$. Here we use capital X to distinguish this set of variables from $(x_1, x_2, \cdots, x_c)$. Then we have

$$\frac{\partial \mu_j}{\partial X_k} = \frac{\partial \mu_j}{\partial x_k} - \frac{\partial \mu_1}{\partial x_k} \qquad (j,k = 2,3,\cdots,c) \tag{18}$$

$$= \texttt{ThF}(\texttt{x}_\texttt{j},\texttt{x}_\texttt{k}) - \texttt{ThF}(\texttt{x}_\texttt{1},\texttt{x}_\texttt{k}) \qquad (j,k = 2,3,\cdots,c) \tag{19}$$

Now let's see how to express the second derivatives of Gibbs free energy in terms of the thermodynamic factors. If we use this compositional variable set $(X_2, X_3, \cdots, X_c)$, the first and second derivatives of G w.r.t. X_j are

$$\frac{\partial G}{\partial X_j} = \sum_{i=2}^{c} \frac{\partial G}{\partial x_i}\frac{\partial x_i}{\partial x_j} + \frac{\partial G}{\partial x_1}\frac{\partial x_1}{\partial x_j} \qquad (j = 2,3,\cdots,c) \tag{20}$$

$$= \frac{\partial G}{\partial x_j} - \frac{\partial G}{\partial x_1} \qquad (j = 2,3,\cdots,c) \tag{21}$$

$$= \mu_j - \mu_1 \qquad (j = 2,3,\cdots,c) \tag{22}$$

$$\frac{\partial^2 G}{\partial X_j \partial X_k} = \sum_{i=2}^{c} \frac{\partial(\mu_j - \mu_1)}{\partial x_i}\frac{\partial x_i}{\partial x_k} + \frac{\partial(\mu_j - \mu_1)}{\partial x_1}\frac{\partial x_1}{\partial x_k} \qquad (j,k = 2,3,\cdots,c) \tag{23}$$

$$= (\frac{\partial \mu_j}{\partial x_k} - \frac{\partial \mu_1}{\partial x_k}) - (\frac{\partial \mu_j}{\partial x_1} - \frac{\partial \mu_1}{\partial x_1}) \qquad (j,k = 2,3,\cdots,c) \tag{24}$$

$$= \texttt{ThF}(\texttt{x}_\texttt{j},\texttt{x}_\texttt{k}) - \texttt{ThF}(\texttt{x}_\texttt{1},\texttt{x}_\texttt{k}) - \texttt{ThF}(\texttt{x}_\texttt{j},\texttt{x}_\texttt{1}) + \texttt{ThF}(\texttt{x}_\texttt{1},\texttt{x}_\texttt{1}) \qquad (j,k = 2,3,\cdots,c) \tag{25}$$

Thermodynamic factors, `ThF(*,*)`, are stored in the member variable of `m_thermodynamic` in the class `P_Phase_Point` in the head file of `Pan_Global_Def.h`. Test Example 8 shows the thermodynamic factors of the components in each stable phase point.

4.6 Hessian matrix of Gibbs free energy

From the second derivatives of Gibbs free energy in previous section, Pandat has the Hessian matrix of Gibbs free energy of a phase. Pandat also calculates the determinant, the eigenvalues and eigenvectors of the Hessian matrix.

Since there is one dependent molar fraction for the molar fraction variables $(x_1, x_2, \cdots, x_n)$, one of the components is selected as the dependent one. Without loss of generality, x_n is selected as the one, i.e., the

last component is considered as the solvent. Then, the second derivatives of Gibbs free energy of a phase form the Hessian matrix, which is an (n-1) × (n-1) symmetrical matrix.

