SOFTWARE METAPAPER

pydiffusion: A Python Library for Diffusion Simulation and Data Analysis

Zhangqi Chen, Qiaofu Zhang and Ji-Cheng Zhao
Department of Materials Science and Engineering, The Ohio State University, College Road, Columbus, OH, US
Corresponding author: Ji-Cheng Zhao (zhao.199@osu.edu)

pydiffusion is a free and open-source Python library designed to solve diffusion problems for both single-phase and multi-phase binary systems. The key features of pydiffusion include fast simulation of multi-phase diffusion and extraction of diffusion coefficients from experimental concentration profiles using forward simulation analysis. pydiffusion also provides various mathematical models for diffusion profile smoothing, diffusion coefficient evaluation, and data optimization. In pydiffusion, diffusion profiles and various phases are easy to define or read from the experimental datasets. Visualization tools based on Matplotlib are also provided to help users present or refine their simulations and analysis.

(1) Overview

Introduction

Diffusion is the transport of matter from one point to another by thermal motion of atoms or molecules [1]. It is one of the fundamental processes in nature. Diffusion in solids was firstly discovered by Roberts Austen in the late 19th century. Since then, various theories and models have been established to describe solid-state diffusion. Tremendous amount of diffusion data has been collected to better understand and predict diffusion associated phenomena such as precipitation, homogenization, solidification and creep deformation. In the past decades, many computational tools [2, 3] are powerful enough to handle complex simulation problems. A fast and stable simulation tool can help researchers understand and optimize diffusion processes in materials.

As one of the fundamental properties in materials, diffusion coefficients are essential inputs for the establishment of mobility database and Integrated Computational Materials Engineering (ICME) [4] based material design and process optimization. Many high-throughput approaches [5] have been developed to establish various materials databases and accelerate materials design. Forward simulation analysis (FSA) is a powerful approach to extract diffusion coefficient data from diffusion couple experiments [6, 7]. It has been applied to various diffusion systems and has demonstrated its robustness over the last several years [8–13].

The pydiffusion software package is an open-source Python library designed to simulate diffusion and analyse diffusion data using various mathematical and simulation models. Compared to commercialized simulation software like DICTRA [2] and PanDiffusion [3], pydiffusion focuses on simulation with diffusion coefficients data instead of mobility databases. The key feature of pydiffusion includes fast simulation of multi-phase systems and extraction of diffusion coefficients from experimental profiles. pydiffusion also provides easy constructors for diffusion systems and profile objects, which makes it easy to learn and perform diffusion simulations and data analysis quickly.

Implementation and architecture

The general architecture of pydiffusion is depicted in **Figure 1**. Diffusion profile and diffusion coefficients are the two basic objects in diffusion studies. They are represented as `DiffProfile` and `DiffSystem` object respectively in pydiffusion. The process from `DiffSystem` to `DiffProfile` is a diffusion simulation, the reversed process is diffusion coefficient ($\bar{D}$) extraction. Users can read/save `DiffProfile` and `DiffSystem` from/to csv files or plot them directly.

