ARL-TR-10169 ● SEP 2025

Python Module for Predicting Interfacial Geometric Energy (PIG-E): Code Documentation

by Heather Murdoch, Benjamin Szajewski, Efrain Hernandez, Matthew Guziewski, and Daniel Magagnosc

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-10169 ● SEP 2025

Python Module for Predicting Interfacial Geometric Energy (PIG-E): Code Documentation

Heather Murdoch, Benjamin Szajewski, Efrain Hernandez, Matthew Guziewski, and Daniel Magagnosc
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
September 2025	Technical Report	20 February 2024	30 September 2025

4. TITLE AND SUBTITLE
Python Module for Predicting Interfacial Geometric Energy (PIG-E): Code Documentation

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Heather Murdoch, Benjamin Szajewski, Efrain Hernandez, Matthew Guziewski, and Daniel Magagnosc

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLA-MF
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-10169

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCIDs: Heather Murdoch, 0000-0001-7710-0577; Matthew Guziewski, 0000-0002-5761-720X; Efrain Hernandez, 0000-0002-7855-1027; Daniel Magagnosc, 0000-0002-1418-9292; Benjamin Szajewski, 0000-0002-5028-414X

14. ABSTRACT
A key parameter when modeling precipitation is the interfacial energy between the matrix and precipitate. The interfacial energy comprises two contributions, a chemical component and an elastic strain energy component. Thermodynamic modeling approaches (i.e., CALPHAD) are able to predict the chemical contribution. However, accounting for the elastic strain energy remains a challenge. Here, a Python code is developed to implement a dislocation-based approach for determining the elastic strain energy contribution. The code, which was developed for high-throughput computations, requires crystallographic data, elastic constants, and an interface orientation relationship for the matrix and precipitate. From these inputs, a bimaterial interface is described using a transformation matrix, which transforms the precipitate crystal lattice to the matrix crystal lattice, and candidate Burgers vectors for the interface. The dislocation density of the interface is then minimized, and the corresponding elastic strain energy is determined using a non-singular approximation of the dislocation strain field.

15. SUBJECT TERMS
interfacial energy, dislocation dynamics, carbides, Python, CALPHAD, high-throughput, Sciences of Extreme Materials

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	52

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Heather Murdoch	(520) 691-6293

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

This work is in collaboration with and in support of the High-Throughput Materials Discovery for Extreme Conditions (HTMDEC) Collaborative Research Alliance.

1. Introduction

The interfacial energy between a matrix and precipitate is a key factor in modeling the evolution of precipitates during heat treatment. Accurately capturing both the size and volume fraction evolution of nanoscale precipitates is critical for predicting the mechanical properties of precipitation strengthened alloys, such as secondary hardening ultrahigh strength steels, heat-treatable aluminum alloys, and complex concentrated alloys. Most kinetic modeling of precipitate evolution focuses on the behavior of the expected strengthening phase, but it can also be insightful to model the competition of various phases. This will be especially true when designing new alloys in novel design spaces in which there is more uncertainty in the phase evolution. Despite the significance of multiple phases in precipitation strengthened alloys, obtaining reliable interfacial energies as inputs to nucleation and growth models remains a challenge.

Generating interfacial energies from experiments is prohibitively challenging. Such measurements rely on extremely high-fidelity observations of precipitate size evolution through transmission electron microscopy, atom probe tomography, or X-ray diffraction. Further complicating matters, the interfacial energy is not directly measured. Instead, it is inferred from the size evolution of the precipitates, thereby requiring the time-consuming preparation of a series of samples at potentially long heat treatment times.

Computational approaches have proven effective in capturing nucleation and growth processes. A range of methods, including atomistic simulations (density functional theory [DFT], molecular dynamics, Monte Carlo) and phase field models, are employed to study nucleation with high fidelity.[1–3] However, each of these methods has drawbacks and limitations: large computational demands, limited simulation cell size, short simulation time scales, or the requisite interatomic potential. These tradeoffs effectively limit their utility for high-throughput modeling of new alloy systems.

To overcome these restrictions, a data-driven, mesoscale dislocation model is implemented for high-throughput interfacial energy calculations. The details of the theory are in Szajewski et al.[4] and an example dataset of parameters for 12 common carbides in steel, as well as the methods for generating the dataset, is described in Murdoch et al.[5]

Here we document the Python code Predicting Interfacial Geometric Energy (PIG-E), which calculates the elastic energy contribution to interfacial energy. First, the structure of PIG_E is described followed by the implementation of the carbide dataset as an input module called Steel_Info.py.

2. Calculation Overview

The elastic strain energy calculations are described in more detail below, but the necessary inputs describing the bimaterial interface are summarized here:

- Shear moduli and Poisson's ratios (i.e., the elastic properties) for the precipitate and the matrix
- A transformation tensor that transforms the precipitate phase lattice to the matrix phase lattice ($F_{B\to A}$)
- Candidate Burgers vectors (b) for the interface

The transformation tensor and Burgers vectors are calculated using:

- The lattice parameters (a, b, c, α, β, γ) of the precipitate and the matrix and
- An orientation relationship (OR).

Essentially there are two main clusters of information: those describing the two phases (elastic and lattice parameters) and those describing the bimaterial interface (OR, $F_{B\to A,}$ b), which are managed within two classes in the PIG_E code.

3. Code Overview

The PIG_E code has three main components that are contained in the individual classes shown in Figure 1: the **CrystalProperties()** class, the **CrystalInterface()** class, and the **NonSingularIsotropicBimaterialInterfaceEnergy**() (subsequently referred to as **NSIBIE()** for brevity**)** class. These three classes are material agnostic and require an associated *{Material}_Info.py* module to communicate necessary material-specific aspects of the system. Global functions utilized in PIG_E are described in Appendix A.

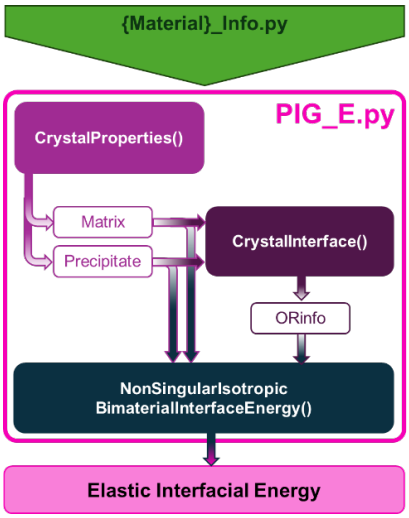

Figure 1. Schematic of PIG_E code for calculating elastic interfacial energy. Definition of classes are in solid boxes while instances of the class are in open boxes.

The **CrystalProperties()** class contains the functions needed to calculate the necessary crystallographic and elastic information about each phase. The functions are listed in Appendix B. There is an instance of the **CrystalProperties()** class for the matrix phase and one for the precipitate phase. These instances are passed to the **CrystalInterface()** class to generate information about the specific interface between the two phases.

The **CrystalInterface()** class generates the transformation matrix and the reference Burgers vectors from the crystallographic information of the **CrystalProperties()** instances for a given OR. Within the **CrystalInterface()** class are several representative ORs and associated functions to align the precipitate and matrix vectors to a common reference frame. Additional ORs can be defined within *{Material}_info.py* to describe the materials system of interest. The functions are listed in Appendix C.

The **NSIBIE()** class takes the instances of the **CrystalProperties()** and **CrystalInterface()** classes describing the matrix, precipitate, and OR to execute the elastic interfacial energy calculation. The results of the calculation are stored in the class, which also includes some functions for visualizing the interfacial energy. The functions are listed in Appendix D.

Functions for the example implementation of *{Material}_Info.py*, *Steel_Info.py* are in Appendix E.

The code is structured to facilitate high-throughput calculations using outputs from Thermo-Calc simulations but can also be run with manually input information about the phases. An example of manually inputting the necessary information is presented in a Jupyter notebook in Appendix F. A batch example Jupyter notebook is in Appendix G. The PIG_E code is in Appendix H and the *Steel_Info.py* module is in Appendix I.

4. Code Components

4.1 CrystalProperties() Class

The class contains the necessary crystallographic and elastic property information about the phase of interest for the elastic interfacial energy calculation. A separate instance of the class is created for each phase. The attributes are shown in Table 1. The first four are initialized with the class; if using the manual input option, only the CrystalStructure attribute is required to be populated.

When the **CrystalProperties()** class is initialized, there is an error check to ensure the initialized crystal structure is in the list of phases for which information has

been provided in *{Material}_Info.py*. There is also an error check to make sure the composition is fractional rather than percent.

Table 1. Attributes of the CrystalProperties() class.