$$\texttt{HSN} = \begin{pmatrix} \frac{\partial^2 G}{\partial x_1^2} & \frac{\partial^2 G}{\partial x_1 \partial x_2} & \frac{\partial^2 G}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_1 \partial x_{n\text{-}1}} \\ \frac{\partial^2 G}{\partial x_2 \partial x_1} & \frac{\partial^2 G}{\partial x_2^2} & \frac{\partial^2 G}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_2 \partial x_{n\text{-}1}} \\ \frac{\partial^2 G}{\partial x_3 \partial x_1} & \frac{\partial^2 G}{\partial x_3 \partial x_2} & \frac{\partial^2 G}{\partial x_3^3} & \cdots & \frac{\partial^2 G}{\partial x_3 \partial x_{n\text{-}1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 G}{\partial x_{n\text{-}1} \partial x_1} & \frac{\partial^2 G}{\partial x_{n\text{-}1} \partial x_2} & \frac{\partial^2 G}{\partial x_{n\text{-}1} \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_{n\text{-}1}^2} \end{pmatrix} \tag{26}$$

Its determinant is given by

$$|\texttt{HSN}| = \begin{vmatrix} \frac{\partial^2 G}{\partial x_1^2} & \frac{\partial^2 G}{\partial x_1 \partial x_2} & \frac{\partial^2 G}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_1 \partial x_{n\text{-}1}} \\ \frac{\partial^2 G}{\partial x_2 \partial x_1} & \frac{\partial^2 G}{\partial x_2^2} & \frac{\partial^2 G}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_2 \partial x_{n\text{-}1}} \\ \frac{\partial^2 G}{\partial x_3 \partial x_1} & \frac{\partial^2 G}{\partial x_3 \partial x_2} & \frac{\partial^2 G}{\partial x_3^3} & \cdots & \frac{\partial^2 G}{\partial x_3 \partial x_{n\text{-}1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 G}{\partial x_{n\text{-}1} \partial x_1} & \frac{\partial^2 G}{\partial x_{n\text{-}1} \partial x_2} & \frac{\partial^2 G}{\partial x_{n\text{-}1} \partial x_3} & \cdots & \frac{\partial^2 G}{\partial x_{n\text{-}1}^2} \end{vmatrix} \tag{27}$$

The determinant of Hessian matrix for phase `f` is available from `HSN(@f)`. The value of `HSN(@f)` is independent of the selection of the solvent component. However, eigenvalues and eigenvectors are dependent on the selection of the solvent component.

A Hessian matrix has real eigenvectors and each eigenvalue has a corresponding eigenvector. The eigenvalues and their eigenvectors are available from `eVal(#*@f)` and `eVec(*#*@f)`.

Above Hessian matrix has eigenvalues of `eVal(#1@f)`, `eVal(#2@f)`, $\cdots$, `eVal(#n-1@f)`. Each eigenvalue has an eigenvector. For example, `eVal(#1@f)` has an eigenvector of (`eVec(`$\texttt{C}_1$`#1@f)`, `eVec(`$\texttt{C}_2$`#1@f)`, $\cdots$, `eVec(`$\texttt{C}_{n-1}$`#1@f)`), where $\texttt{C}_k$ is the name of the k^{th} component.

Test Examples 1 and 8 show how to get the values related to the Hessian matrix of Gibbs free energy of a phase.

4.7 Parallel Tangent Equilibrium

In the phase field modeling of microstructure evolution, the equilibrium among phases at a interface is not considered as a true phase equilibrium, where same component has same equilibrium chemical potential, i.e., common tangent (See Fig. 1(a)). Instead, the parallel tangent equilibrium is used at the interface during phase field modeling.

