

Getting started with the SOLD program

Y. Mishin

George Mason University, Fairfax, VA

ymishin@gmu.edu

January 24, 2003

1 Introduction

SOLD (**S**imulator **O**f **L**attice **D**efects) is a program designed for atomistic simulations of crystals and crystalline defects in metallic materials using the Embedded Atom Method (EAM) [1]. The program implements three atomistic simulation methods: static relaxation (molecular statics), molecular dynamics (MD), and Monte Carlo (MC).

The SOLD program implements the following basic steps:

1. Generates a crystalline lattice of an elemental solid or an ordered compound
2. Performs a desired rotation of the lattice relative to the laboratory coordinate system
3. Creates a rectangular simulation block containing lattice atoms. The edges of the block are parallel to the coordinate axes of the laboratory system
4. Establishes desired boundary conditions of the simulation block in each direction
5. Introduces any number and combination of lattice defects into the simulation block, including vacancies, interstitials, antisite defects, stacking faults, grain boundaries, and surfaces
6. Applies an orthorhombic deformation of the simulation block if necessary
7. Minimizes the total energy of the simulation block using a choice of two minimization methods and several minimization modes
8. Runs molecular dynamics (MD) at a desired temperature
9. Runs a Metropolis MC simulation at a given temperature using a choice of several thermodynamic ensembles

The program generates a detailed output file and saves the final configuration of the simulation block in a special format that includes all atomic coordinates and velocities as well as other essential information, such as the block dimensions, boundary conditions, etc. This configuration file can be downloaded into the program in a new run if the user wishes to add more lattice defects, perform an additional relaxation of the block, or continue an MD or MC simulation. The program also saves snapshots of the simulated system, a history file, and a few other files as explained below.

The program has been written in Fortran 77 and parallelized using the Message Passing Interface (MPI). It is highly portable and should compile and run under virtually any UNIX or LINUX environment containing a Fortran 77 compiler and MPI libraries. This manual explains how to install the program and run it in different modes. With some general knowledge of computer simulation methods and good luck, you should be up and running within minutes.

The program uses EAM potential files stored in the so-called "Voter's format". The present distribution comes with two sets of potential files: for copper and B2-NiAl. The respective EAM potential have been constructed in Refs. [2] and [3].

2 Installing, compiling and running the program

The program comes as an archive `sold.tar`. Copy it into the directory in which you want to install the program, change to that directory, and unpack the archive:

```
tar -xvf sold.tar
```

This command will create in the current directory several input files of the program, a copy of this manual, and subdirectories `source/`, `pot/`, `example1/`, `example2/`, and `example3/`, and `snap/`. The first subdirectory contains all source files while the second one contains EAM potential files. The main Fortran source file is `source/sold.f`. The example directories contain example applications of this program.

Without changing the directory, compile the program using your Fortran 77 compiler and linking the program to MPI libraries. For example, the command

```
f77 -o sold source/sold.f -lmpi
```

issued on SGI Origin 2000 will create an executable `sold` in your current directory. You can modify this command by including the compiler optimization and other options as necessary.

Before running the program, choose the main input file that you want to use. This distribution comes with two main input files: `sold-cu.dat` for Cu and `sold-nial.dat` for NiAl. Run the program with one of them as input by typing, for example,

```
mpirun - np4sold < sold - cu.dat&
```

or

```
mpirun - np4sold < sold - nial.dat&
```

The program will run for a few seconds to a minute and terminate, creating an output file (`cu.out` or `nial.out`) and a final configuration file (`cu.plt` or `nial.plt`). Make sure

that the contents of the output file is identical to the contents of the corresponding “must be” file (`cu.out.mustbe` or `nial.out.mustbe`, respectively) contained in the same directory. This will tell you that the program setup has been successful and you can start running your own simulations. For that you will need to modify the input files according to your simulation task.

