LA-UR-22-28074

PyDislocDyn — Code Manual

Daniel N. Blaschke

February 13, 2024

Los Alamos National Laboratory, Los Alamos, NM, 87545, USA

E-mail: dblaschke@lanl.gov

Contents

1 Introduction

PyDislocDyn [1] is a suite of python programs designed to perform various calculations for basic research in dislocation dynamics in metals with various crystal symmetries in the continuum limit.

Its main features summarize as:

- compute averages of elastic constants (various methods, see [2])
- compute the steady-state displacement (gradient) field of a dislocation for arbitrary character angle in an arbitrary crystal geometry [3, 4]; a generalization to accelerating dislocations is in progress and so far has been implemented for pure screw dislocations [5] and pure edge dislocations [6]
- compute theoretical limiting velocities of dislocations in arbitrary slip systems and with arbitrary character angle [7–10]
- compute the elastic strain energy and line tension for any dislocation [11]
- compute the dislocation drag coefficient B from phonon wind for any dislocation at any velocity [12–15] or stress [16–18].

This manual is intended as a brief introduction on how to use the most common features of PyDislocDyn 1.2.9 and higher [1]. It does not describe every option of every function or method, and we refer to the doc-string of those functions, classes and methods for further details.
PyDislocDyn is developed by the present author and can be downloaded from
github.com/dblaschke-LANL/PyDislocDyn
where it is distributed under the BSD-3 license, (Copyright, Triad National Security, LLC; C Number: C18073), see the included `license.txt` for details.

2 Requirements

- PyDislocDyn 1.2.5 and higher requires Python 3.8 or higher[1] and it is recommended to use the latest version. As of PyDislocDyn 1.2.9, required modules include Numpy ≥ 1.17, Scipy ≥ 1.6, Sympy ≥ 1.6, Matplotlib ≥ 3.1, and Pandas ≥ 1.1 (though Pandas ≥ 1.3 including its optional dependency Jinja2 is recommended).
- Optional, but highly recommended in order to conduct drag coefficient calculations with reasonable speed are: Numba ≥ 0.47 (a just-in-time compiler for python), Joblib ≥ 0.14 (for parallelization), and a Fortran compiler supported by numpy.f2py. To compile the optional Fortran subroutines run '`python -m numpy.f2py -c subroutines.f90 -m subroutines`'.
  Optionally, the subroutines can also be compiled with OpenMP parallelization support by passing the appropriate options for your compiler. For example, with gfortran on Linux or MacOS with python ≤ 3.11 compile with:
  '`python -m numpy.f2py --f90flags=-fopenmp -lgomp -c subroutines.f90 -m subroutines`';
  or with python ≥ 3.12:

[1] The last version to support Python 2.7 was PyDislocDyn 1.2.0, Python 3.5 was supported up to version 1.2.3, and Python 3.6 up to version 1.2.4

'python -m numpy.f2py --dep=openmp -c subroutines.f90 -m subroutines'.
The number of threads used by the subroutines is then controlled by environment variable OMP_NUM_THREADS. Unless set by the user prior to running python in the terminal, PyDislocDyn will attempt to choose an optimal value. Furthermore, PyDislocDyn's joblib parallelization will automatically adjust its 'Ncores' variable to avoid over-committing.

3 Running the code

PyDislocDyn 1.2.x reads its material data from input files, though PyDislocDyn also includes a selection of predefined material data to run simulations immediately. The latter are assembled in dictionaries within 'metal_data.py', see Refs. [19–49].

3.1 Using the front-end scripts

There are four front-end scripts included in PyDislocDyn which all can process one or multiple input files. They are located together with all sub-modules in folder 'pydislocdyn'. All four front-end scripts can optionally be run without input files in which case data from 'metal_data.py' will be used. By passing keyword arguments, an according subset of shipped metals are considered for the calculations. (See the dictionary keys within 'metal_data.py' for valid keywords.)

As a byproduct, sample input files are generated and saved to subfolder 'temp_pydislocdyn'. These input files can also be generated by the user directly via the functions 'writeinputfile()' or 'writeallinputfiles()' defined in 'metal_data.py'.