Attribute	Meaning	Type	Units	Example
		initialized		
CrystalStructure	Name of the phase, in Thermo-Calc notation	string	…	'HCP_A3#2'
MolarVolume	Molar volume of the phase, not used for manual option	float	$\frac{\boldsymbol{m^3}}{\boldsymbol{mol}}$	5.95E-6
PhaseComposition	Composition of the phase in atomic fraction, not used for manual option	dict	atomic fraction	{'C': 0.33, 'Cr':0.67}
Temperature	Temperature at which the lattice and elastic information is calculated, not used for manual option	float	K	588
		calculated : self.LatticeParameters() or self.InputLattice()		
a b c	Lattice constants	float	Å	2.788 2.788 4.406
CellVolume	Optional, if using manual lattice input	float	Å^3	29.689
CBtoRB	Crystal basis to Reference basis matrix: used to orthogonalize crystal lattice	Array of float	Å	$\begin{bmatrix} 2.79 & -1.39 & 0 \\ 0 & 2.41 & 0 \\ 0 & 0 & 4.61 \end{bmatrix}$
g	Crystal metric tensor	Array of float	Å^2	$\begin{bmatrix} 7.77 & -3.89 & 0 \\ -43.89 & 7.77 & 0 \\ 0 & 0 & 19.4 \end{bmatrix}$
		calculated : self.ElasticConstants() or self.InputElastic()		
ShearModulus	Shear modulus of the phase	float	GPa	149.71
Poisson	Poisson's ratio of the phase	float	…	0.27

To standardize the crystallographic notation across crystal structures, which also helps to facilitate the OR calculations, we supply a conversion from the crystal reference frame to scaled fractional coordinates in a common orthonormal basis. This process is readily performed using the following transformation[6]:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} a & b\cos\gamma & c\cos\beta \\ 0 & b\cos\gamma & -c\sin\beta\cos\alpha^* \\ 0 & 0 & c\sin\beta\sin\alpha^* \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (1)$$

where a, b, c are the lattice parameters and α, β, γ are the lattice angles. The angle α^* is the angle in the reciprocal cell corresponding to α, determined by the cosine rule as

$$\cos\alpha^* = \frac{\cos\beta\cos\gamma - \cos\alpha}{\sin\beta\sin\gamma} \tag{2}$$

The transformation matrix CBtoRB (Crystal Basis to Reference Basis) is stored as a property in the instances of the **CrystalProperty()** class that represents both the matrix and precipitate phases.

The metric tensor, g, is another useful tool for performing calculations in arbitrary crystal reference frames. Mathematically, the metric tensor is a tool for measuring distances and angles in a Euclidean space. Here, it is used to calculate plane spacings and plane normal vectors. Using Einstein notation, the metric tensor is defined as

$$g_{ij} = e_i \cdot e_j = \begin{bmatrix} a^2 & ab\cos\gamma & ac\cos\beta \\ ab\cos\gamma & b^2 & bc\cos\alpha \\ ac\cos\beta & bc\cos\alpha & c^2 \end{bmatrix} \tag{3}$$

where e_i and e_j are the basis vectors. The metric tensor is stored as a property in the instances of the **CrystalProperty()** class that represent the matrix and precipitate phases. The transformation matrix, CBtoRB, and metric tensor, g, are employed as utilities in the ORs in **CrystalInterface()**. However, since both are properties of the crystal, they are defined and stored within **CrystalProperties()**.

4.1.1 Manual Input Option

Although the code is oriented towards high-throughput calculations with input from Thermo-Calc, the user can manually input both the lattice parameters and elastic constants. To manually input crystallographic information, the function is **CrystalProperties**.InputLattice(). To manually input elastic constants, the function is **CrystalProperties**.InputElastic(). The details of these functions are in Appendix B. An example implementation using manual input is included in Appendix F.

4.1.2 High-Throughput Option

Details about both the calculation of lattice and elastic parameters as a function of output data from Thermo-Calc can be found in ARL-TR-10028[5] but are summarized below. Input data is expected to be in a format containing a molar volume and a composition as a function of phase and crystal structure. This information is not required to be input from Thermo-Calc and could be generated

from other sources (e.g., PyCalphad). An example implementation using input from Thermo-Calc in a batch calculation is included in Appendix G.

4.1.2.1 Lattice Parameters

The function **CrystalProperties**.LatticeParameters() calculates the lattice spacing from molar volume (e.g., input from Thermo-Calc output) and assigns those lattice parameters to the instance of the class. To do so, it uses the LatticeParameterInfo() function from the *{Material}_Info.py* module, which returns the material-specific dictionaries for the number of atoms per unit cell (Unit), the average ratio of lattice parameters *b*:*a* and *c*:*a* (Ratio), and the angles of the unit cell (CellAngles). These dictionaries are all organized using the name of the phase as the primary keys.

The molar volume (*VM*) is converted to the volume of a unit cell (V_{cell}) using the equation

$$VM\ \left[\frac{m^3}{mol}\right]\left(\frac{mol}{N_{AV}}\right)\left(\frac{10^{-10}\text{Å}}{m}\right)^3\left(\frac{atoms}{unit\ cell}\right) \rightarrow V_{cell}\ [\text{Å}] \quad (4)$$

where the number of atoms per unit cell comes from the Unit dictionary above and N_{AV} is Avogadro's number. The volume of the cell is related to the lattice parameters (a, b, c) and cell angles (α, β, γ):

$$V_{cell} = a\ b\ c\ (1 - \cos^2\alpha - \cos^2\beta - \cos^2\gamma + 2\cos\alpha\cos\beta\cos\gamma) \quad (5)$$

where the lattice parameters b and c are replaced by the values from the Ratio dictionary to solve for a single lattice parameter a. The angle dependent term is calculated through the function **CrystalProperties**.Angles(), which takes the cell angles as input.

4.1.2.2 Elastic Parameters

The function **CrystalProperties**.ElasticConstants() assigns shear modulus and Poisson's ratio values to the instance of the class based on crystal structure, composition, and temperature. It is essentially a wrapper function calling the respective functions for the matrix and precipitate phases from *{Material}_Info.py*. These have been separated into ElasticConstantsMATRIX() and ElasticConstantsPRECIP() in *{Material}_Info.py* as it is generally assumed that the matrix phase of a given system is more likely to have a complex relationship. For example, in steels, the bulk and shear moduli have been experimentally measured and empirically quantified as a function of composition and time,[7] whereas the elastic constants of carbide phases are generally from DFT calculations.[5]

4.2 CrystalInterface Class

The **CrystalInterface()** class calculates the information necessary to describe the OR between the matrix and precipitate phases. The attributes of the class are shown in Table 2. The **CrystalInterface()** class takes as inputs the instances of the **CrystalProperties()** class for the matrix phase and precipitate phase, which includes the necessary information about lattice parameters and crystal structure to determine the possible ORs and calculate the transformation matrix.

Table 2. Attributes of the CrystalInterface() class

Attribute	Meaning	Type	Units	Example
	Initialized			
a_Matrix b_Matrix c_Matrix	Lattice parameters of the matrix phase	float	Å	2.87
Precipitate	Name of the phase (in Thermo-Calc notation)	str		'M7C3_D101'
a1 a2 a3	Lattice parameters of the precipitate phase	float	Å	4.5 6.9 11.9
CurrentORs	List of ORs that have existing functions in the class	list		['Bain', 'KurdjumovSachs', …, 'Bagaryatski', 'Isaichev']
bCount	Number of Burgers vectors in the interface plane for the OR	dict		{'Bain' :2, 'KurdjumovSachs' :3, …, 'Bagaryatski' :2, 'Isaichev' :2, }
InterfaceSubUnits*	A scaling factor for unit cells that contain more than one precipitate structural unit relative to a matrix structural unit on the interface plane	dict		'M7C3_D101' : {'Bagaryatski' : {'matrix' : np.array([[3.,1.,1]]).T, 'precip' : np.array([[2.,1.,1.]]).T, …}
	calculated from self.GetInterfaces()			
FBToA	Dictionary of results of transformation matrices in global reference frame	dict		{'Bagaryatski': array([[1.05, 0, 0], [0, 0.88, 0], [0, 0, 0.27]]), 'Isaichev': array([[1.05, 0, 0], [0, 0.93, 0.58], [0, 0, 0.37]])}
b	Dictionary of results of Burgers vectors	dict	m	{'Bagaryatski': array([[2.42e-10, 0, 0], [0, 3.96e-10, 0]]), 'Isaichev': array([[2.42e-10, 0, 0], [0, 3.96e-10, 0]])}

*Uses function GetSubUnits() from *{Material}_Info* module

An OR is defined as parallel vectors: $A_1||B_1$, $A_2||B_2$, and $A_3||B_3$, where **A** is the matrix reference frame and **B** is the precipitate reference frame. We define the $A_3||B_3$ as the interface plane normal; therefore $\{A_1,A_2\}\perp A_3$ and $\{B_1,B_2\}\perp B_3$. The CBtoRB attribute of the **CrystalProperty()** class transforms the vectors given in the OR to the desired scaled arrays of [A_1, A_2, A_3] called OR_alpha and [B_1, B_2, B_3] called OR_beta. These matrix and precipitate reference frames then need to be mapped to a common basis through a coordinate transformation. The matrices Q_α and Q_β:

$$Q_\alpha = \left[\frac{A_1}{\|A_1\|_2};\ \frac{A_3 \times A_1}{\|A_1\|_2\ \|A_3\|_2};\ \frac{A_3}{\|A_3\|_2}\right]^{-1} \tag{6a}$$

$$Q_\beta = \left[\frac{B_1}{\|B_1\|_2};\ \frac{B_3 \times B_1}{\|B_1\|_2\ \|B_3\|_2};\ \frac{B_3}{\|B_3\|_2}\right]^{-1} \tag{6b}$$

are used for this transformation, and $F_{B\rightarrow A}$ is the tensor that transforms the precipitate phase lattice to the matrix phase lattice:

$$[Q_\alpha A_1;\ Q_\alpha A_2;\ Q_\alpha A_3] = F_{B\rightarrow A}\left[Q_\beta B_1;\ Q_\beta B_2;\ Q_\beta B_3\right] \tag{7}$$

The Burgers vectors are also transformed to the common basis:

$$Q_\alpha b_\alpha = F_{B\rightarrow A} Q_\beta b_\beta \tag{8}$$

The **CrystalInterface()** class returns the $F_{B\rightarrow A}$ matrix and the Burgers vectors in the common basis. The nomenclature of Equations 6–8 is shown with the associated variables in the **CrystalInterface()** class in Table 3.