DiffProfile

The `DiffProfile` object is the fundamental representation of diffusion concentration profile data in pydiffusion. It includes the information of 1-dimensional (1D) grids with distance and composition data. For multi-phase diffusion profile, the positions of phase boundaries/interfaces are also included.
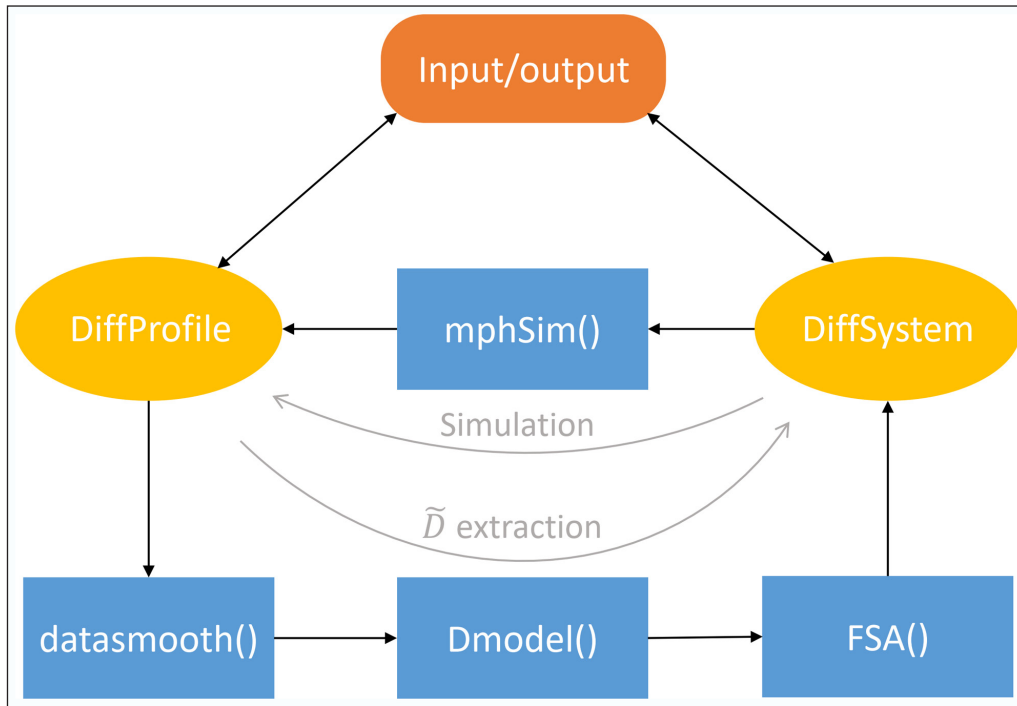
Figure 1: General architecture of the pydiffusion package.

There are many types of profile data in diffusion analysis. For experimental raw data, `DiffProfile` can be constructed by providing composition and distance data. For the initial profile data used for a simulation, `step()` method is usually used to construct a step profile based on pre-constructed grids. 1D grids can be constructed using the `mesh()` function, which provides nonlinear/linear meshing of 1-dimensional grids. For convenience, pydiffusion also provides `save_csv()` and `read_csv()` methods to save/read `DiffProfile` to/from csv format files.

`DiffSystem`
The `DiffSystem` is a representation of diffusion coefficients data of a binary system. For each phase within the binary diffusion system, the `DiffSystem` includes its solubility range and a function representing its compositional dependence of interdiffusion coefficients, where spline functions are usually used. The `DiffSystem` can be constructed with a given spline function for each phase. User can also construct the `DiffSystem` with solubilities and diffusion coefficient data.

Like `DiffProfile`, `save_csv()` and `read_csv()` can save/read `DiffSystem` as well. Users can choose to either save `DiffProfile` and its corresponding `DiffSystem` together within one csv file or save them separately.

`mphSim()`
`mphSim()` is the core diffusion simulation method within pydiffusion. Its functionality is to use given diffusion coefficients to simulate diffusion process for a certain amount of time. The keyword arguments of `mphSim()` are

```
def mphSim(profile, diffsys, time, liquid=0,
output=True, name='')
```

`profile` is the initial `DiffProfile` before simulation, `diffsys` is the `DiffSystem` during simulation, `time` is the diffusion time in seconds and `liquid` gives the position of liquid phase in the current geometry. By default, `liquid=0` means there is no liquid phase during simulation. `liquid=1/-1` is corresponding to liquid phase attached to left/right of the current geometry. The `mphSim()` method returns the simulation result as a `DiffProfile` object.