3.1.1 polycrystal_averaging.py

Calling 'python pydislocdyn/polycrystal_averaging.py inputfile1 inputfile2 ...' will compute average elastic constants (second and if applicable third order) using Voigt, Reuss, Hill, and (for cubic only) the Kröner method, see [2] and references therein for details. Results are written to a text file, 'averaged_elastic_constants.txt'. Since pydislocdyn (from version 1.2.9) is a 'regular package' in python, all front-end scripts can also be run via python's -m option, e.g. in the present case:
'python -m pydislocdyn.polycrystal_averaging inputfile1 inputfile2 ...'.

3.1.2 linetension_calcs.py

Calling 'python pydislocdyn/linetension_calcs.py inputfile1 inputfile2 ...' will compute the line tension of the dislocations / slip systems defined in the input files as a function of velocity 'beta' (normalized by the average transverse sound speed or by one derived from Lamé constant mu read from the input file if present) and dislocation character angle 'theta', see [11] for details on the method. The default resolution of Nbeta=500 and Ntheta=600 can be changed by passing options on the command line, e.g.:
'--Nbeta=100 --Ntheta=100'.
Option --Nbeta=0 can be abused to skip line tension calculations and to just generate the plots using results from a previous run. Additional command line options include: the resolution of the discretized polar angle --Nphi, an option to skip generating the plots via --skip_plots, an option to override the automatic number of parallel (joblib) processes to spawn via --Ncores, and more; run linetension_calcs --help for a list of all supported command line options and their

expected data types.
Results are saved as compressed text files with naming scheme 'LT_metalname.dat.xz' and results from a previous run will first be moved to 'LT_metalname.dat.bak.xz'. These result files can be conveniently read into a pandas.DataFrame by using function 'read_2dresults' which can be imported from 'pydislocdyn'. Additionally, a number of plots are generated and saved as pdf files.

3.1.3 dragcoeff_iso.py

Calling 'python pydislocdyn/dragcoeff_iso.py inputfile1 inputfile2 ...' will compute the drag coefficient of the dislocations defined in the input files in the isotropic limit as a function of (normalized) velocity 'beta' and as a function of resolved shear stress in units of mPa s, see [12, 15, 16] for details on the method. If the input file defines an anisotropic crystal, the code will first compute the averaged isotropic elastic constants to be used in the calculation. Note that there exists no accurate averaging scheme for third order elastic constants, so measured isotropic values should always be preferred, or better yet: compute the drag coefficient in the semi-isotropic approach using 'dragcoeff_semi_iso.py'. Results are written to text files 'drag_metalname.dat', and additionally a number of plots are generated and saved as pdf files.

Supported command line options (to change defaults, all optional) include: the resolution of the normalized velocities --Nbeta, the resolution of various discretized variables, --Nphi, --Nphi1, --Nq1, --Nt (just the base value as variable t is adaptively refined), an option to skip generating the plots via keyword --skip_plots, an option to override the automatic number of parallel (joblib) processes to spawn via --Ncores, additional options to be passed on to subroutine dragcoeff_iso (defined in the phononwind.py module) via --phononwind_opts="{'keyword':value,...}", and more; run dragcoeff_iso --help for a list of all supported command line options and their expected data types. Option --Ncores=0 can be abused to skip drag coefficient calculations and to just generate the plots using results from a previous run.

3.1.4 dragcoeff_semi_iso.py

Calling 'python pydislocdyn/dragcoeff_semi_iso.py inputfile1 inputfile2 ...' will compute the drag coefficient in units of mPa s of the dislocations defined in the input files as a function of (normalized) velocity 'beta' and dislocation character angle 'theta' as well as resolved shear stress, see [13, 14, 17] for details on the semi-isotropic approximation to dislocation drag. The default resolution of Nbeta=99 and Ntheta=21 can be changed by passing options on the command line, e.g.: '--Nbeta=50 --Ntheta=11'.
Additional command line options include: the resolution of various discretized variables, --Nphi, --Nphi1, --Nq, --Nq1, --NphiX, --Nt (just the base value as variable t is adaptively refined), an option to skip generating the plots via --skip_plots, an option to override the automatic number of parallel (joblib) processes to spawn via --Ncores, additional options to be passed on to subroutine dragcoeff_iso via --phononwind_opts="{'keyword':value,...}", and more; run dragcoeff_semi_iso --help for a list of all supported command line options and their expected data types. Option --Ncores=0 can be abused to skip drag coefficient calculations and to just generate the plots using results from a previous run.