Table 3. Variables and associated symbols from Equations 6–8, which describe the transformation matrix process.

Variable	Symbol
OR_alpha	[A_1, A_2, A_3]
OR_beta	[B_1, B_2, B_3]
Qalpha	Q_α
Qbeta	Q_β
alpha	$[Q_\alpha A_1;\ Q_\alpha A_2;\ Q_\alpha A_3]$
beta	$\left[Q_\beta B_1;\ Q_\beta B_2;\ Q_\beta B_3\right]$
b0	b_α
b	$Q_\alpha b_\alpha$
FBToA	$F_{B\rightarrow A}$

The **CrystalInterface()** class has been pre-populated with some common ORs, which are delineated in Table 4; additional ORs can be added through the **NewOrientationRelationships()** class located in the *{Material}_Info.py* module. Details of the ORs can be found in the cited references and in Murdoch et al.[5]

Table 4. Orientation relationships in the current PIG_E code and Steel_Info material-specific module.

OR	Matrix / Precip.	Location	References
Bain	BCC / FCC	PIG_E	[8–10]
Kurdjumov - Sachs	BCC / FCC	PIG_E	[11, 12]
Nishiyama - Wasserman	BCC / FCC	PIG_E	[12]
Pitsch	BCC / FCC	PIG_E	[13]
Pitsch-Schrader	BCC / HCP	PIG_E	[9, 14]
MCη Pitsch-Schrader variant	BCC / HCP	Steel_Info	
Burgers	BCC / HCP	PIG_E	[12]
Bagaryatski [	BCC / Ortho	PIG_E	[12, 15, 16]
Isaichev	BCC / Ortho	PIG_E	[15]
Fe5C2	BCC / Mono	Steel_Info	[17]
KSI	BCC / Mono	Steel_Info	[5]

Each OR function is structured in the same way:

1) Parallel crystallographic directions and planes stored in arrays
2) Orthogonalization and scaling of the arrays
3) Determination of the burgers vectors

The OR function returns the orthogonalized and scaled arrays (OR_alpha, OR_beta) and the Burgers vectors (b0). A detailed example of the construction of an OR function is in the Manual input example Jupyter notebook in Appendix F.

The orthogonalization and scaling of the OR arrays are accomplished using the basis transformation matrix (CBtoRB) and the metric tensor (g) for the matrix or precipitate phase. For the OR directions, the basis transformation is accomplished by multiplying the crystallographic directions by CBtoRB. The result is a properly scaled vector in the common orthonormal basis.

Scaling the crystallographic planes requires the additional step of finding the normal direction of the interface plane. For cubic crystal systems, the normal direction is simply the Miller indices of the interface plane. However, for non-orthonormal unit cells (i.e., hexagonal, orthorhombic, or monoclinic) the normal vector does not necessarily match the Miller indices for the interface plane. Fortunately, the normal vector is readily determined using the metric tensor and properties of the reciprocal lattice. Regardless of crystal system, a reciprocal lattice vector, by definition, is perpendicular to the corresponding crystal plane. The reciprocal lattice vector is calculated by multiplying the interface plane Miller indices by the inverse of the metric tensor. Note that the inverse of the metric tensor is the reciprocal metric tensor. In PIG_E.py, the PlaneNormalVector() method within the **CrystalInterface()** class performs this calculation. Finally, the normal

vector is transformed and scaled into the common orthonormal basis as before by multiplying by CBtoRB.

There can be considerable mismatch between the sizes of the unit cells of the matrix and precipitate phases. In such cases, the interface is modeled as multiple units of the smaller phase rather than a 1:1 match. For example, this difference is readily illustrated for two different ORs for the cementite/ferrite interface, as depicted in Figure 2. For the Bagaryatski OR between cementite and the ferrite matrix, there is a 1:1 alignment of unit cells compared with a repetition of two matrix unit cells to one cementite unit cell in the Isaichev OR. The ratios of unit cells necessary to describe the alignment of ORs is contained in the InterfaceSubUnits attribute of the **CrystalInterface()** class. The information to populate the attribute is pulled from the *{Material}_Info.py*; a snippet of the dictionary corresponding to the example is shown here for reference, where the subunits illustrated in Figure 2 are in bold:

CEMENTITE_D011 : {Bagaryatski : {matrix : np.array([[1, **1**, 1]]).T,...

Isaichev : {matrix : np.array([[1, **2**, 1]]).T,...}

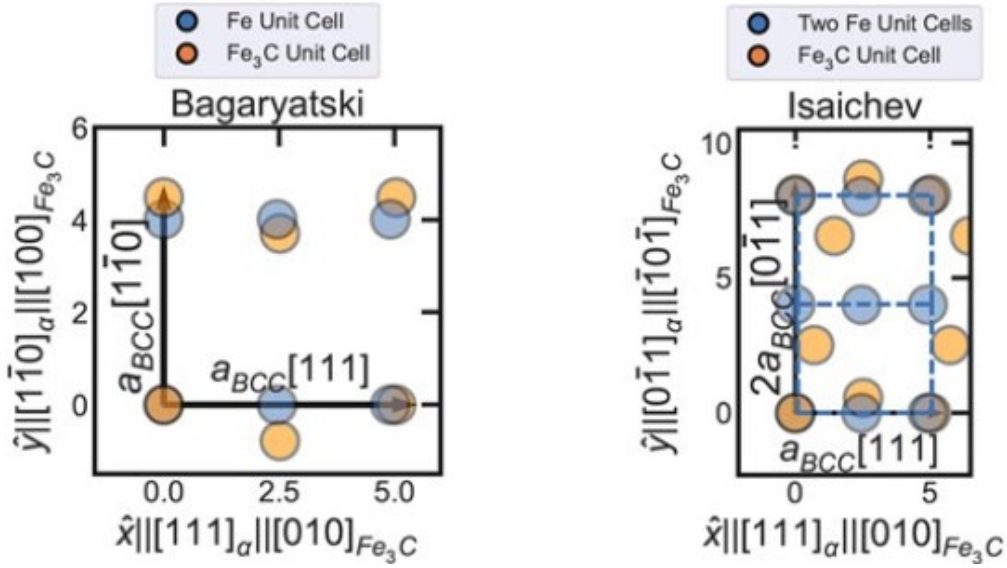

Figure 2. Interfaces between Fe_3C/cementite and ferrite matrix for the Bagaryatski and Isaichev ORs. The Bagaryatski OR has a 1:1 relationship compared with a 1:2 for Isaichev, where two repeating unit cells of the ferrite matrix match/align with one unit cell of the Fe_3C.

4.3 NonSingularIsotropicBimaterialInterfaceEnergy() Class

The **NSIBIE()** class calculates the elastic interfacial energy between matrix and precipitate assuming 1) isotropic elastic constants (shear modulus, Poisson's ratio) and 2) employing the non-singular dislocation formulation. Within this formulation the Burgers vector of each infinitesimal segment of dislocation is distributed throughout a finite volume surrounding the dislocation core. This contrasts with

discrete dislocations where Burgers vectors are represented by a Heaviside function. The conventional core cut-off length scale, r_0, is replaced by a characteristic distribution length scale, a. In this work, we employ the minimum Burgers vector length in the reference state as the core distribution length scale, i.e., aNS = Afactor*b, where Afactor is 1. An example of the effect of the characteristic length scale on the stress fields is shown in Figure 3 and compared with the discrete dislocation theory. The far-field isotropic bimaterial linear elastic stress calculations presented in Lubarda et al.[18,19] are incorporated into the non-singular theory of Cai et al.[20]

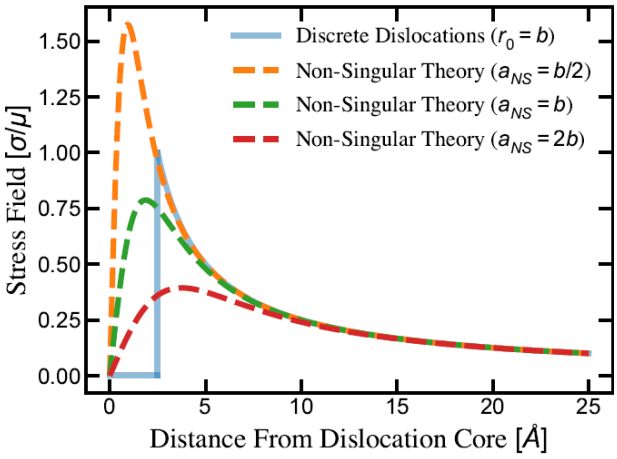

Figure 3. Example of the effect of the characteristic length scale aNS on the stress field, and the difference between non-singular theory and discrete dislocation theory (example Afactors of ½, 1, 2).

The **NSIBIE()** class takes as inputs the instances of the **CrystalProperties()** class that describe the matrix and precipitate phases, the name of the OR for which the interfacial energy will be calculated, and the instance of the **CrystalInterface()** class describing the ORs for the matrix/precipitate pairing. Table 5 describes the attributes of the class.

Table 5. Attributes of the NonSingularIsotropicBimaterialInterfaceEnergy() class.

