EasyDD Startup Guide

Oxford Materials

March 5, 2021

Contents

1 Getting EasyDD

EasyDD is a MATLAB-based discrete dislocation plasticity code developed in Oxford based on DDLab. The official build is publicly available to be downloaded from GitHub. To begin working on this code without affecting the original repository, you may fork it to your own repository. As you make modifications in your repository, the original repository may change; to update these changes in your own repository, you must sync the fork.

2 Input file

To run a simulation, you must first specify and run an input file. This generates all of the material-specific parameters and specifies subprocesses which EasyDD will use to run the simulation. Subprocesses are defined as function handles and may be custom-made by the user. If a mandatory parameter is failed to be defined in the input file, a default value will be used so that the simulation can run anyways.

2.1 Input variables

Material parameters

- `rn`: $N \times 4$ matrix designating the positions and `flags` of all N nodes

$$\texttt{rn} = \begin{bmatrix} x_1 & y_1 & z_1 & \texttt{flag}_1 \\ x_2 & y_2 & z_2 & \texttt{flag}_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & \texttt{flag}_N \end{bmatrix}$$

`flag` = 0: mobile node; `flag` = 7: immobile node; `flag` = 6: node only allowed to move on its surface; `flag` = 67: virtual node at pseudo-infinity

- `links`: $M \times 8$ matrix designating the start and end node `id`'s, the normalised burgers vector (b_{xi}, b_{yi}, b_{zi}), and slip plane normal (n_{xi}, n_{yi}, n_{zi}) for each of the M segments

$$\texttt{links} = \begin{bmatrix} \alpha & \beta & b_{x1} & b_{y1} & b_{z1} & n_{x1} & n_{y1} & n_{z1} \\ \beta & \gamma & b_{x2} & b_{y2} & b_{z2} & n_{x2} & n_{y2} & n_{z2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \delta & \epsilon & b_{xM} & b_{yM} & b_{zM} & n_{xM} & n_{yM} & n_{zM} \end{bmatrix}$$

- `MU`: material's shear modulus
- `NU`: material's Poisson's ratio
- `a`: dislocation core isotropic spreading radius
- `Ec`: energy per unit length of the elastic contribution of the core of a screw dislocation
- `Bcoeff`: drag coefficients corresponding to different dislocation orientations; definition depends on mobility law
- `rotMatrix`: rotation matrix which rotates crystal slip system axes in mobility law
- `amag`: lattice parameter in units of μm
- `mumag`: material's shear modulus in units of MPa

Nodal network hyper-parameters

- `maxconnections`: total number of segments permitted to be connected to one node

- `lmax`: maximum length permitted for any segment
- `lmin`: minimum length permitted for any segment
- `areamax`: maximum area preferred between three consecutive nodes in a dislocation
- `areamin`: minimum area preferred between three consecutive nodes in a dislocation
- `rmax`: maximum distance any node may move during a given time increment
- `rann`: maximum distance two non-consecutive segments or nodes can be from each other in order to interact
- `rntol`: error tolerance for the integrator in order to adjust the time increment
- `dt0`: maximum permitted absolute time increment
- `dt`: current absolute time increment
- `totalSimTime`: absolute maximum time duration of simulation
- `holdingTime`: absolute time duration before any loading is applied

Geometry & boundary condition variables

- `dx`, `dy`, `dz`: cantilever (cuboid) beam size in x,y,z directions
- `mx`, `my`, `mz`: number of finite elements along x,y,z directions
- `u_dot`, `sign_u_dot`, `u_bar`, `u_tilde`, . . . : boundary condition for nodal displacement
- `f_dot`, `sign_f_dot`, `f_bar`, `f_tilde`, . . . : boundary condition for nodal force

- `r_hat`: boundary condition for nodal reaction force
- `calculateLoadingFunctionArgs`: struct containing variables for loading conditions

Post-processing parameters

- `printfreq`: frequency per time-step that nodal information is printed
- `saveFreq`: frequency per time-step that workspace variables are saved
- `plotFreq`: frequency per time-step that the nodal positions are plotted
- `plim`: maximum permitted value for any of the x, y and z values plotted
- `viewangle`: angle orientation of plot display [`azimuth`,`elevation`]
- `plotArgs`: struct containing variables for plotting
- `plotFlags`: struct containing flags to enable plotting

Parallelisation & flags

- `n_threads`: number of threads per GPU block
- `para_scheme`: parallelisation scheme for traction calculations (`para_scheme` = 0: none; `para_scheme` = 1: over dislocations; `para_scheme` = 2: over surface elements)
- `CUDA_flag`: enables compiling of CUDA code
- `CUDA_segseg_flag`: enables seg-seg force computation parallelisation
- `doremesh`: enables remesh phase for each time-step
- `docollision`: enables collision phase for each time-step