By default, the lowest dislocation limiting velocity is computed on the fly for every character angle and drag coefficient calculations will be skipped beyond these velocities (on a per character angle basis, see 'inf' entries in the resulting 'drag_anis_metalname.dat.xz' files). This behavior

is controlled by Boolean option '--skiptransonic'. Additionally, a number of plots are generated and saved as pdf files. Drag coefficient results (as functions of velocity) are written to files with naming scheme 'drag_anis_metalname.dat.xz'; these can be conveniently read into a pandas dataframe by using function 'read_2dresults' which can be imported from 'pydislocdyn'.

3.2 Anatomy of an input file

Input files are text files assigning values to keywords known to PyDislocDyn. All entries adhere to the format
`keyword = value`
where all numerical values must be set in SI units. Comments are allowed and start with '#'. Required keywords include:

- sym: defines the crystal symmetry, recognized values are iso, fcc, bcc, cubic (for cubic I), hcp, tetr (for tetragonal I), trig (for trigonal I), tetr2 (for tetragonal II), orth (for orthorhombic), mono (for monoclinic), and tric (for triclinic), see [50] for an overview over these crystal systems.
- rho: the material density
- lattice constant(s) and angles: lattice constant a is always required, lcb and c may be required depending on sym (keyword b is reserved for the Burgers vector direction, see below); angles between lattice basis vectors (only required by lower symmetries where they cannot be inferred from keyword sym) are denoted by alpha, beta, gamma and are provided in degrees (not radians). The unit cell volume is then automatically computed.
- Second order elastic constants (SOEC): for sym in one of iso–tetr, every required elastic constant is set with its own keyword (e.g. c44 etc.); for lower symmetries, all values are assigned to one keyword, cij, and must be a assembled in one line separated by commas. The order is ascending, i.e. c11 comes before c12 etc.; see [50] for further details on which elastic constants need to be provided for each crystal system.
- n0 or Millern0: requires comma separated values and denotes the slip plane normal in Cartesian coordinates or Miller indices for planes (i.e. in reciprocal space); its length is ignored, Miller indices Millern0 (if provided) are automatically converted to Cartesian coordinates n0, and n0 is automatically normalized.
- b or Millerb: requires comma separated values and denotes the Burgers vector direction in Cartesian coordinates or Miller indices; its length is only used to infer the Burgers vector length burgers (unless the latter is provided explicitly in the input file, see optional keywords below). If provided, Miller indices Millerb are automatically converted to Cartesian coordinates b, and b is automatically normalized.

The following keywords are optional (depending on the intended use case):

- burgers defines the Burgers vector length; if omitted it is inferred from Millerb or b (in which case their length matters).
- name: a name for your instance of the Dislocation class; defaults to the file name of the input file. Though not strictly necessary it is recommended to set a name.

- `T`: the material temperature where density and elastic constants were measured (or calculated); defaults to 300K if omitted.
- `lam` and `mu`: polycrystal averages for the Lamé constants; if omitted, they are calculated by averaging over the (anisotropic) SOECs.
- Third order elastic constants (TOEC), these are required for phonon wind (dislocation drag) calculations: for `sym` in one of `iso–tetr`, every required elastic constant is set with its own keyword (e.g. `c123` etc.); for lower symmetries, all values are assigned to one keyword, `cijk`, and must be a assembled in one line separated by commas. The order is ascending, i.e. c111 comes before c112 etc.; see [50] for further details on which elastic constants need to be provided for each crystal system.
- `alpha_a`: denotes the thermal expansion coefficient; only used if dislocation drag is computed for multiple temperatures >=T.

3.3 Working with the classes

3.3.1 The 'Dislocation' class

To work with an instance of PyDislocDyn's main class, the `Dislocation` class, it is recommended to import the following:
`'from pydislocdyn import readinputfile'`
and then to use that function to initialize an instance of the Dislocation class using an input file. By default, only pure screw and pure edge dislocations are initialized; to initialize more than 2 character angles, set option `Ntheta` accordingly when calling `readinputfile(...)`.