Within a single phase, diffusion can be simulated using the finite difference method based on Fick's two diffusion laws:

$$J = -D\frac{dC}{dx}, \quad \frac{dC}{dt} = -\frac{dJ}{dx} \qquad \text{Eq. 1}$$

For a binary diffusion system with multiple phases, phase boundary/interface is controlled by moving boundary condition:

$$v_{\alpha\beta} = \frac{dx_{If}}{dt} = \frac{J_\alpha - J_\beta}{C_\alpha - C_\beta} \qquad \text{Eq. 2}$$

In `mphSim()`, phase boundaries/interfaces are independent from simulation grids, i.e. not attached to any grids, the geometry nearby a phase interface is illustrated in **Figure 2**. The sharp interface model is used so a phase interface is a point with no width. There are two composition values C_α and C_β assigned to each interface, which corresponds to the solubility limits of two adjacent phases α and β. The phase interface can move across any simulation grids following Eq. 2. Once a simulation grid is passed by an interface, it will be assigned to the adjacent phase.
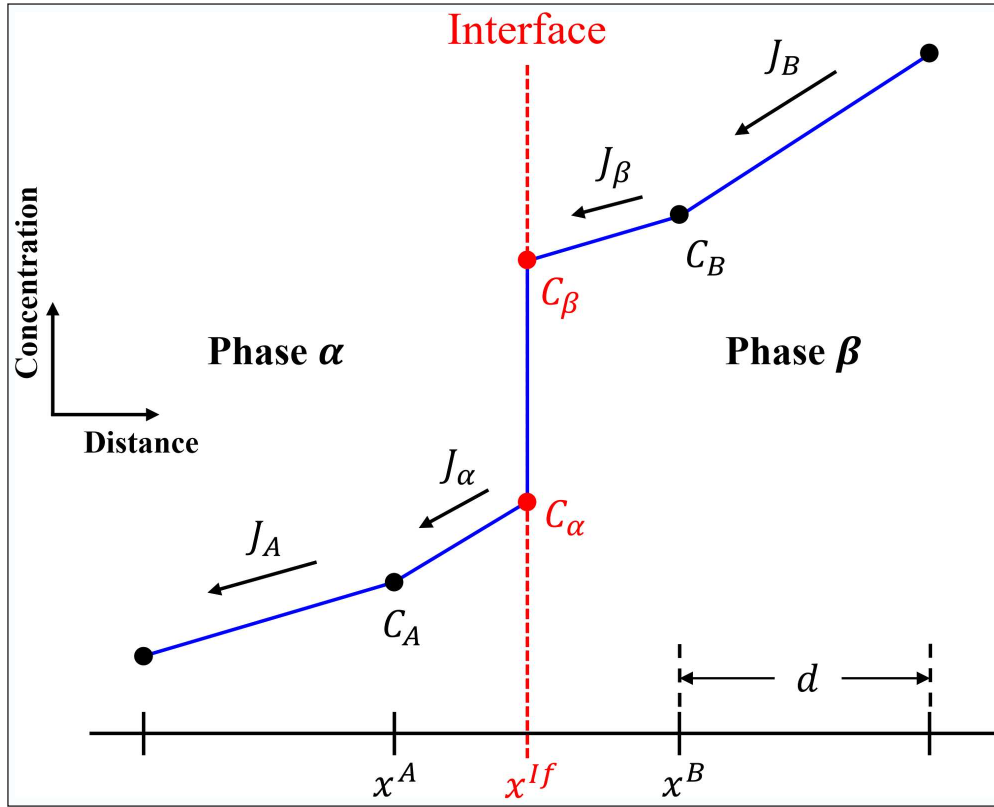
Figure 2: Configuration nearby an interface on simulation grids. The dash line denotes the interface at location x^{If}. The height of each dot represents the concentration value (C) at each grid and the interface. Connecting those dots, yielding the solid curve, is the concentration profile during simulation. The arrows represent the mass flux J between adjacent grids.