3 Program parameters

The file `source/parameter.f` contains global parameters of the program that determine memory resources required by the program. Such parameters include the maximum number of atoms in the simulation block (`maxatom`), maximum number of chemical species (`nspmax`), maximum number of atoms in the basis of the unit cell of the lattice (`nbasmx`), maximum number of neighbors with which an atom can interact (`maxnb`), etc. The total memory required by the program depends primarily on the parameters `maxatom` and `maxnb`. This is because the program constructs and keeps in the memory a table of neighbors of every atom. This table has dimensions $3 \times \text{maxatom} \times \text{maxnb}$ and strongly dominates the overall amount of memory required by the program. The parameters set up in the current version of the parameter file present a suitable set for typical atomistic simulations and should not cause any memory problems on a regular workstation or Linux PC:

```
parameter (  
& maxatom = 20000, ! max number of atoms in block  
& nbasmx = 16,    ! max number of basis atoms  
& nspmax = 4,     ! max number of chem. species  
& maxnb = 200,   ! max number of neighbors  
& maxint = 100,  ! max number of interstitials  
& maxpnt = 5000, ! max number of tabulated values for potls  
& maxmcstg = 10  ! max number of MC stages  
& )
```

You may have to adjust these parameters to your specific simulation problem and/or computational resources.

4 Main input file

The main input file of the program navigates you through the basic steps of the simulation (Table 4). It can point to secondary input files that control individual steps of the simulation. This Section describes the structure of the main input file. Have an example of a main input file (e.g., `sold-cu.dat` or `sold-nial.dat`) before you as you read this Section. Note that all input data are read by the program with the formatless read instruction `READ (unit,*)`, so that you need not worry about spaces, format etc. If a secondary input file is stored in a different directory, the path to it should also be included with the file name, for example `'pot/pcu.dat'`. This equally applies to output files. In most cases, relative paths are sufficient but some environments may require absolute paths.

Step	Action	Secondary input file	Output file
1	Interpolate EAM potential files	potential files, *.dat	
2	Generate the simulation block	block generation file, *.gen	
3	Introduce lattice defects	defect file, *.def	
4	Deform the simulation block		
5	Minimize the block energy	minimization file, *.min	
6	Run MD	MD file, *.md	report file, snapshots
7	Run MC	MC file, *.mc	history file, occupation file, snapshots
8	Save the final configuration		configuration file, *.plt

The first line of the main input file specifies the name of the main output file that will be created by the program. The second line gives the number of chemical elements involved in the simulation. This line is followed by lines containing the chemical symbol of each element (e.g., 'Cu' or 'Ni') and its atomic mass in a.m.u. The elements are automatically numbered by the program in the order they are listed in this file.

The following several lines point to the EAM potential files that will be used by the program. The pair interaction functions are listed first, followed by electron density functions and then embedding functions. The electron density and embedding functions appear in the order in which the elements were introduced into the program. The pair interaction functions are listed in an "upper diagonal" manner. For example, for a ternary system with elements A, B and C the order of pair interaction files must be A-A, A-B, A-C, B-C, C-C. After reading the potential files, the program develops a cubic spline interpolation through the tabulated values. The spline coefficients are stored in arrays and are used for calculating the potential functions in the course of the simulation.

In the next line, choose between generating a new simulation block (type any non-zero value) and re-using a configuration saved by a previous run (type zero). In the following line, type the name of the block generation file (e.g., 'fcc-cu.gen' or 'nial.gen') or the configuration file (e.g., 'cu.plt' or 'nial.plt'). The structure of both files will be explained later.

The next line contains the name of the secondary input file describing lattice defects introduced into the simulation block. The structure of the defect file will also be explained later (Section 7). If you type 'none' for the defect file name, no defects will be introduced and the program will proceed to the next step.

The four numbers given in the next line determine the deformation applied to the simulation block. The first three numbers (let us call them s_x , s_y and s_z) give the scaling factors in the x , y , and z directions parallel to the block edges. Using these factors, the block sizes and Cartesian coordinates of all atoms are subject to a linear transformation $(x, y, z) \rightarrow (s_x x, s_y y, s_z z)$. Furthermore, all atoms with coordinates $y > 0$ and $y \leq 0$ are translated by t (respectively, $-t$) Angstroms, where t is the fourth number in the deformation line. This translation will obviously create a gap of width $2t$ between the two halves of the block. This transformation is useful in simulating surfaces and certain types of planar fault. If you want to skip the deformation step,

the deformation line should be

```
1.0 1.0 1.0 0.0 - initial deformation factors
```

The following two lines specify the energy minimization mode and give the name of the minimization file, i.e., file containing minimization parameters. The minimization procedure will be discussed in Section 8. The next line gives the name of the MD file containing parameters of the MD simulation ('none' if no MD simulation is needed). This is followed by a line specifying the name of the MC file ('none' if no MC simulation is needed). The last line specifies the name of the file in which the final configuration will be saved in a special format. If the file name is 'none', the final configuration will not be saved.