Upon initialization, several quantities are automatically calculated and stored as attributes, in particular: the tensors of elastic constants in Voigt notation are saved as attributes `.C2` (SOEC) and (if applicable) `.C3` (TOEC), the (polycrystalline averages of) longitudinal and transverse sound speeds `.cl` and `.ct`, as well as the average Bulk modulus `.bulk`, Young's modulus `.young`, Poisson's ratio `.poisson`, and the edge of the first Brillouin zone in the Debye approximation `.qBZ`. If `sym` is `fcc`, `bcc`, or `cubic`, also the Zener anisotropy ratio `.Zener` and elastic constant `.cp`=$(c_{11}-c_{12})/2$ are additionally calculated. Furthermore, the dislocation line and slip directions are determined for all requested dislocation character angles `theta`.

Function `readinputfile()` defines some additional defaults and options making the initialization of the Dislocation class easier: Instead of passing an array `theta` (default: `None`), one may pass an integer `Ntheta` (default: 2) and a Boolean keyword `symmetric` (default: `True`), and array `theta` is subsequently generated with `Ntheta` entries between 0 and $\pi/2$ (or in the interval $[-\pi/2, \pi/2]$ if `symmetric=False` and the latter keyword may also be included in the input file as it depends on the slip plane geometry whether the full range needs to be considered). Boolean keyword `isotropify` finally allows to initialize an isotropic instance of the Dislocation class from an input file for an anisotropic crystal by automatically averaging the elastic constants.

Once initialized, many calculations are available via the classes methods such as:

- `.compute_Lame()` to calculate the isotropic averages of the SOECs
- `.computesound(v)` to calculate the sound speeds for a wave propagating in direction v

- `.computevcrit()` to calculate limiting velocities for all character angles initialized in the class
- `.findvcrit_smallest()` returns the lowest limiting velocity for all character angles (independent of `Ntheta`)
- `.findRayleigh()` calculates the Rayleigh wave speed for every initialized character angle, i.e. for the slip directions of dislocations of every character angle, see [10].
- `.find_vRF()` calculates 'radiation-free' velocities, i.e. transonic edge dislocation velocities that theory predicts are free of shock waves, see [51, 52].
- `.computeuij(beta)` calculates the steady-state dislocation displacement gradient field in Cartesian coordinates aligned with the crystal as a function of polar angle `phi` in the plane perpendicular to the dislocation line using the Stroh / integral method [3] for all initialized character angles. Argument `beta` is the normalized velocity (e.g. `v/ct` by default, where `ct` is the averaged transverse sound speed). The result is stored in attribute `.uij`.
- `.computeuk(beta)` calculates the steady-state dislocation displacement field (no gradient) in Cartesian coordinates aligned with the crystal as a function of polar angle `phi` in the plane perpendicular to the dislocation line using the Stroh / integral method [3] for all initialized character angles. Argument `beta` is the normalized velocity (e.g. `v/ct` by default, where `ct` is the averaged transverse sound speed). The result is stored in attribute `.uk`.
- `.alignC2()` computes the tensor of SOEC in coordinates aligned with the dislocation line and slip plane normal; it's result, '.C2aligned', contains an array of SOEC tensors whose entries correspond to the character angles.
- `.computerot()` is automatically called by `.alignC2()` and calculates the rotation matrices for all character angles necessary to align coordinates with the dislocation line and slip plane normal
- `.alignuij()` rotates .uij (the result of `.computeuij()`) into coordinates aligned with the dislocation line and slip plane normal and stores the result in `.uij_aligned`; `.computerot()` (or `alignC2()`) must be called first.
- `.alignuk()` rotates .uk (the result of `.computeuk()`) into coordinates aligned with the dislocation line and slip plane normal and stores the result in `.uk_aligned`; `.computerot()` (or `alignC2()`) must be called first.
- `.computeuij_acc_screw(a,beta)` calculates the dislocation displacement gradient field of a screw dislocation accelerating from rest at rate `a` at the time where its normalized velocity matches `beta`, provided the plane perpendicular to the dislocation line is a reflection plane. The result is stored in `.uij_acc_screw_aligned` and, as the name suggests, is calculated in coordinates aligned with the dislocation line and slip plane normal[2].

[2]Note: this method and attribute were renamed .computeuij_acc() → .computeuij_acc_screw() and .uij_acc_aligned → .uij_acc_screw_aligned in PyDislocDyn 1.2.7.

- `.computeuij_acc_edge(a,beta)` likewise calculates the dislocation displacement gradient field of an edge dislocation accelerating from rest at rate `a` at the time where its normalized velocity matches `beta`, provided the plane perpendicular to the dislocation line is a reflection plane. The result is stored in `.uij_acc_edge_aligned` and, as the name suggests, is calculated in coordinates aligned with the dislocation line and slip plane normal.