To ensure stability, the simulation time step Δt must be strictly under control. In every simulation loop within `mphSim()`, time step Δt is limited by the following three rules:

1. Within every phase, von Neumann analysis [14] gives the upper limit of simulation time step to prevent simulation instability.

$$\Delta t < \frac{d^2}{2\tilde{D}} \qquad \text{Eq. 3}$$

   In which d is the grid size, $\tilde{D}$ is the interdiffusion coefficient.
2. Each phase interface can pass by one grid at most after each loop. The purpose of this rule is to help arrange grids easier by the end of each simulation loop.
3. For the grids nearby phase boundaries, the two rules above cannot keep their composition values within their solubility limits. An additional rule must be assigned. In **Figure 2**, $C_A < C_\alpha$ and $C_B < C_\beta$ must be ensured after applying Fick's 2nd law, which is:

$$C_A + \Delta C_A = C_A - \frac{J_\alpha - J_A}{d} \cdot \Delta t < C_\alpha$$

$$C_B + \Delta C_B = C_B - \frac{J_B - J_\beta}{d} \cdot \Delta t > C_\beta \qquad \text{Eq. 4}$$

So Δt must satisfy:

$$\Delta t < \frac{(C_\alpha - C_A)d}{J_A - J_\alpha} \quad \text{only if } J_A > J_\alpha$$

$$\Delta t < \frac{(C_B - C_\beta)d}{J_B - J_\beta} \quad \text{only if } J_B > J_\beta \qquad \text{Eq. 5}$$

In diffusion with a liquid phase attached, diffusion coefficients in liquid phase ($\sim 5 \times 10^{-9} \text{m}^2/\text{s}$) is extremely high in comparison with a solid phase. Therefore, liquid can only be attached to the end of the geometry in `mphSim()`, which is controlled by the `liquid` parameter. During simulation, the liquid phase only provides a diffusion flux at the phase boundary/interface.

For diffusion simulations with thin films, users can setup the initial profile with a very thin phase on the left/right. `mphSim()` will delete such phase once it is consumed during simulation.

`mphSim()` can handle most of diffusion simulation for complex systems with multiple phases. For easy simulation of single-phase systems, users can also use `sphSim()` method, in which moving boundary condition is not considered.

`datasmooth()`

The `datasmooth()` function is responsible for smoothing experimental diffusion profile data. Better quality of diffusion coefficients can be extracted using

Sauer-Freise equation [15] afterwards. `datasmooth()` uses moving average method to smooth diffusion profiles. For multi-phase systems, data within different phases are smoothed separately. Then sharp phase interfaces will be created between two adjacent phases to obtain more accurate solubility information of the multiple phases. This step is very important especially for diffusion systems in which diffusion coefficients change dramatically among the various phases [8, 9, 16]. `datasmooth()` function returns smoothed data as a `DiffProfile` object with much more grids compared with original one, which is done by interpolation after data smoothing.

`Dmodel()`

The `Dmodel()` function is used to create a `DiffSystem` object based on smoothed profile by the `datasmooth()` function. The output of `DiffSystem` will be used as the initial diffusion coefficients of upcoming FSA. `Dmodel()` will ask users to model each phase consequently. There are two interpolation methods to choose to represent the composition dependence of diffusion coefficients: `splrep()` and `UnivariateSpline()`, both of which belongs to the `scipy.interpolate` module. `UnivariateSpline()` is useful for smoothed data, while `splrep()` is suitable for more scattering data.

`FSA()`

`FSA()` represents the forward simulation analysis, which can extract interdiffusion coefficients from raw diffusion profile data [6, 7]. Its keyword arguments are

```
def FSA(profile_exp, profile_sm, diffsys, time,
Xlim=[], n=[400, 500], w=None, name='')
```

in which, `profile_exp` is the `DiffProfile` contains raw experimental data, `profile_sm` is the output of `datasmooth()` on `profile_exp`, `diffsys` is a `DiffSystem` object represents initial guess of diffusion coefficients, which is usually the output of `Dmodel()`, and `time` is diffusion time in seconds. A step profile is used as initial profile for every simulation, `Xlim` is used as the composition values of the two ends if provided.

In FSA, `diffsys` is used to simulate a diffusion process through `mphSim()` method and compared with `profile_exp` repeatedly. After each simulation, `diffsys` is modified using `Dadjust()` method. The difference between simulation and `profile_exp` will be calculated through comparing their concentration values at all grids. Once the difference is small enough, the simulation will stop. The final `diffsys` after several modifications will be output as FSA result.