Attribute	Meaning	Type	Units	Example
	initialized			
muMatrix	Shear modulus (μ) of the matrix.	float	GPa	80
nuMatrix	Poisson's ratio (ν) of the matrix.	float	…	0.3
muPrecipitate	Shear modulus (μ) of the precipitate.	float	GPa	182
nuPrecipitate	Poisson's ratio (ν) of the precipitate.	float	…	0.26
FBToA	Transformation matrix from the precipitate to the matrix.	rank-2 tensor	…	
b_1, b_2	Burgers vectors in natural matrix lattice.	vector	Å	2
Afactor	Prefactor for characteristic distribution length scale, a_NS = Afactor * b. Default is 1.	float	…	1.0
	calculated from self.__init__()			
delta	Partition factor between deformation of the matrix and deformation of the precipitate into the shared reference state.	float		$(0 \leq \delta \leq 1)$
xi1, xi2	Line directions for dislocation arrays 1 and 2.	vector	…	Vector with unit magnitude
b1, b2	Burgers vectors for dislocation arrays 1 and 2 in the reference state.	vector	Å	2.5
d1,d2	Dislocation spacing for dislocation arrays 1 and 2.	float	Å	100
Interface Energy	Computed interface energy.	float	J/m^2	0.27
Asymptotic Analytical Interface Energy	Approximate asymptotic analytical interface energy.	float	J/m^2	0.27

On initialization, the necessary attributes of the matrix and precipitate phases are assigned, and the interfacial energy is calculated. For a full description of the calculations, see Szajewski et al.[4] The full set of equations is in Appendix D and are briefly described below.

The Frank–Bilby equation[21,22] describes interface compatibility through the deformation of the matrix and precipitate natural states (F_{Matrix} and $F_{Precipitate}$) using an arbitrary vector in the interface plane, p:

$$\sum_{i=1}^{2}\left(\frac{n\times\xi_1}{d_i}\cdot p\right)b_i^{ref} = \left(F_{Precipitate}^{-1} - F_{Matrix}^{-1}\right)p \qquad (9)$$

where ξ_i and d_i are unique line directions and spacings for each of two sets of interface dislocations. The Burgers vector displacements of the dislocations are b^{ref} and n is the unit normal vector. This is represented schematically in Figure 4.

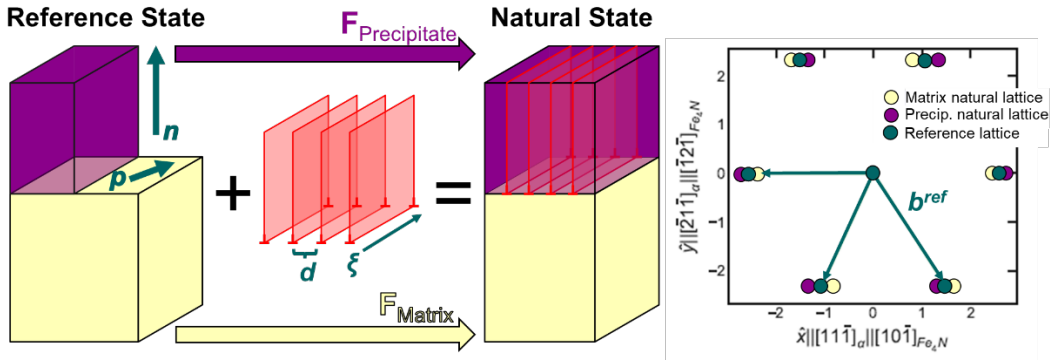

Figure 4. Schematic of matrix and precipitate phase relationships.

The deformation gradients, F, produce coherency strains and stresses in the bulk, σ_C. The dislocations produce a stress σ_D, which combined in the far field (∞) superpose:

$$\sigma_D^{\infty} + \sigma_C^{\infty} = 0 \tag{10}$$

Equations 9 and 10 are solved given the lattice and elastic constants for the phases and their OR. There may be multiple Burgers vector solutions depending on the OR so each one is tested in order to find the minimum interfacial energy:

$$\gamma = \frac{\|\xi_1 \times \xi_2\|_2}{2d_1 d_2} \int_{\partial A} \Delta u \cdot (\sigma_C + \sigma_D) \cdot dA \tag{11}$$

where Δu is the displacement of the precipitate phase relative to the matrix phase and the area, A, of the interface is defined by the parallelogram bounded by the two parallel dislocation arrays.

An analytical equation for the interfacial energy was also derived to complement the numerical solution, which is detailed in Appendix D. This equation makes several assumptions relative to Equation 11 but can provide alternative insights into the behavior of the interface energy with respect to materials properties.

For testing and debugging purposes, three functions are included to visualize the tensorial components of the mechanical fields along the interface. Specifically, these three functions illustrate the displacement vector field (*PlotDisplacement()*), the stress tensor field (*PlotStress()*), and the scalar energy density field (*PlotEnergyDensity()*). All three functions are callable directly from an instance of the **NSIBIE()** class and require no additional input arguments. Both *PlotStress()* and *PlotDisplacement()* contain several component fields, which may be toggled

between. Within *PlotDisplacement()* displacements u_x and u_y are on the slip plane and may be selected individually within the call to contourf. Similarly, within *PlotStress()* both σ_{yz} and σ_{xz} are shear components contained within the slip plane and may be selected individually within the call to contourf.

4.4 {Material}_Info.py

The PIG_E code utilizes a materials-specific companion module to describe the system; during the development of the code, the material system described is common carbides in martensitic steels,[5] with the module named **Steel_Info.py.** Additionally, since the current effort aimed to integrate with Thermo-Calc results for high-throughput calculations, the phase names are those used in the Thermo-Calc notation rather than common names (i.e., "CEMENTITE_D011" rather than more colloquial options such as cementite or Fe_3C)

Users must create their own materials-specific module in this framework to run the PIG_E code. The necessary components of the material-specific module will be described in the following section and are as follows:

- Summary of system: defining phases of interest, their crystal structure, and their ORs
- LatticeParameterInfo: dictionaries of crystallographic information about the phases necessary to convert molar volume to lattice parameters
- ElasticConstantsMATRIX/ ElasticConstantsPRECIP: dictionaries of elastic constants for phases of interest
- OR functions: any specific ORs that are not in the catalogue of common ORs already in the PIG_E code

The module is structured to facilitate high-throughput calculations of elastic interfacial energy from data about precipitate phases that have been output from thermodynamic software (e.g., Thermo-Calc). If the user is manually inputting the lattice parameters and elastic constants, only the "Summary of System" section and the relevant OR, if not already included in PIG_E, are necessary.

4.4.1 System Summary

There are three functions to globally describe the phases and elements of interest in the given system (Table 6). These quickly identify the labels of the phases to be used and their ORs. Error checks are built in during the subsequent calculations based on these definitions of the system, i.e., if a phase is not in the values of GetPrecipitateList() it will not run.

Table 6. Materials system summary functions.

Function	Returns	Values
GetMatrixPhase()	The name of the matrix phase	['BCC_A2']
GetPrecipitateList()	List of the names of the precipitate phases currently considered in the system; these are in Thermo-Calc nomenclature to facilitate high-throughput computations	['FCC_A1#2', …, 'M3C2_D510']
GetPrecipitateORs()	Dictionary of possible ORs for each named phase	{'FCC_A1#2': ['Bain', 'KurdjumovSachs', 'NishiyamaWasserman', 'Pitsch'], … 'M3C2_D510': ['Bagaryatski', 'Isaichev']}
GetPhaseElements()	List of elements considered in the current system	['C', 'Mn', 'Si', 'Cr', 'Ni', 'Mo', 'V', 'Co', 'Al', 'W', 'Cu', 'Ti', 'B', 'N', 'S', 'Fe']

4.4.2 Orientation Relationships

The dictionary in GetPrecipitateORs() is structured such that multiple possible ORs can be listed for a given phase; this will calculate the energy associated with each OR to compare the relative energies. Each OR has a function associated with it to calculate the transformation matrix. The name of the OR in the GetPrecipitateORs() list must be the same as the name of the function. Several OR functions for common ORs are contained in the **CrystalInterface()** class in the generalized PIG_E.py module (Table 4); however, if a matrix-precipitate interface follows an OR that does not have an existing function, it must be included in the material-specific module. For example, the Kurdjumov-Sachs OR is in PIG_E.py but the OR specific to the KSI carbide phase is not.

Additional ORs that are desired for the user-specific materials system are added through the **NewOrientationRelationships()** class in *{Materials}_Info*. The **CrystalInterface()** class in PIG_E inherits the **NewOrientationRelationships()** class, updating the necessary dictionaries and making the new OR functions accessible.

To demonstrate this, we consider a possible KSI OR developed by finding high symmetry directions within $M_{17}C_5$ that align with the most likely Burgers vector direction in the ferrite, the [111].[5] An OR for the monoclinic KSI carbide ($M_{17}C_5$) was not found in the literature. Therefore, an OR was proposed through comparison with similar crystal structures and analysis of relevant distances and angles between atoms. The KSI carbide has no apparent analog to other ORs, and so it was necessary to inspect atomic positions to identify potential O-lattice sites. By

examining bond angles, it was found that the c planes in the KSI carbide matched the typical BCC close packed slip system (i.e., slip in the [111] direction on the [011] plane).

Similarly, the Pitsch-Schrader OR needed to be adapted for the MC η phase. Specifically, the hexagonal lattice for V_6C_5 is rotated 30° relative to the typical hexagonal close packed (HCP) lattice. To create the modified Pitsch-Schrader OR, the HCP lattice directions, as defined in Pitsch-Schrader, are transformed by a 30° rotation around the c-axis. The subsequent scaling and basis change of the lattice vectors are unchanged.