- Method `.plotdisloc()` generates a color-mesh plot of one component (choose with keyword `'component'`) of the dislocation displacement gradient field of one character angle (choose with keyword `'character'`) and saves it as a pdf file. If velocity `beta` is set, the dislocation field is computed on the fly (and other arguments of this function are passed along to the `.computeuij()` method). Setting `showplt=True` will display the plot inline in addition to saving a pdf; this option requires that the user first set `'%matplotlib inline'`, as the default matplotlib back-end set by PyDislocDyn is `'Agg'` (in order to facilitate running in a remote terminal session).

- `.computeEtot()` will calculate the elastic strain self energy of the dislocation from the previously calculated displacement gradient field for all character angles and stores the result in attribute `.Etot`

- `.computeLT()` will calculate the line tension from `.Etot` for all character angles and stores its result in `.LT`. Warning: this calculation requires a high resolution in character angle, `Ntheta>=100` is recommended. Furthermore, since numerical differentiation is involved, the array `.LT` will be shorter than the array `.Etot` by two entries, i.e. in order to get results for pure screw and edge, `theta` must contain one additional angle past each end of the interval.

3.3.2 The 'metal_props' class

The `metal_props` class is one of the parents of the `Dislocation` class. It lets the user who is not interested in dislocations work with elastic constants (including polycrystal averages), compute sound speeds, convert Miller indices to Cartesian indices and more. Instances of this class can be populated using the same input file format discussed above in section 3.2, though keywords relating to dislocations (i.e. `b`, `n0`, `Millerb`, `Millern0`, `burgers`) can be omitted. In order to read an input file to initialize an instance of the `metal_props` class, import the following:
`'from pydislocdyn.polycrystal_averaging import readinputfile'`

3.3.3 The 'strain_poly' class

The `strain_poly` class can be used to determine which type of infinitesimal strain deformation is sensitive to which elastic constants and is thus helpful in preparing simulations using third party software to calculate elastic constants. In particular, it can calculate the polynomial following from a Taylor expansion in y as a function of (symbolic) elastic constants, where y parameterizes the deformation, see e.g. [18, 53] and references therein for details. To use this class, run
`'from pydislocdyn import strain_poly'`
Two arguments are required to initialize an instance: The sympy symbol parametrizing the deformation, `y` (defaults to `sympy.symbols('y')`), and a keyword selecting the crystal system, `sym` (defaults to `cubic`, all values known to the `Dislocation` class are allowed also here). Method `.generate_poly(epsilon,order=3)` calculates a Taylor expansion to the requested order in `y` whose coefficients are (sympy symbols of) the elastic constants; `epsilon` is the infinitesimal strain

in Voigt notation (i.e. a list with 6 entries) which must depend on symbol `y`. An additional helper method, `.generate_alpha(epsilon)`, can be used to calculate the symmetric deformation matrix in Cartesian coordinates that would lead to infinitesimal strain `epsilon`; the latter need not necessarily depend on symbol y, but may be purely numeric. Both methods support the optional Boolean keyword `preserve_volume` (defaults to `False`), which may be set to force the deformation to preserve the unit cell volume by ensuring that the deformation matrix has determinant 1.

3.4 Using the modules

The most common classes and functions can be accessed directly by importing the main package: '`pydislocdyn`'. In particular this includes the following objects:

- `Voigt` and `UnVoigt` to convert tensors between Voigt notation and Cartesian coordinates.
- `elasticC2` and `elasticC3` to generate the tensors (i.e. the multidimensional numpy arrays) of second and third order elastic constants in Cartesian coordinates (default) or in Voigt notation (set `voigt=True`).
- `elasticS2` and `elasticS3` to calculate the second and third order compliance tensors from the elastic (or stiffness) tensors.
- the `strain_poly` class as described above in Section 3.3.3.
- helper functions `writeinputfile` and `writeallinputfiles` to write data defined in the python dictionaries of sub-module `metaldata` to PyDislocDyn input files.
- the `Dislocation` class as well as a function, `readinputfile`, to populate an instance from an input file, see Sections 3.3.1 and 3.2.
- `read_2dresults`, to read files containing results from line tension or drag coefficient calculations into a `pandas.DataFrame` as described in Section 3.1.
- the `metal_props` class, see Section 3.3.2; a function to populate an instance from an input file is available from sub-module `polycrystal_averaging.readinputfile`.
- function `phonondrag` to calculate the drag coefficient B in units of mPa s as a function of velocity for an instance of the Dislocation class.
- function `B_of_sigma` to calculate the drag coefficient B as a function of stress σ from a previously determined fitting function for $B(v)$ as a function of dislocation velocity v.
- function `plotuij` generates a heat map plot of a 2-dimensional dislocation field.