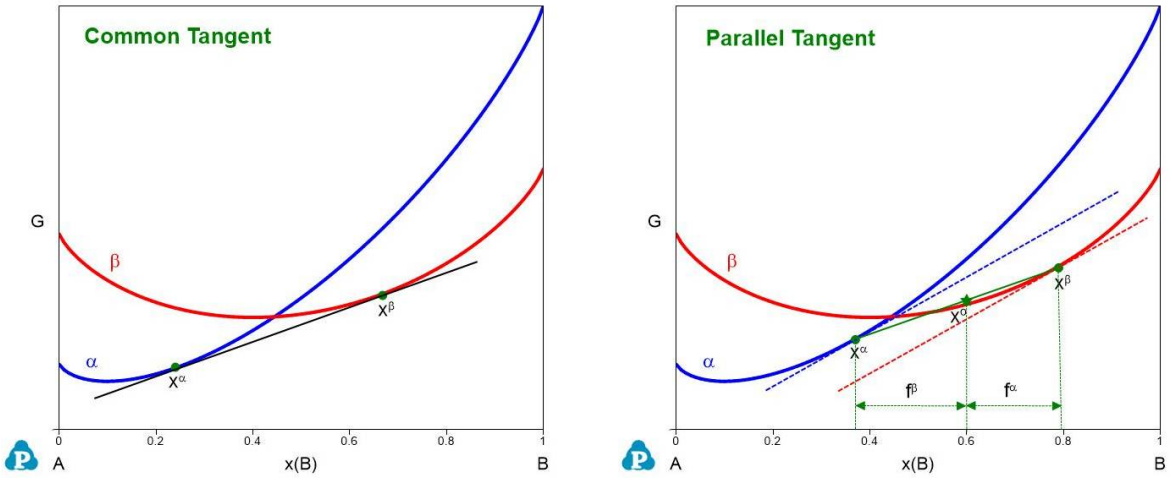
Parallel tangent equilibrium assumes that the difference of the chemical potential of same component in the phases at interface are same for all components, i.e.,

$$\mu_i^\alpha - \mu_i^\beta = \mu_j^\alpha - \mu_j^\beta \qquad (i, j = 1, 2, \cdots, c) \tag{28}$$

This constraint is equalvanlent to that the tangent lines to Gibbs free energy curves of phases are parallel to each other, see Fig. 1(b). Therefore, this type of equilibrium is called parallel tangent equilibrium. For the

example of a parallel tangent equilibrium in Fig. 1(b), the overall composition, x°, and the phase fractions, f^α and f^β, are given and then the parallel tangents (dash lines) are calculated to get the equilibrium composition of each phase, x^α and x^β.

Test Example 9 shows how to calculate a parallel tangent equilibrium.



(a) Common tangent equilibrium

(b) Parallel tangent equilibrium

Figure 1: Two types of equilibria

5 Examples

This chapter demonstrates how to use `PanEngine` API functions with examples. There are seven test examples. The main program (`main.cpp`) is for user to select which example to run. All the source files (`*.cpp`) can be found in the folder `/source/`. Following sections give a brief review of each example.

5.1 Test Example 1

This first test example is in file of `PanEngineTest_1.cpp`. It uses a number of point calculations to demonstrate the following functions:

```
//  Define a PanEngine user pointer
user = definePanEngineUser(msg);

//  Set PanEngine configuration
s = user->pe_set_configuration(config);

//  Import a database
s = user->pe_import_database(db);

// Set calculation properties
s = user->pe_set_calc_property_type(calc_prop);

//  Set calculation condition
s = user->pe_set_calculation_condition(calc);

//  Calculate a global phase equilibrium
s = user->pe_calc_point_global(calc_point);

//  Calculate a global phase equilibrium with initial
s = user->pe_calc_point_global_with_initial_point(calc_point);

//  Delete a PanEngine user pointer
deletePanEngineUser(user);
```

It demonstrates how to set up calculation conditions. The example shows how to get the physical properties of molar volume and density. The Hessian matrix of Gibbs free energy of a phase, its determinant, eigenvalues and corresponding eigenvectors are also available. After `sys_ptr->m_condition->m_driving_force` is set to be `true`, the driving force for a phase with respect to an equilibrium state can be obtained.

5.2 Test Example 2

The second test example is in file of `PanEngineTest_2.cpp`. It mainly demonstrates how to use a "`for`" loop to calculate multiple points using the function `pe_calc_point_global_with_initial_point`. However, the first point has to be calculated with the function `pe_calc_point_global` to get initial point.

5.3 Test Example 3

The third test example is in file of `PanEngineTest_3.cpp`. It demonstrates how to calculate local phase equilibria for a point at different temperatures and multiple points with different compositions using the function of `pe_calc_point_local_with_initial_point`. Similar to the previous example (Test Example 2), the first point has to be calculated with the function `pe_calc_point_global` to get initial point.

