

# DDLAB Primer

A. Arsenlis, Lawrence Livermore National Laboratory (arsenlis1@llnl.gov)  
W. Cai, Stanford University (caiwei@stanford.edu)

## Introduction

The purpose of this document is to give a brief description of the DDLAB software for end users who are interested in writing input files and running simulations using the software. This document does not provide an exhaustive description of the algorithms in the code, but rather gives a description of necessary elements of the input deck for the code and the execution procedures. If advanced users are interested in the algorithms of the code, they may simply look at the source code. We have tried to write and organize the subroutines in a compact yet readable manner.

DDLAB consists of a series of MATLAB functions that can be used to perform small dislocation dynamics simulations. The functions are driven by a main MATLAB script named **dd3d.m**. The **dd3d.m** script calculates the forces on the dislocation line segments, updates the positions of nodes, performs topological changes (including collisions between lines and splitting of highly connected dislocation nodes), and remeshes the system as needed. It conducts these operations for as many cycles as specified by the user in the input deck. The code is designed for small dislocation geometries with relatively few dislocation segments because it scales with the square of the total number of segments, or links, in geometry. It was originally written as a development and debug tool for the ParaDiS code developed at LLNL. We do not guarantee the results obtained by DDLAB. Use it at your own risk. For your information, ParaDiS (Parallel Dislocation Simulator) is a massively parallel code written in C. To obtain a copy of ParaDiS please contact Vasily Bulatov (bulatov1@llnl.gov).

## Execution Procedure

To execute the software you must first install MATLAB or Octave. MATLAB is a commercial software package sold by MathWorks and may be obtained from [www.mathworks.com](http://www.mathworks.com). If you don't have access to MATLAB, you may try Octave ([www.octave.com](http://www.octave.com)), which is shareware that has some of the same core capabilities as MATLAB. DDLAB has not been fully tested with Octave.

To execute the code from an input script named **input.m** type the following lines on the MATLAB Command Line:

```
>>input  
>>dd3d
```

To execute the code from a saved dataset named **saveddata.mat** type the following lines on the MATLAB Command Line:

```
>>load saveddata
>>dd3d
```

The dd3d.m script automatically saves a restart file at the end of every cycle. The restart file is overwritten every cycle. The name of the restart file that is saved is **restart.mat**. To load a restart file and execute the code follow the execution procedure given for the saved dataset.

The user may interrupt the code execution with a by pressing Ctrl-C while the MATLAB Command Line window is active. To continue the execution of the simulation after a Ctrl-C termination, it is recommended that the user reload the restart file followed by the dd3d execution statement.

## DDLAB input deck

Please look at the sample DDLAB input deck below:

### inputgeomfrinit.m

```
rn      = [-500 -500 1000 7;
           500 500 -1000 7;
           0 0 0 0];

links = [1 3 0.5 0.5 0.5 -1 1 0;
         3 2 0.5 0.5 0.5 -1 1 0];

MU = 1;
NU = 0.305;
maxconnections=8;
lmax = 2000;
lmin = 200;
areamin=lmin*lmin*sin(60/180*pi)*0.5;
areamax=20*areamin;
a=lmin/sqrt(3)*0.5;
Ec = MU/(4*pi)*log(a/0.1);
totalsteps=200;
dt0=1e7;
mobility='mobbcc1';
integrator='int_eulerbackward';
rann = 10.0;
rntol = 2*rann;
doremesh=1;
docollision=1;
doseparation=1;
plotfreq=1;
plim=10000;
appliedstress =1e-3.*[2 0 1; 0 2 -1; 1 -1 0];
viewangle=[45 45 ];
printfreq=1;
printnode=3;
rmax=100;
```

The input deck above contains the basic information that must be given before the **dd3d** execution can be run in the MATLAB Command Line window. The geometry of the system is given in two data structures: `rn` and `links`. This example can be run by:

```
>>inputgeomfrinit
>>dd3d
```

The `rn` data structure gives the positions of the physical and discretization nodes in the system and any flags associated with those positions. The size of `rn` is four columns wide and the number of nodes long. The first three columns, contains the x,y,z coordinates of the node, and the fourth column contains a flag. Currently there are only two node flags used in the code. A flag equal to zero means that the node is regular node, a flag equal to 7 means that the node is immobile (fixed).

The `links` data structure gives the information of the dislocation segments that connect the nodes. The `links` data structure is eight columns wide and the total number of links long. The first two columns give the node-ids of the starting and ending nodes of the dislocation segment. The 3<sup>rd</sup>-5<sup>th</sup> columns of `links` give the Burgers vector of the dislocation line in Cartesian coordinates, and the 6<sup>th</sup>-8<sup>th</sup> columns of `links` gives glide plane of the dislocation segment.