Additional functions are available within the sub-modules:

- `pydislocdyn.elasticconstants` with functions related to elastic constants,
- `pydislocdyn.metaldata` which contains a number of python dictionaries of material data included in PyDislocDyn (see `pydislocdyn.metaldata.all_metals`, which is a set of keywords for metals included in this sub-module),

- `pydislocdyn.phononwind` which contains function `phonondrag` to calculate the drag coefficient B as mentioned in the list above, as well as its subroutines,
- `pydislocdyn.dislocations` which contains the `StrohGeometry` class (one of two parents of the `Dislocation` class) and functions used under the hood by the `Dislocation` class.
- `pydislocdyn.subroutines`, if it is compiled using `numpy.f2py` and a Fortran compiler, provides some faster alternatives to a subset of functions defined in `pydislocdyn.phononwind` and `pydislocdyn.dislocations`; the code will use them automatically if they are available (as indicated by the Boolean variable `pydislocdyn.usefortran`).
- the front-end scripts which are described in Sec. 3.1, `pydislocdyn.polycrystal_averaging`, `pydislocdyn.linetension_calcs`, and `pydislocdyn.dragcoeff_semi_iso`, are also submodules; classes and functions useful to the user are aliased to the top level and are thus available directly under `pydislocdyn`, as listed above.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy under contract 89233218CNA000001. In particular, the author is grateful for the support of the Physics and Engineering Models sub-program element of the Advanced Simulation and Computing program.

References

[1] D. N. Blaschke, *PyDislocDyn*, version 1.2.9, 2018–2024, URL: https://github.com/dblaschke-LANL/PyDislocDyn.

[2] D. N. Blaschke, "Averaging of elastic constants for polycrystals", *J. Appl. Phys.* **122** (2017) 145110, arXiv:1706.07132 [cond-mat.mtrl-sci].

[3] D. J. Bacon, D. M. Barnett, and R. O. Scattergood, "Anisotropic continuum theory of lattice defects", *Prog. Mater. Sci.* **23** (1980) 51–262.

[4] J. P. Hirth and J. Lothe, *Theory of Dislocations*, second edition, (New York: Wiley, 1982).

[5] D. N. Blaschke, "A general solution for accelerating screw dislocations in arbitrary slip systems with reflection symmetry", *J. Mech. Phys. Solids* **152** (2021) 104448, arXiv:2009.00167 [cond-mat.mtrl-sci].

[6] D. N. Blaschke, K. Dang, S. Fensin, and D. J. Luscher, "Properties of accelerating edge dislocations in arbitrary slip systems with reflection symmetry", *Materials* **16** (2023) 4019, arXiv:2303.10461 [cond-mat.mtrl-sci].

[7] D. N. Blaschke, "How to determine limiting velocities of dislocations in anisotropic crystals", *J. Phys.: Cond. Mat.* **33** (2021) 503005, arXiv:2107.01220 [cond-mat.mtrl-sci].

[8] D. N. Blaschke, J. Chen, S. Fensin, and B. Szajewski, "Clarifying the definition of 'transonic' screw dislocations", *Phil. Mag.* **101** (2021) 997–1018, arXiv:2008.13760 [cond-mat.mtrl-sci].

[9] L. J. Teutonico, "Dynamical behavior of dislocations in anisotropic media", *Phys. Rev.* **124** (1961) 1039–1045.

[10] D. M. Barnett, J. Lothe, K. Nishioka, and R. J. Asaro, "Elastic surface waves in anisotropic crystals: a simplified method for calculating Rayleigh velocities using dislocation theory", *J. Phys. F: Met. Phys.* **3** (1973) 1083–1096.

[11] D. N. Blaschke and B. A. Szajewski, "Line tension of a dislocation moving through an anisotropic crystal", *Phil. Mag.* **98** (2018) 2397–2424, arXiv:1711.10555 [cond-mat.mtrl-sci].