Because `mphSim()` will be applied several times within `FSA()`, an accurate and efficient simulation is required for the `FSA()` method. Before the first simulation, grids will be arranged efficiently to accelerate all simulation runs. Based on Eq. 3, lower $\tilde{D}$ requires smaller grid size while higher $\tilde{D}$ affords larger grid size, so that higher time step Δt can be assigned to each simulation loop. Therefore, the simulation grid size d follows $d \propto D^\alpha$ based on `profile_sm`, in which α is a constant with a default value of 0.3 which has been tested well in various systems. This meshing process can be done by `automesh()`

method. In FSA's keyword arguments, `n` is an integer list with length 2 which indicates the minimum and maximum number of grids produced by `automesh()`. More grids will help perform simulation with more details but at the expense of lower speed.

Parameter `w` is a list whose length equals to the number of phases. It provides the weights used to calculate differences between simulation and experimental profile `profile_exp`. By default, all phases have the same weight. This parameter is useful when diffusion data of multiple phases have different accuracy.

`FSA()` uses `Dadjust()` function to adjust `diffsys` after each simulation run. Users may select from Phase Mode and Point Mode to adjust diffusion coefficients during FSA. Phase Mode only increases or decreases all diffusion coefficients $\tilde{D}$ together within one phase, and the shape of $\tilde{D}$ doesn't change after adjustment. The Point Mode adjusts selected points across the solubility range individually; In other words, both the magnitude and shape of $\tilde{D}$ curve changes. Compared to the Point Mode, the Phase Mode has less degree of freedom but higher stability.

Quality control

The core simulation mechanism in pydiffusion was developed in 2013 by Zhang and Zhao [6]. Diffusion simulation is hard to perform especially for those systems with multiple phases or highly composition dependent diffusion coefficients, such as Ti-(Mo,Nb,Ta), Cu-Zn and Mg-Al. Compared with the original Matlab implementation, pydiffusion has much higher stability and efficiency on simulating those binary systems. Usually, `mphSim()` function can complete a diffusion simulation within 30 seconds using default settings. For systems with highly composition dependent diffusion coefficients, the pydiffusion simulation speed is hundreds of times faster than the original Matlab code. In addition, the diffusion profiles simulated using pydiffusion also have much greater details, especially for thin-layer with extremely slow diffusion. This is due to the advanced meshing approach implemented in the pydiffusion package.

Sauer-Freise equation [15] is one of the approaches to extract interdiffusion coefficients from diffusion profiles. It is also a good validation for simulation reliability. **Figure 3** shows this validation using diffusion coefficient data of the Ni-Mo system at 1100°C [6]. Solid line is the diffusion coefficients (`DiffSystem`) used in the simulation, while crosses represent Sauer-Freise calculation results based on simulated profile. We can observe that the original diffusion coefficients and Sauer-Freise results are almost identical. The deviation at two ends are unavoidable in simulations with finite difference method. The reproducible diffusion coefficients demonstrate the robustness of pydiffusion.