4.4.3 Lattice Parameter

The general approach to calculate lattice parameters in a high-throughput method from thermodynamic software was described in Section 4.1.2.1, Lattice Parameters; full details are in Murdoch et al.[5] The required inputs are stored in LatticeParameterInfo() as dictionaries for the number of atoms in a unit cell, ratios of lattice parameters (i.e., b:a and c:a), and the unit cell angles (α, β, and γ). These dictionaries must use the phase name as the key value.

4.4.4 Elastic Properties

The elastic constants for the matrix and precipitate are determined using the ElasticConstantsMATRIX() and ElasticConstanctPRECIP() functions, respectively. Full details of how the elastic constants are generated in Steel_Info.py are included in Murdoch et al.[5] Briefly, the matrix, which is α-martensite, is described using functional relationships for the shear modulus and Poisson's ratio, which account for the composition of the matrix and the temperature.

The shear modulus as a function of temperature for α-martensite was derived in Ghosh and Olson[7]:

$$G_{\alpha'} = \left(8.068 + \sum x_j \left(\frac{d\mu}{dx_j}\right)_{Total}\right)\left[1 - 0.48797\left(\frac{T}{T_C}\right)^2 + 0.12651\left(\frac{T}{T_C}\right)^3\right] x10^{10}\ Pa\ for\ T < T_C \quad (12)$$

where T is the temperature and T_C is the Curie temperature. Alloying effects are incorporated through the $d\mu/dx$ term. The Poisson's ratio as a function of temperature and composition for the matrix phase is derived from the shear and bulk moduli, where the shear modulus is calculated from Equation 12, and the bulk modulus is[7]:

$$B_{\alpha-Fe} = 17.187\left[1 - 0.28029\left(\frac{T}{T_C}\right)^2 + 0.07221\left(\frac{T}{T_C}\right)^3\right] x10^{10}\ Pa\ for\ T < T_C \tag{13}$$

Poisson's ratio is then calculated as:

$$\nu = \frac{3B_{\alpha-Fe} - 2G_{\alpha'}}{2(3B_{\alpha-Fe} + G_{\alpha'})} \tag{14}$$

Elastic constants for the precipitate phases were tabulated as empirical values based on a survey of the literature. Depending on the phase of interest, composition, and availability of literature values, different rules were developed. For instance, the elastic constants in the M6C_E93 system were found to follow a linear dependence on the lattice parameter. For six phases, the elastic constants are determined from the primary alloying element (excluding carbon). In the remaining five phases, fixed values for the elastic constants are used.

5. Conclusion

The PIG_E code calculates the elastic strain energy contributions to interfacial energy between a precipitate and a matrix given some crystallographic and elastic constants using a dislocation-based approach. This approach also requires an interface OR for both the matrix and precipitate. PIG_E constructs a bimaterial interface, which is described by a transformation matrix and candidate Burgers vectors for the interface. During the calculation, the Burgers vectors set, which minimizes the dislocation density of the interface, is selected, and the corresponding elastic interfacial energy is determined using a non-singular approximation for the strain field at the dislocation core.

The PIG_E code contains generalized functions for executing elastic interfacial energy calculations. PIG_E is amenable to both manual input and high-throughput calculations. For manual operation, the user supplies phase-specific inputs for the matrix and precipitate. During high-throughput calculations, the phase-specific inputs are derived from CALPHAD simulations. The high-throughput option is structured specifically to couple to CALPHAD-based software, in this case Thermo-Calc. Both manual and high-throughput options require a supporting {Material}_Info.py file with materials-specific information necessary for the calculations. This structure was chosen to enable users to easily implement their material system of interest. {Material}_Info.py contains information on the relevant phases, including crystallographic information, functions for determining elastic constants, and the potential ORs. Here, an example Steel_Info.py file is described so that the user can generate their own supporting file for the system of interest.

6. References

1. Guziewski M, Coleman SP, Weinberger CR. Interface energetics and structure of the pearlitic microstructure in steels: an atomistic and continuum investigation. Acta Materialia. 2018;155:1–11.
2. Heo TW, Chen L-Q. Phase-field modeling of nucleation in solid-state phase transformations. JOM. 2014;66(8):1520–1528.
3. Soisson F, Martin G. Monte Carlo simulations of the decomposition of metastable solid solutions: transient and steady-state nucleation kinetics. Physical Review B. 2000;62(1):203.
4. Szajewski BA et al. Rapid estimation of the bimetallic interface energy for engineering steels; to be published, 2025.
5. Murdoch H, Szajewski B, Guziewski M, Hernandez-Rivera E, Field D, Magagnosc D. Assessment of materials data for high-throughput interfacial energy calculations for prevalent carbides. DEVCOM Army Research Laboratory (US); 2024. Report No.: ARL-TR-10028.
6. De Graef M, McHenry ME. Structure of materials: an introduction to crystallography, diffraction and symmetry. Cambridge University Press; 2012.
7. Ghosh G, Olson GB. The isotropic shear modulus of multicomponent Fe-base solid solutions. Acta Materialia. 2002;50(10):2655–2675.
8. Li X et al. A systematical evaluation of the crystallographic orientation relationship between MC precipitates and ferrite matrix in HSLA steels. Materials. 2022;15(11):3967.
9. Nagakura S, Oketani S. Structure of transition metal carbides. Transactions of the Iron and Steel Institute of Japan. 1968;8(5):265–294.
10. Yamasaki S. Modelling precipitation of carbides in martensitic steels. University of Cambridge; 2004.
11. Haghdadi N et al. Effect of ferrite-to-austenite phase transformation path on the interface crystallographic character distributions in a duplex stainless steel. Acta Materialia. 2018;145:196–209.
12. Bhadeshia HKDH. Worked examples in the geometry of crystals. The Institute of Materials; 2001.

13. Zhang W-Z et al. Unified rationalization of the Pitsch and T–H orientation relationships between Widmanstätten cementite and austenite. Acta Materialia. 2000;48(9):2209–2219.

14. Zahiri AH et al. The role of mechanical loading in bcc-hcp phase transition: tension-compression asymmetry and twin formation. Acta Materialia. 2022;241:118377.

15. Guziewski M. Multiscale study of the pearlitic microstructure in carbon steels: atomistic investigation and continuum modeling of iron and iron-carbide interfaces. Colorado State University; 2018.

16. Ohmori Y. χ-carbide formation and its transformation into cementite during the tempering of martensite. Transactions of the Japan Institute of Metals. 1972;13(2):119–127.

17. Yakel H. Crystal structures of stable and metastable iron-containing carbides. International Metals Reviews. 1985;30(1):17–44.

18. Lubarda V. Energy analysis of dislocation arrays near bimaterial interfaces. International Journal of Solids and Structures. 1997;34(9):1053–1073.

19. Lubarda V, Kouris D. Stress fields due to dislocation arrays at interfaces. Mechanics of Materials. 1996;23(3):191–203.

20. Cai W et al. A non-singular continuum theory of dislocations. Journal of the Mechanics and Physics of Solids. 2006;54(3):561–587.

21. Bilby BA, Bullough R, Smith E. Continuous distributions of dislocations: a new application of the methods of non-Riemannian geometry. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences. 1955;231(1185):263–273.

22. Frank FC. The resultant content of dislocations in an arbitrary intercrystalline boundary. Symposium on The Plastic Deformation of Crystalline Solids; 1950 May; Pittsburgh, PA. p. 2-2. NAVEXOS-P-834; vol. 150.

Appendix A. Global Functions in PIG_E

ConvertHCP4toHCP3(HCP)

Purpose	Changes from Miller-Bravais to Miller indices for HCP.
Required inputs	**HCP** : Miller-Bravais indices (4).
Optional inputs	None.
Returns	Miller indices(3).

TensorSort(M**)**

Purpose	Sorts a tensor according to the diagonal terms.
Required inputs	**M** : a symmetric tensor.
Optional inputs	None.
Returns	Sorted tensor.

Appendix B. Functions in CrystalProperties Class

Angles(*self*, alpha, beta, gamma):

Purpose	Function to calculate the angle dependent term of the cell volume. CellVolume = a * b * c (1-cos^2α-cos^2β-cos^2γ+2 cosα cosβ cosγ)
Required inputs	**alpha**, **beta**, **gamma**: cell angles in degrees.
Optional inputs	None.
Returns	Angle-dependent term of the cell volume.

InputLattice(*self*, a, b, c, alpha = 90, beta = 90, gamma = 90):

Purpose	Defining the lattice manually, rather than from Thermo-Calc.
Required inputs	**a**, **b**, **c** : lattice parameters in angstroms.
Optional inputs	**alpha**, **beta**, **gamma** : cell angles in degrees, default is 90°.
Returns	Assigns lattice parameters to class. Assigns cell volume to class in same dimension as lattice parameter input, e.g., angstroms^3. Generates matrix to orthogonalize crystal lattice to a reference lattice, CBtoRB.