[12] D. N. Blaschke, E. Mottola, and D. L. Preston, *On the velocity dependence of the dislocation drag coefficient from phonon wind*, tech. rep. LA-UR-16-24559, Los Alamos Natl. Lab., 2018.

[13] D. N. Blaschke, "Velocity dependent dislocation drag from phonon wind and crystal geometry", *J. Phys. Chem. Solids* **124** (2019) 24–35, arXiv:1804.01586 [cond-mat.mtrl-sci].

[14] D. N. Blaschke, "Properties of dislocation drag from phonon wind at ambient conditions", *Materials* **12** (2019) 948, arXiv:1902.02451 [cond-mat.mtrl-sci].

[15] D. N. Blaschke, E. Mottola, and D. L. Preston, "Dislocation drag from phonon wind in an isotropic crystal at large velocities", *Phil. Mag.* **100** (2020) 571–600, arXiv:1907.00101 [cond-mat.mtrl-sci].

[16] D. N. Blaschke, A. Hunter, and D. L. Preston, "Analytic model of the remobilization of pinned glide dislocations: Including dislocation drag from phonon wind", *Int. J. Plast.* **131** (2020) 102750, arXiv:1912.08851 [cond-mat.mtrl-sci].

[17] D. N. Blaschke and D. J. Luscher, "Dislocation drag and its influence on elastic precursor decay", *Int. J. Plast.* **144** (2021) 103030, arXiv:2101.10497 [cond-mat.mtrl-sci].

[18] D. N. Blaschke, L. Burakovsky, and D. L. Preston, "On the temperature and density dependence of dislocation drag from phonon wind", *J. Appl. Phys.* **130** (2021) 015901, arXiv:2104.08650 [cond-mat.mtrl-sci].

[19] R. W. Hertzberg, R. P. Vinci, and J. L. Hertzberg, *Deformation and Fracture Mechanics of Engineering Materials*, fifth edition, (Wiley, 2012).

[20] S. G. Epstein and O. N. Carlson, "The elastic constants of nickel-copper alloy single crystals", *Acta Metall.* **13** (1965) 487–491.

[21] R. Lowrie and A. M. Gonas, "Single-crystal elastic properties of tungsten from 24° to 1800°C", *J. Appl. Phys.* **38** (1967) 4505–4509.

[22] J. F. Thomas, "Third-order elastic constants of aluminum", *Phys. Rev.* **175** (1968) 955–962, Erratum-ibid. 181 (1969) 1370.

[23] Y. Hiki and A. V. Granato, "Anharmonicity in noble metals; higher order elastic constants", *Phys. Rev.* **144** (1966) 411–419.

[24] J. Leese and A. E. Lord, "Elastic stiffness coefficients of single-crystal iron from room temperature to 500°C", *J. Appl. Phys.* **39** (1968) 3986–3988.

[25] F. F. Voronov, V. M. Prokhurov, E. L. Gromnitskaya, and G. G. Ilina, "Second- and third-order elastic moduli of a molybdenum single crystal", *Phys. Met. Metallogr.* **45** (1978) 123, [*Fiz. Met. Metalloved.* **45** (1978) 1263].

[26] G. A. Alers, J. R. Neighbours, and H. Sato, "Temperature dependent magnetic contributions to the high field elastic constants of nickel and an Fe-Ni alloy", *J. Phys. Chem. Solids* **13** (1960) 40–55.

[27] G. A. Saunders and Y. K. Yoğurtçu, "The effect of hydrostatic and uniaxial pressure on the elastic constants of cadmium", *J. Phys. Chem. Solids* **47** (1986) 421–427.

[28] B. E. Powell and M. J. Skove, "Linear and volume compressibilities and isothermal third-order elastic constants", *J. Appl. Phys.* **56** (1984) 1548–1549.

[29] E. R. Naimon, "Third-order elastic constants of magnesium. I. experimental", *Phys. Rev. B* **4** (1971) 4291–4296.

[30] G. W. C. Kaye and T. H. Laby, *Tables of Physical and Chemical Constants*, web edition, 2004, URL: https://web.archive.org/web/20190506031327/http://www.kayelaby.npl.co.uk/.

[31] M. W. Riley and M. J. Skove, "Higher-order elastic constants of copper and nickel whiskers", *Phys. Rev. B* **8** (1973) 466–474.