FSA has been applied to many alloy systems for the past several years [6, 8]. **Figure 3** illustrates the comparison between simulation and experimental data as a testing result of `FSA()`. Extracted diffusion coefficients is shown in **Figure 3** for the Ni-Mo diffusion couple that was annealed at 1100°C for 800 h. Good agreements
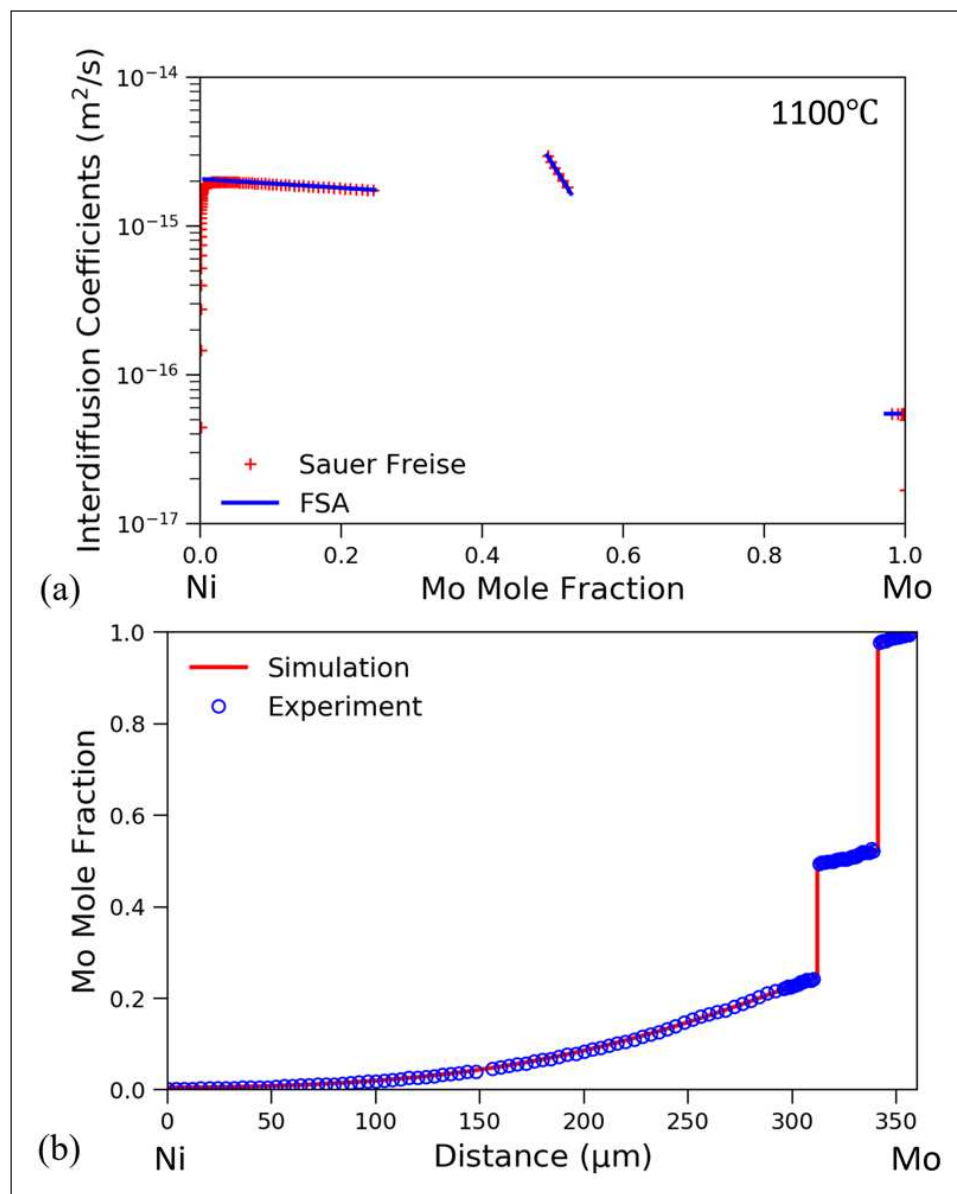
Figure 3: (a) Comparison of the input interdiffusion coefficients (lines) for the Ni-Mo system to `mphSim()` with those obtained by applying the Sauer-Freise method to the simulated profile (crosses); **(b)** Comparison between simulated diffusion profile (line) and experimental profile (open circles) of a Ni/Mo diffusion couple after being annealed at 1100°C for 800 h.

between simulation and raw data in **Figure 3** validates the robustness of FSA in pydiffusion. pydiffusion has utilized FSA to extract reliable diffusion coefficients from various types of experiments, such as liquid-solid diffusion couples [17], two-step annealing diffusion multiples [9], and thin film diffusion couples [18].

(2) Availability

Operating system
pydiffusion can run on Linux, OSX or Windows with supported version of Python installed.

Programming language
Python 3.5+

Additional system requirements
None.