LatticeParameters(*self*):

Purpose	Calculating lattice spacing from molar volume (e.g., input from ThermoCalc output).
Required inputs	None.
Optional inputs	None.
Returns	Assigns lattice parameters (a, b, c) to class. Generates matrix to orthogonalize crystal lattice to a reference lattice, CBtoRB.
Notes	Uses the following precompiled dictionaries from {Materials}_Info.py describing the unit cells of each phase: Unit: no. of atoms/unit cell Ratio: ratios of lattice parameters in the cell, b:a and c:a CellAngles: angles of the cell in degrees

Orthogonlize(*self*):

Purpose	Generates matrix to orthogonalize crystal lattice to a reference lattice, CBtoRB.
Required inputs	None.
Optional inputs	None.
Returns	Crystal Basis to Reference Basis matrix (CBtoRB).

MetricTensor(*self*):

Purpose	Generates the crystal metric tensor.
Required inputs	None.
Optional inputs	None.
Returns	Metric tensor (g).

InputElastic(*self*, Shear, Poisson):

Purpose	Manually input elastic constants rather than calculate from precompiled information.
Required inputs	**Shear**: Shear modulus of the phase in GPa. **Poisson**: Poisson's ratio of the phase.
Optional inputs	None.
Returns	Assigns ShearModulus and Poisson's attributes to class.

ElasticConstants(*self*, Martensite=True, Curie=False):

Purpose	Assigns shear modulus and Poisson values to class based on crystal structure, composition, and temperature.
Required inputs	None.
Optional inputs[a]	**Martensite**: Whether to use the equations for alpha'-martensite or alpha-ferrite. The default is True. **Curie**: Whether to use composition-dependent Curie temperature for matrix shear modulus equation. The default is False.
Returns	Assigns ShearModulus and Poisson's attributes to class.

[a] Due to the focus on steel while developing PIG_E.py, some parameters necessary for the calculation of the elastic constants in a ferrite/martensitic matrix are optional arguments in the function—namely, Booleans for whether the matrix will be considered martensitic and whether to consider the composition dependence of the Curie temperature.

Appendix C. Functions in CrystalInterface Class

GetInterfaces(*self*):

Purpose	Function to calculate the transformation matrix (FBToA) and Burgers vectors (b) for the interface.
Required inputs	None.
Optional inputs	None.
Returns	Populates FBToA and b dictionaries with results using the orientation relationship (OR) name as keys.

GetRotationMatrix(*self* , X, Z,M=np.eye(3)):

Purpose	Function to get rotation matrix from e to m sigma_m = Q * sigma_e *Q^T v_m = Q * v_e C' = R \cdot R \cdot C R^T \cdot R^T Equation 11a–11b in Appendix D.
Required inputs	**X** : Vector in the interface plane expressed in local crystallographic coordinates. **Z** : Vector normal to X (interface plane normal) expressed in local crystallographic coordinates.
Optional inputs	**M**: Basis transformation matrix.
Returns	Rotation matrix.

GetDeformationGradient(*self* , beta, alpha):

Purpose	Function to get deformation gradient. alpha = F{B→A}beta Equation 12 in Appendix D.
Required inputs	**beta** : List of three precipitate vectors in the common reference frame. **alpha** : List of three matrix vectors in the common reference frame.
Optional inputs	None.
Returns	F_{B->A} : Linear map, mapping the precipitate vectors to matrix vectors (all in common reference frame).

PlaneNormalVector(*self* , hkl, g):

Purpose	Function to calculate Miller indices of a crystal plane normal vector.
Required inputs	**hkl:** Miller indices of the crystal plane. **g:** Metric tensor for the crystal.
Optional inputs	None.
Returns	Normal vector.

PlaneSpacing(*self* , hkl, g):

Purpose	Function to calculate the d-spacing of a given lattice plane.
Required inputs	**hkl:** Miller indices of the crystal plane. **g:** Metric tensor for the crystal.
Optional inputs	None.
Returns	d-spacing.

{**OrientationRelationshipName**}(*self*):

Purpose	Return the components necessary to describe the specific orientation relationship.
Required inputs	None.
Optional inputs	None.
Returns	OR_alpha : scaled matrix vectors for the OR OR_beta : scaled precipitate vectors for the OR b0 : reference Burgers vectors

Appendix D. Functions and Equations in NonSingularIsotropicBimaterialInterfaceEnergy() Class

PlotDisplacement(*self,* Field='ux')

Purpose	Plot displacement field along interface.
Required inputs	None.
Optional inputs	Field : can be 'ux' or 'uy'. Default is 'ux'.
Returns	Returns a contour plot of the displacement field along the interface. The displacement field includes the superposition of the homogeneous linear displacment and the heaviside dislocation-induced displacement fields. The two relevant in-plane displacement components are Z[0,x,y] = u_{x} and Z[1,x,y] = u_{y}. An example call to plot is shown as CS = ax.contourf(X/Angs, Y/Angs, Z[1],levels = 55).

PlotStress(*self,* Field='Sxz')

Purpose	Plot stress field along the interface.
Required inputs	None.
Optional inputs	Field : can be 'Sxz' or 'Syz'. Default is 'Sxz'.
Returns	Returns a contour plot of the stress field along the interface. The stress field includes the superposition homogeneous coherency stress and the heterogeneous dislocation stress fields. The two relevant shear components are Z[3,x,y] = sigma_{yz} and Z[4,x,y]= sigma_{xz}. An example call to plot is shown as CS = ax.contourf(X/Angs, Y/Angs, Z[4],levels = 55).

PlotEnergyDensity(*self*)

Purpose	Plot of the energy density field along the interface.
Required inputs	None.
Optional inputs	None.
Returns	Returns a contour plot of the energy density field along the interface. The energy density field includes the superposition of the homogeneous elastic strain energy and the heterogenous dislocation-induced strain energy. An example call to plot is shown as CS = ax.contourf(X/Angs, Y/Angs, Z, levels = 55).

LubardaStressFields(*self*, a, x, y, D, bedge, bscrew)

Purpose	Calculate periodic array of dislocations.
Required inputs	a : dislocation core parameter x : x-dimension y : y-dimension D : dislocation spacing bedge : Burgers vector for edge dislocation bscrew : Burgers vector for screw dislocation
Optional inputs	None.
Returns	Periodic array of dislocations, Equations 15a–f below.

LubardaFarFieldStress(*self,* D, bedge, bscrew, Precipitate = True)

Purpose	
Required inputs	D : dislocation spacing bedge : Burgers vector for edge dislocation bscrew : Burgers vector for screw dislocation
Optional inputs	Precipitate : The default is True.
Returns	Equations 17a–f below.

The NonSingularIsotropicBimaterialInterfaceEnergy() class performs the dislocation-based calculations necessary to generate the interfacial energy. A brief overview of the equations is in this appendix with a full discussion available in a companion literature paper.[1] The equation numbers used in this appendix are congruent with those in the paper and those referenced in the Python code.

The Frank–Bilby equation describes the relationship between requisite misfit dislocation densities ($\sim 1/d_i$) and mismatched strain between two half-spaces representing a bimaterial system ($F^{-1}_{\text{Precipitate}} - F^{-1}_{\text{Matrix}}$). The relationship arises through kinematic compatibility, in that dislocations accommodate mismatched strains between the two bimaterial phases.

$$\sum_{i=1}^{2} \left(\frac{n \times \xi_i}{d_i} \cdot p \right) b_i^{ref} = \left(F_{Precipitate}^{-1} - F_{Matrix}^{-1} \right) p \tag{1}$$

Far away from the interface, both phases are in a stress-free equilibrated state. The stress in the bulk of either phase comprises far-field dislocation stresses and additional coherency stresses, the sum of which is zero.

$$\sigma_D^{\infty} + \sigma_C^{\infty} = 0 \tag{2}$$

Equations 1 and 2 together represent two coupled boundary value problems, one for each phase and coupled through Equation 1. The interface energy is excess energy required to form the interface relative to the bulk crystal. It is defined as the mechanical work required to displace the two crystals relative to each other in the presence of interface stress.

$$\gamma = \frac{\|\xi_1 \times \xi_2\|_2}{2 d_1 d_2} \int_{\partial A} \Delta u \cdot (\sigma_C + \sigma_D) \cdot dA \tag{3}$$

D-1. Solution Algorithm

We assume two continuous mappings between a shared reference lattice (deformed state) and two natural lattices representing bulk crystals. These maps are parameterized by δ and a deformation gradient ($F_{B \to A}$) mapping the precipitate phase to the matrix phase.

$$F_{Matrix}(\delta) = (1 - \delta) I + \delta F_{B \to A} \tag{4a}$$

$$F_{Precipitate}(\delta) = \delta I + (1 - \delta) F_{B \to A}^{-1} \tag{4b}$$

[1] Szajewski BA et al. Rapid estimation of the bimetallic interface energy for engineering steels; to be published, 2025.