[32] K. D. Swartz, W. B. Chua, and C. Elbaum, "Third-order elastic constants of tin and of a tin-indium alloy", *Phys. Rev. B* **6** (1972) 426–435.

[33] R. Ramji Rao and C. S. Menon, "Lattice dynamics, third-order elastic constants, and thermal expansion of titanium", *Phys. Rev. B* **7** (1973) 644–650.

[34] K. D. Swartz and C. Elbaum, "Third-order elastic constants of zinc", *Phys. Rev. B* **1** (1970) 1512–1517.

[35] A. Singh, R. P. S. Rathore, and R. M. Agrawal, "Phonons and elastic constants for scandium, zirconium and magnesium", *Acta Phys. Hung.* **72** (1992) 133–140.

[36] G. D. Barrera and A. Batana, "Lattice dynamics, thermal expansion, and third-order elastic constants of seminoble metals", *phys. stat. sol. (b)* **179** (1993) 59–75.

[37] D. C. Jiles and S. B. Palmer, "Third-order elastic constants of erbium", *J. Appl. Phys.* **52** (1981) 1113–1115.

[38] Y. K. Yoğurtçu, G. A. Saunders, and P. C. Riedi, "Third-order elastic constants and acoustic-mode vibrational anharmonicity of cobalt", *Phil. Mag. A* **52** (1985) 833–846.

[39] R. Srinivasan and K. S. Girirajan, "The second and third order elastic constants of alkali metals", *J. Phys. Chem. Solids* **34** (1973) 611–620.

[40] R. Ramji Rao and A. Ramanand, "Third-order elastic constants of uniaxial crystals", *phys. stat. sol. (a)* **58** (1980) 11–36.

[41] J. R. Rumble, ed., *CRC Handbook of Chemistry and Physics*, 102nd edition, (CRC Press, 2021).

[42] Y. K. Vekilov, O. M. Krasilnikov, A. V. Lugovskoy, and Y. E. Lozovik, "Higher-order elastic constants and megabar pressure effects of bcc tungsten: Ab initio calculations", *Phys. Rev. B* **94** (2016) 104114.

[43] T. Suzuki, "Second- and third-order elastic constants of aluminum and lead", *Phys. Rev. B* **3** (1971) 4007–4014, Erratum-ibid. 4 (1971) 3779.

[44] W. Wasserbäch, "Third-order constants of a cubic quasi-isotropic solid", *phys. stat. sol. (b)* **159** (1990) 689–697.

[45] A. Seeger and O. Buck, "Die experimentelle Ermittlung der elastischen Konstanten höherer Ordnung", *Z. Naturf.* **15a** (1960) 1056–1067.

[46] L. J. Graham, H. Nadler, and R. Chang, "Third-order elastic constants of single-crystal and polycrystalline columbium", *J. Appl. Phys.* **39** (1968) 3025–3033.

[47] R. T. Smith, R. Stern, and R. W. B. Stephens, "Third-order elastic moduli of polycrystalline metals from ultrasonic velocity measurements", *J. Acoust. Soc. Amer.* **40** (1966) 1002–1008.

[48] H. Kiewel, L. Fritsche, and T. Reinert, "Calculation of nonlinear effective elastic constants of polycrystalline materials", *J. Appl. Phys.* **79** (1996) 3963–3966.

[49] G. V. Samsonov, *Handbook of the Physicochemical Properties of the Elements*, (New York: IFI/Plenum, 1968).

[50] K. Brugger, "Pure modes for elastic waves in crystals", *J. Appl. Phys.* **36** (1965) 759–768.

[51] H. Gao, Y. Huang, P. Gumbsch, and A. J. Rosakis, "On radiation-free transonic motion of cracks and dislocations", *J. Mech. Phys. Solids* **47** (1999) 1941–1961.

[52] D. N. Blaschke, T. Duong, and M. J. Demkowicz, "Comparing theoretical predictions of radiation-free velocities of edge dislocations to molecular dynamics simulations", *Phys. Rev. B* **108** (2023) 224102, `arXiv:2305.06980 [cond-mat.mtrl-sci]`.

[53] J. Gu, C. Wang, B. Sun, W. Zhang, and D. Liu, "High-pressure third-order elastic constants of MgO single crystal: First-principles investigation", *Z. Naturforsch.* **74** (2019) 447–456.