Dependencies
NumPy [19]
SciPy [19]
Pandas [20]
Matplotlib [21]

List of contributors
· Zhangqi Chen – Development and Testing
· Qiaofu Zhang – Development of an FSA matlab code (precursor of pydiffusion)
· Ji-Cheng Zhao – Project supervision

Software location
Archive
 Name: Zenodo
 Persistent identifier: https://doi.org/10.5281/zenodo.1473778

(3) Reuse potential

As the core functionality of pydiffusion, diffusion simulation and diffusion coefficient evaluation are very useful in materials research and development, especially for diffusion (mobility) database establishment and ICME-based materials design. Because a diffusion system is easy to define in pydiffusion, researchers can also utilize the simulation tools in this package to estimate diffusion length and phase growth for their future experiments. pydiffusion also provides various functions to help diffusion analysis, such as Matano plane calculation, the Hall method [22] to estimate the impurity diffusion coefficients, et al.

More features will be added to the pydiffusion package in the future to provide more simulation and analysis tools. For example, the functions to implement 2-dimenstional (2D) numerical simulations will be provided to simulate diffusion in 2D. Various thermodynamics and kinetics modelling tools based on CALPHAD approach will also be included in pydiffusion so that users can perform mobility assessment using various diffusion datasets. These features will make pydiffusion a general and convenient tool to perform efficient and accurate diffusion data analysis and forward simulations.

New users can find example documentation in the repository, which illustrates usage of the core functions in pydiffusion. Example datasets and scripts are also provided. All methods in pydiffusion provide full explanations in their documentation strings. Users who are interested in collaboration or seeking technical support are welcome to contact authors by email.

Competing Interests

The authors have no competing interests to declare.

References

1. **Glicksman, M E** 2000 Diffusion in Solids: Field Theory, Solid-State Principles, and Applications, Wiley.
2. **Borgenstam, A, Höglund, L, Ågren, J** and **Engström, A** 2000 DICTRA, a tool for simulation of diffusional transformations in alloys. *J. Phase Equilibria*, 21: 269–280. DOI: https://doi.org/10.1361/105497100770340057
3. **Chen, S-L, Cao, W-S, Zhang, F, Zhang, C, Zhu, J** and **Zhang, J-Y** 2013 Development of a computational tool for materials design. *Adv. Manuf.*, 1: 123–129. DOI: https://doi.org/10.1007/s40436-013-0021-6
4. **Allison, J, Backman, D** and **Christodoulou, L** 2006 Integrated computational materials engineering: A new paradigm for the global materials profession. *JOM*, 58: 25–27. DOI: https://doi.org/10.1007/s11837-006-0223-5
5. **Zhao, J-C** 2006 Combinatorial approaches as effective tools in the study of phase diagrams and composition–structure–property relationships. *Prog. Mater. Sci.*, 51: 557–631. DOI: https://doi.org/10.1016/j.pmatsci.2005.10.001
6. **Zhang, Q** and **Zhao, J-C** 2013 Extracting interdiffusion coefficients from binary diffusion couples using traditional methods and a forward-simulation method. *Intermetallics*, 34: 132–141. DOI: https://doi.org/10.1016/j.intermet.2012.11.012
7. **Zhang, Q, Chen, Z, Zhong, W** and **Zhao, J-C** 2017 Accurate and efficient measurement of impurity (dilute) diffusion coefficients without isotope tracer experiments. *Scr. Mater.*, 128: 32–35. DOI: https://doi.org/10.1016/j.scriptamat.2016.09.040
8. **Zhang, Q** and **Zhao, J-C** 2014 Impurity and interdiffusion coefficients of the Cr–X (X = Co, Fe, Mo, Nb, Ni, Pd, Pt, Ta) binary systems. *J. Alloys Compd.*, 604: 142–150. DOI: https://doi.org/10.1016/j.jallcom.2014.03.092
9. **Chen, Z, Liu, Z-K** and **Zhao, J-C** 2018 Experimental determination of impurity and interdiffusion coefficients in seven Ti and Zr binary systems using diffusion multiples. *Metall. Mater. Trans. A.*, 49: 3108–3116. DOI: https://doi.org/10.1007/s11661-018-4645-9
10. **Zhong, W** and **Zhao, J-C** 2017 First reliable diffusion coefficients for Mg-Y and additional reliable diffusion coefficients for Mg-Sn and Mg-Zn. *Metall. Mater. Trans. A.*, 48: 5778–5782. DOI: https://doi.org/10.1007/s11661-017-4378-1
11. **Vivès, S, Bellanger, P, Gorsse, S, Wei, C, Zhang, Q** and **Zhao, J-C** 2014 Combinatorial approach based on interdiffusion experiments for the design of thermoelectrics: Application to the Mg$_2$(Si,Sn) alloys. *Chem. Mater.*, 26: 4334–4337. DOI: https://doi.org/10.1021/cm502413t
12. **Liu, Y, Liu, D, Du, Y, Liu, S, Kuang, D, Deng, P, Zhang, J, Du, C, Zheng, Z** and **He, X** 2017 Calculated interdiffusivities resulting from different fitting functions applied to measured concentration profiles in Cu-rich fcc Cu-Ni-Sn alloys at 1073 K. *J. Min. Metall. Sect. B Metall.*, 53: 255–262. DOI: https://doi.org/10.2298/JMMB170626022L
13. **Delhaise, A M** and **Perovic, D D** 2018 Study of solid-state diffusion of Bi in polycrystalline Sn using electron probe microanalysis. *J. Electron. Mater.*, 47: 2057–2065. DOI: https://doi.org/10.1007/s11664-017-6011-x
14. **Hoffman, JD** and **Frankel, S** 2001 Numerical Methods for Engineers and Scientists, CRC Press.
15. **Sauer, F** and **Freise, V** 1962 Diffusion in binären gemischen mit volumenänderung. *Zeitschrift Für Elektrochemie, Berichte Der Bunsengesellschaft Für Phys.*