5.4 Test Example 4

The fourth test example is in file of `PanEngineTest_4.cpp`. Most of the codes in this file are for setting solidification conditions and parameters, such as database name, units, alloy compositions, solvent component, solidification model, and step size. The major function used in this test example is `pe_solidification_simulation`. The solidification result could be output to a file with the function

```
string output_result(string file_name, vector<Solidification_Node>& solidification_result)
```

which is defined in this example file.
The function

```
bool progress_4(char* msg, Solidification_Parameter& sp, vector<Solidification_Node>& node)
```

is a callback function which will be called from `PanEngine` to send back the intermediate results during simulation.

5.5 Test Example 5

The fifth test example is in file of `PanEngineTest_5.cpp`. This example is also for solidification simulation and similar to the previous one. It demonstrates how to simulate solidification processes for multiple alloys with different compositions.
The function

```
bool progress_5(char* msg, Solidification_Parameter& sp, vector<Solidification_Node>& node)
```

is the callback function.

5.6 Test Example 6

The sixth test example is in file of `PanEngineTest_6.cpp`. This example demonstrates how to find the liquidus surface and calculate liquidus slops using the following two functions:

```
s=user->pe_find_liquidus_surface(liquid_name, calc_point)
```

and

```
s=user->pe_calc_liquidus_slope(liquid_name, solvent_name, p_pt)
```

The example uses a double "for" loops to calculate the liquidus surface points and slops for a series of points in the Al-Mg-Zn system.

5.7 Test Example 7

The seventh test example is in file of PanEngineTest_7.cpp. This example uses three stages to locate a point on the liquidus surface in the Al-Mg-Zn ternary system where the temperature reaches the minimum. The first stage of search uses the step size of dx=0.1. The second stage uses step size of dx=0.01 to search in the compositional neighborhood of the found composition in the first stage. And the third stage uses the step size of dx=0.001 to search in the compositional neighborhood of the found composition in the second stage. Even through this method is of brute-force, it works fine for such a simple problem. The major function used in this example is pe_find_liquidus_surface.

5.8 Test Example 8

The eighth test example is in file of PanEngineTest_8.cpp. This example demonstrates how to obtain the thermodynamic properties such as thermodynamic factors and Hessian matrix, and the kinetic properties such as mobility and diffusivity. The example calculates the thermodynamic factor, Hessian matrix, mobility, and diffusivity for an alloy in the Fe-Ni-Cr system. For the calculation of Hessian matrix, mobility and diffusivity, the name of the solvent component has to be defined, as is given below,

```
sys_ptr->m_condition->m_solvent = string("Ni");
```

which is in the function of set_calculation_condition_8.

The example shows how to extract those properties from a calculated point.

5.9 Test Example 9

This last test example is in file of PanEngineTest_9.cpp. This example demonstrates how to calculate a parallel tangent equilibrium.

The example calculates the two-phase parallel tangent equilibrium between Fcc and gamma_double_prime (γ'') in a pseudo-ternary Ni-Al-Nb. The common tangent equilibrium is first calculated from

```
s = user->pe_calc_point_global(calc_point);
```

to get the initial values of the parallel tangent equilibrium. Then, reset the phase fractions of the two phases by

```
calc_point->m_ppt[0]->m_f = 0.5;
calc_point->m_ppt[1]->m_f = 0.5;
```

Call the function of calculating point local with initial point

```
s = user->pe_calc_point_local_with_initial_point(calc_point, true);
```

with a `true` value for the second argument to calculate the parallel tangent equilibrium. When the second argument is `true`, the function will calculate the parallel tangent equilibrium. If the second argument is `false`, the function will calculate the normal local common tangent equilibrium.

Example 9 continues with a set of randomly generated temperatures, overall compositions and phase fractions to calculate the parallel tangent equilibria.

CompuTherm, LLC

Copyright© 2012-2021