Through rearranging Equation 1 and considering $p| \quad |\xi_1$or $p| \quad |\xi_2$, it may be shown that the two dislocation line directions are given by the following relations, which are consistent with Bollman's O-lattice theory:

$$\left(F^{-1}_{\mathrm{Pr}ecipitate}(\delta) - F^{-1}_{Matrix}(\delta)\right)^{-1} b_2^{ref}(\delta)| \quad |\xi_1(\delta) \tag{5a}$$

$$\left(F^{-1}_{\mathrm{Pr}ecipitate}(\delta) - F^{-1}_{Matrix}(\delta)\right)^{-1} b_1^{ref}(\delta)| \quad |\xi_2(\delta) \tag{5b}$$

In a small-strain, linear elastic framework, the coherency stress in the reference state as a function of the deformation gradient is:

$$\sigma_C = \frac{1}{2}\mathbb{C}(F^{-1} + F^{-T} - 2I) \tag{6}$$

Equation 2 may be parameterized by δ and is reduced to finding δ such that Equation 2 is satisfied:

$$\sigma_D^{\infty}\left(d_{1,2}(\delta), b_{1,2}^{ref}(\delta), \xi_{1,2}(\delta)\right) + \sigma_C^{\infty}\left(F(\delta)\right) = 0 \tag{7}$$

Equation 7 may be expressed as a minimization problem, which is readily amenable to numerical solution, i.e.:

$$\delta^* = \underset{0\le\delta\le1}{\operatorname{argmin}} \sum_{i,j=1}^{2} \sum_{\substack{k\in\{Matrix,\\ Precipitate\}}} \Big[e_i \otimes e_j : \mathbb{S}_k\big[\sigma_{D,k}^{\infty}\left(d_{1,2}(\delta), b_{1,2}^{ref}(\delta), \xi_{1,2}(\delta)\right) + \sigma_C^{\infty}\left(F(\delta)\right)\big]\Big]^2 \tag{8}$$

The displacement discontinuity shown in Equation 3 is the sum of a continuous linear displacement due to the mismatched coherency strains, a discontinuous step function due to dislocations, and an arbitrary constant:

$$\Delta u(x) = -\left(F^{-1}_{Precipitate}(\delta) - F^{-1}_{Matrix}(\delta)\right)x - u_D(x) + \frac{b_1^{ref}(\delta^*) + b_2^{ref}(\delta^*)}{2} \tag{9}$$

The discontinuous step function is represented by a staircase function acting at each dislocation core with the ceiling function denoted by $\lceil\cdot\rceil$:

$$u_D(x) = \sum_{i=1}^{2} \left\lceil \frac{n\times\xi_i}{d_i} \cdot x \right\rceil b_i^{ref} \tag{10}$$

D-2. Orientation Relationships and Transformation Matrix

We denote crystal reference frames with bases **A** and **B**. Both crystal reference frames must be mapped into a common basis. We denote the maps between the crystal reference frames and the common basis as:

$$Q_\alpha = \left[\frac{A_1}{\|A_1\|_2};\ \frac{A_3\times A_1}{\|A_1\|_2\ \|A_3\|_2};\ \frac{A_3}{\|A_3\|_2}\right]^{-1} \tag{11a}$$

$$Q_\beta = \left[\frac{B_1}{\|B_1\|_2};\ \frac{B_3\times B_1}{\|B_1\|_2\ \|B_3\|_2};\ \frac{B_3}{\|B_3\|_2}\right]^{-1} \tag{11b}$$

With this notation, the semicolon (;) separates columns and $\|.\|_2$ denotes the 2-norm. The linear map ($F_{B\to A}$) that transforms the precipitate phase to the matrix phase satisfies:

$$[Q_\alpha A_1;\ Q_\alpha A_2;\ Q_\alpha A_3] = F_{B\to A}\left[Q_\beta B_1;\ Q_\beta B_2;\ Q_\beta B_3\right] \tag{12}$$

D-3. Dislocation Stress Fields

We consider dislocations aligned along *z* with slip plane normal *x*. The stress fields for the non-singular dislocation framework are given by:

$$\hat{\sigma}_{xx}^{NS} \approx \left(\frac{\mu b_{edge}}{2\pi(1-\nu)}\right)\left(\frac{x}{x^2+y^2+a_{NS}^2}\right)\left(1-\frac{2(x^2+a_{NS}^2)}{x^2+y^2+a_{NS}^2}\right) \tag{13a}$$

$$\hat{\sigma}_{yy}^{NS} \approx -\left(\frac{\mu b_{edge}}{2\pi(1-\nu)}\right)\left(\frac{x}{x^2+y^2+a_{NS}^2}\right)\left(1+\frac{2(y^2+a_{NS}^2)}{x^2+y^2+a_{NS}^2}\right) \tag{13b}$$

$$\hat{\sigma}_{xy}^{NS} \approx \left(\frac{\mu b_{edge}}{2\pi(1-\nu)}\right)\left(\frac{y}{x^2+y^2+a_{NS}^2}\right)\left(1-\frac{2x^2}{x^2+y^2+a_{NS}^2}\right) \tag{13c}$$

$$\hat{\sigma}_{xz}^{NS} \approx -\left(\frac{\mu b_{screw}}{2\pi}\right)\left(\frac{y}{x^2+y^2+a_{NS}^2}\right)\left(1+\frac{a_{NS}^2}{x^2+y^2+a_{NS}^2}\right) \tag{13d}$$

$$\hat{\sigma}_{yz}^{NS} \approx \left(\frac{\mu b_{screw}}{2\pi}\right)\left(\frac{x}{x^2+y^2+a_{NS}^2}\right)\left(1+\frac{a_{NS}^2}{x^2+y^2+a_{NS}^2}\right) \tag{13e}$$

$$\hat{\sigma}_{zz}^{NS} \approx \nu\left(\hat{\sigma}_{yy}^{NS}+\hat{\sigma}_{xx}^{NS}\right) \tag{13f}$$

These expressions are readily extensible to the bimaterial system through the introduction of several parameters ($\mu\cdot, \omega, \nu\cdot, \beta, \epsilon$):

$$\hat{\sigma}_{xx}^{NS,L} \approx \left(\frac{\mu^* b_{edge}}{2\pi\left(1-\nu_{Precip}\right)}\right)\left(\frac{x}{x^2+y^2+a_{NS}^2}\right)\left(1-\frac{2\omega(x^2+a_{NS}^2)}{x^2+y^2+a_{NS}^2}\right) \tag{14a}$$

$$\hat{\sigma}_{yy}^{NS,L} \approx -\left(\frac{\mu^* b_{edge}}{2\pi\left(1-\nu_{Precip}\right)}\right)\left(\frac{x}{x^2+y^2+a_{NS}^2}\right)\left(1+\frac{2\omega(y^2+a_{NS}^2)}{x^2+y^2+a_{NS}^2}\right) \tag{14b}$$

$$\hat{\sigma}_{xy}^{NS,L} \approx \left(\frac{\mu^* b_{edge}}{2\pi\left(1-\nu_{Precip}\right)}\right)\left(\frac{y}{x^2+y^2+a_{NS}^2}\right)\left(1-\frac{2\omega x^2}{x^2+y^2+a_{NS}^2}\right) \tag{14c}$$

$$\hat{\sigma}_{xz}^{NS,L} \approx -\left(\frac{\epsilon b_{screw}}{2\pi}\right)\left(\frac{y}{x^2+y^2+a_{NS}^2}\right)\left(1+\frac{a_{NS}^2}{x^2+y^2+a_{NS}^2}\right) \tag{14d}$$

$$\hat{\sigma}_{yz}^{NS,L} \approx \left(\frac{\epsilon b_{screw}}{2\pi}\right)\left(\frac{x}{x^2+y^2+a_{NS}^2}\right)\left(1+\frac{a_{NS}^2}{x^2+y^2+a_{NS}^2}\right) \tag{14e}$$

$$\hat{\sigma}_{zz}^{NS,L} \approx \nu^*\left(\hat{\sigma}_{yy}^{NS,L}+\hat{\sigma}_{xx}^{NS,L}\right) \tag{14f}$$

Equations 14a–f may be extended to periodic arrays of dislocations. The fields are analytically soluble in the isotropic case:

$$\hat{\sigma}_{xx}^{NS,L} \approx \left(\frac{\mu^* b_{edge} X_i}{2\pi d_i(1-\nu_{Precip})}\right)[S_3(p_i,q_i)+\omega q_i S_6(p_i,q_i)] \tag{15a}$$

$$\hat{\sigma}_{yy}^{NS,L} \approx \left(\frac{\mu^* b_{edge} X_i}{2\pi d_i(1-\nu_{Precip})}\right)\left[S_3(p_i,q_i)-2\omega S_4(p_i,q_i)-\omega\frac{X_i^2+2A_i^2}{q_i}S_6(p_i,q_i)\right] \tag{15b}$$

$$\hat{\sigma}_{xy}^{NS,L} \approx \left(\frac{\mu^* b_{edge} X_i}{2\pi d_i(1-\nu_{Precip})}\right)[S_2(p_i,q_i)-2\omega X_i^2 S_5(p_i,q_i)] \tag{15c}$$

$$\hat{\sigma}_{xz}^{NS,L} \approx -\left(\frac{\epsilon b_{screw}}{2\pi d_i}\right)[S_2(p_i,q_i)+A_i^2 S_5(p_i,q_i)] \tag{15d}$$

$$\hat{\sigma}_{xz}^{NS,L} \approx \left(\frac{\epsilon b_{screw} X_i}{2\pi d_i}\right)\left[S_3(p_i,q_i)-\frac{A_i^2}{2q_i}S_6(p_i,q_i)\right] \tag{15e}$$

$$\hat{\sigma}_{zz}^{NS,L} \approx \nu^*\left(\hat{\sigma}_{yy}^{NS,L}+\hat{\sigma}_{xx}^{NS,L}\right) \tag{15f}$$

Expressions for $S_{1,\ldots,6}$ are given in Section D-5. Equations 15a–f may be expressed in tensorial form as:

$$\sigma_D'(d_i) = \begin{bmatrix} \sigma_{xx}^{NS,L} & \sigma_{xy}^{NS,L} & \sigma_{xz}^{NS,L} \\ \sigma_{yx}^{NS,L} & \sigma_{yy}^{NS,L} & \sigma_{yz}^{NS,L} \\ \sigma_{zx}^{NS,L} & \sigma_{zy}^{NS,L} & \sigma_{zz}^{NS,L} \end{bmatrix} \tag{16}$$