Chemie, 66: 353–362. DOI: https://doi.org/10.1002/bbpc.19620660412

16. **Zhu, L, Zhang, Q, Chen, Z, Wei, C, Cai, G-M, Jiang, L, Jin, Z** and **Zhao, J-C** 2017 Measurement of interdiffusion and impurity diffusion coefficients in the bcc phase of the Ti–X (X = Cr, Hf, Mo, Nb, V, Zr) binary systems using diffusion multiples. *J. Mater. Sci.*, 52: 3255–3268. DOI: https://doi.org/10.1007/s10853-016-0614-0

17. **Zhong, W** and **Zhao, J-C** 2017 First experimental measurement of calcium diffusion in magnesium using novel liquid-solid diffusion couples and forward-simulation analysis. *Scr. Mater.*, 127: 92–96. DOI: https://doi.org/10.1016/j.scriptamat.2016.09.008

18. **Delhaise, A M, Chen, Z** and **Perovic, D D** 2018 Solid-state diffusion of Bi in Sn: effects of β-Sn grain

orientation. *J. Electron. Mater.*, 8. DOI: https://doi.org/10.1007/s11664-018-6621-y

19. **van der Walt, S, Colbert, S C** and **Varoquaux, G** 2011 The NumPy array: A structure for efficient numerical computation. *Comput. Sci. Eng.*, 13: 22–30. DOI: https://doi.org/10.1109/MCSE.2011.37

20. **McKinney, W** 2010 Data structures for statistical computing in Python. *Proc. 9th Python Sci. Conf.*, 1697900, 51–56. http://conference.scipy.org/proceedings/scipy2010/mckinney.html.

21. **Hunter, J D** 2007 Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.*, 9: 90–95. DOI: https://doi.org/10.1109/MCSE.2007.55

22. **Hall, L D** 1953 An analytical method of calculating variable diffusion coefficients. *J. Chem. Phys.*, 21: 87. DOI: https://doi.org/10.1063/1.1698631