For example, if we want to find out the vector that connects the starting node to the end node of segment  $i$  in the `links` array, it can be calculated by,

```
vec = rn(links(i,2),1:3) - rn(links(i,1),1:3);
```

The line direction (unit vector) of segment  $i$  is then,

```
unitvec = vec / norm(vec);
```

With `rn` and `links` defined, the geometry of the problem is completely defined. When initializing the geometry with these definitions please make sure that Burgers vector are conserved at all of the nodes in the system. The rest of the lines in the input deck set the conditions for the simulation. Here is a short description of the rest of the input variables:

MU	shear modulus
NU	Poisson's ratio
maxconnections	maximum number of segments a node may have
lmax	maximum length of a dislocation segment (for remesh)
lmin	minimum length of a dislocation segment (for remesh)
areamin	minimum area criterion (for remesh)
areamax	maximum area criterion (for remesh)
a	dislocation core radius used for non-singular force calculation
Ec	dislocation core energy per unit length and burgers vector squared for a screw dislocation (should always be a function of a)
totalsteps	number of cycles that are run for completion of dd3d command
dt0	largest timestep that can be taken during a cycle
mobility	name of the function used to calculate the velocity of the nodes
integrator	name of the time integration scheme used to update nodal positions

rann	annihilation distance used to calculate the collision of dislocation lines
rntol	solution tolerance used to control the automatic timestepping
doremesh	a flag set either to 0 or 1 that turns the remesh functions off or on respectively
docollision	a flag set either to 0 or 1 that turns the collision detection off or on respectively
doseparation	a flag set either to 0 or 1 that turns the splitting algorithms for highlyconnected nodes off or on respectively
plotfreq	number of cycles between plots of geometry
plim	limits of plotting space
appliedstress	external stress given in a three by three symmetric tensor
viewangle	angle of viewpoint for the 3D plot of the geometry
printfreq	number of cycles between monitored node write statements
printnode	nodeid of the monitored node
rmax	maximum distance a node may travel in one cycle

Along with **inputgeomfrinit.m** which shows the operation of a Frank-Read source, we have also included some other simple demonstration geometries:

**inputgeombinaryjunction.m** shows zipping of a binary junction under no applied stress

**inputgeommultijunction.m** shows zipping of a ternary junction under no applied stress

Along with the two data structures `rn` and `links` there are other data structures that the code creates that may be useful to the end user:

`fn(nodeid, :)` force as a row vector on node number `nodeid` in Cartesian coordinates

`fseg(linkid, :)` force as a row vector on link number `linkid` in Cartesian coordinates. The first three entries correspond to the force on the start node of the link and the last three entries correspond to the force on the end node due to the segment

`vn(nodeid, :)` velocity as a row vector on node number `nodeid` in Cartesian coordinates

`connectivity(nodeid, :)` connectivity information of node number `nodeid`. The `connectivity` array can be considered as the “inverse” of the `links` array. It specifies the position of a node in the `links` array. The first column (`nlinks`) specifies the total number of links that node `nodeid` has. It is then

followed by  $(2 * nlinks)$  columns, specifying the `linkid` and a flag (1 or 2) for the links that `nodeid` is connected to. The flag equals 1 if the node is the start node of the link and equals 2 if the node is the end node of the link.

A series of mobility functions are included in the distribution named `mobbcc0`, `mobbcc0b` and `mobfcc1`. `mobbcc0` and `mobbcc0b` are intended to simulate bcc behavior in which the screw dislocations are able to glide in any plane normal to their Burgers vector. The difference between them is slight and appears in the mobility of dislocations of mixed character. `mobfcc1` is intended to simulate fcc behavior in which the screw dislocation are not able to cross-slip. The mobility function is chosen in the input deck with the `mobility` input parameter.

A series of time integration schemes are also included with the distribution named `int_eulerforward`, `int_eulerbackward`, and `int_newtonkrylov`. `int_eulerforward` is an explicit forward integration scheme. `int_eulerbackward` is a simple implicit time integration scheme, and `int_newtonkrylov` is a more sophisticated inexact implicit time integration scheme based on the Newton-Krylov matrix-free solution method. The core of the `int_newtonkrylov` employs software written by C. T. Kelley. The time integration scheme is specified in the input deck with the `integrator` input parameter.

Included in the software distribution is a geometry checker that will not allow a simulation to begin unless it is given a valid geometry and generates error messages for bad initial geometries guiding the user as to what need to be corrected.

Enjoy simulating dislocation dynamics with DDLAB. Feel free to send comments to the email addresses listed above.

A. Arsenlis and W. Cai