- `doseparation`: enables separation phase for each time-step
- `dovirtmesh`: enables virtual remeshing phase for each time-step

Subprocesses

- `mobility`: specifies mobility law function subprocess (e.g. `@mobfcc0`)
- `integrator`: specifies integrator function subprocess (e.g. `@int_trapezoid`)
- `simType`: specifies simulation geometry and assigns types of boundary conditions to degrees of freedom (e.g. `@cantileverBending`)
- `calculateLoading`: specifies general loading conditions (e.g. `@constantLoading`)
- `loading`: calculates time-dependent boundary conditions at degrees of freedom (e.g. `@displacementControl`)
- `calculateTractions`: specifies how the tractions on surface nodes are calculated (e.g. `@calculateAnalyticTractions`)

2.2 Parameter constraints

It is important that the input parameters fulfill a set of logical constraints which are imposed by the simulation geometry and nodal network. Poorly defined parameters may result in the simulation generating numerical instabilities and nonphysical results. In the following, some recommended constraints for higher accuracy simulations are listed.

$$2 * \texttt{lmin} < \texttt{lmax} \tag{1}$$

$$0 \leq 4 * \texttt{areamin} < 4 * \frac{1}{2} * \texttt{lmin}^2 < \texttt{areamax} \leq \frac{\sqrt{3}}{2} * \frac{1}{2} * \texttt{lmax}^2 \tag{2}$$

$$2 * \texttt{a} \leq \texttt{rann} < \texttt{lmin} \tag{3}$$

$$\texttt{rntol} \sim \texttt{rmax} < \texttt{rann} \tag{4}$$

$$\texttt{areamin} = 2 * \texttt{rntol} * \texttt{lmax} \tag{5}$$

2.3 Parameter units

Material parameters must be defined with consistent units to avoid a large disparity in scales. In the following table, simulation units in SI are listed for different physical dimensions. Let α and μ respectively be the lattice parameter and the shear modulus of the material in SI units. Let $B = 1\,[\mathrm{Pa} \cdot \mathrm{s}]$ represent the scaling of drag units.

Dimension	**Simulation units**
Length [m]	α
Pressure [Pa]	μ
Drag [Pa · s]	B
Time [s]	B/μ
Velocity $[\mathrm{m} \cdot \mathrm{s}^{-1}]$	$\alpha \cdot \mu/B$
Force $[\mathrm{Pa} \cdot \mathrm{m}^2]$	$\mu \cdot \alpha^2$
Nodal force [Pa · m]	$\mu \cdot \alpha$

Note that `amag` and `mumag` are respectively the lattice parameter in units of μm and the shear modulus in units of MPa. As an example, consider Zirconium; its lattice parameter and shear modulus are respectively $\alpha = 3.233 \times 10^{-10}\,[\mathrm{m}]$ and $\mu = 3.71 \times 10^{10}\,[\mathrm{Pa}]$. In the following table, assignment values are converted to the corresponding physical values in SI units.

Example: Zirconium

Assignment value	**Physical value**
$\texttt{MU} = 1$	$\texttt{MU} \cdot \mu = 3.71 \times 10^{10}\,[\mathrm{Pa}]$
$\texttt{Bcoeff} = 180\mathrm{E}{-6}$	$\texttt{Bcoeff} \cdot B = 180 \times 10^{-6}\,[\mathrm{Pa} \cdot \mathrm{s}]$
$\texttt{dt} = 37.1$	$\texttt{dt} \cdot B/\mu = 1 \times 10^{-9}\,[\mathrm{s}]$
$\texttt{a} = 90$	$90 \cdot \alpha = 2.91 \times 10^{-8}\,[\mathrm{m}]$
$\texttt{f\_dot} = 0.1$	$\texttt{f\_dot} \cdot \mu^2 \cdot \alpha^2/B = 14.387\,[\mathrm{Pa} \cdot \mathrm{m}^2 \cdot \mathrm{s}^{-1}]$
$\texttt{amag} = 3.233\mathrm{E}{-4}$	$3.233 \times 10^{-4}\,[\mu\mathrm{m}]$
$\texttt{mumag} = 3.71\mathrm{E}4$	$3.71 \times 10^{4}\,[\mathrm{MPa}]$

Suppose that we wish to set the length of dimension x of the cuboid geometry to $10\,[\mu\mathrm{m}]$. This can be done by performing the assignment `dx = 10/amag`. Expressing the physical units of `dx` explicitly yields

$$\texttt{dx} = \frac{10}{\texttt{amag}} \cdot \alpha = \frac{10}{3.233 \times 10^{-4}} \cdot 3.233 \times 10^{-10}\,[\mathrm{m}] = 10 \cdot 10^{-6}\,[\mathrm{m}] = 10\,[\mu\mathrm{m}]$$