In the limit that $x \pm \infty$ Equations 15a–f simplify to asymptotic fields far away from the dislocation cores. These fields are given by:

$$\sigma_{xx}^{\infty} = \frac{\beta b_{edge}\mu^*}{2d_i(1-\nu_{Precip})} \tag{17a}$$

$$\sigma_{yy}^{\infty} = \frac{(\beta^2-2(1-\omega))b_{edge}\mu^*}{2\beta d_i(1-\nu_{Precip})} \tag{17b}$$

$$\sigma_{xy}^{\infty} = 0 \tag{17c}$$

$$\sigma_{xz}^{\infty} = 0 \tag{17d}$$

$$\sigma_{yz}^{\infty} = \frac{\epsilon(1-\omega)b_{screw}}{2\beta d_i} \tag{17e}$$

$$\sigma_{zz}^{\infty} = \nu^{*}\left(\sigma_{xx}'^{\infty} + \sigma_{yy}'^{\infty}\right) \tag{17f}$$

D-4. Materials Data

The lattice constants and angles defining the lattice $(a_1, a_2, a_3, \alpha, \beta, \gamma)$ are related through the molar volume:

$$V_m = \frac{N_A}{Z} a_1 a_2 a_3 \sqrt{1 + 2\cos\alpha\cos\beta\cos\gamma - \cos^2\alpha - \cos^2\beta - \cos^2\gamma} \tag{18}$$

D-5. Asymptotic Analytical Model

As an alternative to Equation 3, we have derived an approximate analytical model. This model does not use the non-singular dislocation theory and relies on a conventional core cutoff parameter. The model may be succinctly expressed as:

$$\gamma_{Analytic} \approx \frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{2}\left(\frac{n\times\xi_i}{2\pi d_i} \otimes b_i^{ref}\right) : \left(R(\xi_j)\sigma_j^0 R^T(\xi_j)\right) \ln\left(\frac{d_j}{2\pi r_0 \|\xi_1\times\xi_2\|_2}\right) \tag{19}$$

where

$$\sigma_j^0 = \begin{bmatrix} \frac{\xi_1\cdot\xi_2}{\|\xi_1\times\xi_2\|_2} \\ 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} -\epsilon b_{screw,j} \\ \frac{\mu^* b_{edge,j}}{1-\nu_{Precip}} \\ 0 \end{bmatrix} \tag{20a}$$

$$R(\xi_j) = \begin{bmatrix} \xi_j \\ n\times\xi_j \\ n \end{bmatrix}^T \tag{20b}$$

The constants in the bimaterial stress fields are given by:

$$\omega = \begin{cases} 1-\beta & if\ Precip \\ 1+\beta & if\ Matrix \end{cases} \tag{A.1a}$$

$$\epsilon = \begin{cases} \mu_{Precip}(1-k_4) & if\ Precip \\ \mu_{Matrix}(1+k_4) & if\ Matrix \end{cases} \tag{A.1b}$$

$$\nu^* = \begin{cases} \nu_{Precip} & if\ Precip \\ \nu_{Matrix} & if\ Matrix \end{cases} \tag{A.1c}$$

and

$$\mu^* = \frac{1+\alpha}{1-\beta^2}\mu_{Precip} \tag{A.2a}$$

$$\alpha = \frac{\Gamma(k_1+1)-k_2-1}{\Gamma(k_1+1)+k_2+1} \tag{A.2b}$$

$$\beta = \frac{\Gamma(k_1-1)-k_2+1}{\Gamma(k_1+1)+k_2+1} \quad \text{(A.2c)}$$

Additionally, the infinite series listed when summing over arrays of dislocations are given by:

$$S_2(p,q) = \frac{\pi \sin(2\pi p)}{\cosh(2\pi q)-\cos(2\pi p)} \quad \text{(B.2a)}$$

$$S_3(p,q) = \frac{(\frac{\pi}{q})\sinh(2\pi p)}{\cosh(2\pi q)-\cos(2\pi p)} \quad \text{(B.2a)}$$

$$S_4(p,q) = \frac{(2\pi^2)(\cosh(2\pi q)\cos(2\pi p)-1)}{(\cosh(2\pi q)-\cos(2\pi p))^2} \quad \text{(B.2c)}$$

$$S_5(p,q) = \frac{(\pi^2/q)(\sinh(2\pi q)\sin(2\pi p))}{(\cosh(2\pi q)-\cos(2\pi p))^2} \quad \text{(B.2d)}$$

$$S_6(p,q) = \frac{\left(-\frac{\pi}{q^2}\right)[(\cosh(2\pi q)-\cos(2\pi p))\sinh(2\pi q)+2\pi q(\cos(2\pi p)\cosh(2\pi q)-1)]}{(\cosh(2\pi q)-\cos(2\pi p))^2} \quad \text{(B.2e)}$$

The term $S_2(p,q)$ is particularly relevant in computing the interface energy (Equation 3). Two useful integrals are the following:

$$\int_{r_0}^{1-r_0} S_2(p,0)dp = 0 \quad \text{(B.3a)}$$

$$\int_{r_0}^{1-r_0} pS_2(p,0)dp = \ln(2\pi r_0) \quad \text{(B.3b)}$$

Appendix E. Functions in Steel_Info.py

GetMatrixPhase()

Purpose	The name of the matrix phase.
Required inputs	None.
Optional inputs	None.
Returns	The name of the matrix phase.

GetPrecipitateList():

Purpose	List of the names of the precipitate phases considered in the system.
Required inputs	None.
Optional inputs	None.
Returns	List of precipitate phases currently considered in the system.

GetPrecipitateORs():

Purpose	Dictionary of possible orientation relationships (ORs) for each named carbide phase.
Required inputs	None.
Optional inputs	None.
Returns	Dictionary of possible ORs for each named carbide phase.

GetPhaseElements():

Purpose	List of elements considered in the current system.
Required inputs	None.
Optional inputs	None.
Returns	List of elements considered in the current system.

LatticeParameterInfo(Phase):

Purpose	Dictionaries describing the crystallography of the phase. Phase names are the dictionary keys.
Required inputs	Phase: Name of the phase (in Thermo-Calc notation).
Optional inputs	None.
Returns	Unit: Number of atoms in the unit cell for the phase. Ratio: Ratio of lattice parameter b to a and c to a for the phase. Cell Angles: Angles alpha/beta/gamma of the phase lattice

ElasticConstantsMATRIX(CrystalCLASS, Martensite=True, Curie=False):

Purpose	Calculates elastic constants of ferrite or martensite matrix as a function of composition and temperature. Equations from Ghosh & Olson.[1]
Required inputs	**CrystalCLASS**: The instance of the class to which the properties will be added.
Optional inputs	**Martensite**: Whether to use the equation for martensite. The default is True. If False, ferrite equation is used. **Curie**: Whether to use a compositionally dependent Curie temperature or that of pure iron (Fe). The default is False. If using a Curie temperature, the units are in kelvin (K).
Returns	ShearModulus: Shear modulus of the matrix in GPa. Poisson: Poisson's ratio of the matrix.

[1] Ghosh G, Olson GB. The isotropic shear modulus of multicomponent Fe-base solid solutions. Acta Materialia. 2002;50(10):2655–2675.

ElasticConstantsPRECIP(CrystalCLASS):

Purpose	Returns the elastic constant of a carbide phase (as a function of composition). Contains dictionaries by phase. Sub-dictionary keys include specific elements if constants vary by composition, and "Alloy" for an average/ not varied by composition. For details see ARL-TR-10028.[2]
Required inputs	**CrystalCLASS**: The instance of the class to which the properties will be added.
Optional inputs	None.
Returns	ShearModulus: Shear modulus of the precipitate in GPa. Poisson: Poisson's ratio of the precipitate.

ConvertHCP4toHCP3(HCP)

Purpose	Changes from Miller-Bravais to Miller indices for HCP.
Required inputs	**HCP** : Miller-Bravais indices (4).
Optional inputs	None.
Returns	Miller indices (3).

GetSubUnits():

Purpose	Dictionary of phases for unit cells that contain more than one precipitate structural unit relative to a matrix structural unit on the interface plane.
Required inputs	None.
Optional inputs	None.
Returns	Dictionary of subunits by precipitate and OR Keys are phase name Subdictionary keys are: OR name, then: matrix : subunits of matrix unit cell precip : subunits of precipitate unit cell

[2] Murdoch H, Szajewski B, Guziewski M, Hernandez-Rivera E, Field D, Magagnosc D. Assessment of materials data for high-throughput interfacial energy calculations for prevalent carbides. DEVCOM Army Research Laboratory (US); 2024. Report No.: ARL-TR-10028.

Appendix F. Manual Example Jupyter Notebook

This appendix appears as a PDF attached to the report.

Appendix G. Batch Example Jupyter Notebook

This appendix appears as a PDF attached to the report.

Appendix H. PIG_E Code

This appendix appears as a PDF attached to the report.

Appendix I. Steel_Info.py

This appendix appears as a PDF attached to the report.

List of Symbols, Abbreviations, and Acronyms

DFT	density functional theory
HCP	hexagonal close packed
NSIBIE	NonSingularIsotropicBimaterialInterfaceEnergy
OR	orientation relationship
PIG-E	Predicting Interfacial Geometric Energy

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CI TECH LIB