5 Generating the simulation block

All information required for constructing a new simulation block is read from the block generation file, for which we conventionally use the extension `.gen`. Examples are given by files `fcc-cu.gen` and `nial.gen`. The block generation file has the following structure (have an example file before you for comparison).

The first three lines contain Cartesian coordinates of three translation vectors of the lattice (\mathbf{a}_1 , \mathbf{a}_2 , \mathbf{a}_3) in the laboratory coordinate system. The coordinates must be in Angstroms and the vectors do not have to be orthogonal.

The fourth line contains the number of basis atoms (sublattices) in the structure and is followed by lines containing coordinates and chemical sorts of the basis atoms. The first three numbers in each line are atomic coordinates in Angstroms and the fourth number specifies the atom's chemical sort (remember that all chemical sorts are numbered in the main input file).

The following three lines contain coordinates of three mutually perpendicular vectors (\mathbf{n}_1 , \mathbf{n}_2 , \mathbf{n}_3) which define crystallographic directions parallel to the edges of the intended simulation block. The units of the coordinates do not matter because the program normalizes these vectors immediately after reading them. It also checks that the vectors are orthogonal, and if they are not, the program issues an error message and stops. If they are orthogonal, the program proceeds to determine the periodicity of the Bravais lattice (i.e., lattice generated by vectors \mathbf{a}_1 , \mathbf{a}_2 , and \mathbf{a}_3) along the directions \mathbf{n}_1 , \mathbf{n}_2 , and \mathbf{n}_3 . The periodicity is characterized by the number of planes of the Bravais lattice before it repeats itself in each direction. Let the respective numbers be k_1 , k_2 and k_3 . The program then rotates the lattice to make the crystallographic directions \mathbf{n}_1 , \mathbf{n}_2 and \mathbf{n}_3 parallel to the coordinate axes x , y and z of the laboratory system.

The next line of the file specifies the sizes of the rectangular simulation block in the x , y and z directions as the numbers of lattice planes parallel to those directions. For example, the line

```
-6 20 -8 - inner block dimensions
```

means that the block contains 6 planes of the Bravais lattice in the x direction, 20 planes in the y direction, and 8 planes in the z direction. This line also specifies boundary conditions in the respective directions, with a negative sign corresponding to a periodic boundary condition and a positive sign to a fixed boundary condition. Periodic boundary conditions are well known in

atomistic simulations and will not be discussed here. The fixed boundary condition is less common and will be discussed in Section 6. In the example above, the block has periodic boundary conditions in the x and z directions and a fixed boundary condition in the y direction. Importantly, the block size in any periodic direction must be a multiple of the respective periodicity k_i , otherwise a stacking fault will be created between the simulation block and its periodic image. Ideally, you are expected to know the periodicities k_1 , k_2 , and k_3 in advance, but in practice you can use the following trick:

1. Type an arbitrary negative number for each periodic direction.
2. Run the code. The program will calculate the actual periodicities k_i in all directions and report them in the output file. If for any periodic direction your chosen block size is not a multiple of the respective k_i , the program will stop.
3. Look up the actual periodicities in the output file. The message will look like this

```
NUMBER OF PLANES BEFORE LATTICE REPEATS IN EACH DIRECTION
      3      5      2
```

where the second line gives you the k_i values.

4. Knowing them, modify the block generation file and restart the run.

Once the block sizes and boundary conditions are established, the program places atoms on lattice positions inside the simulation block as well as in the buffer and fixed regions in the case of fixed boundary conditions (See Section 6). This completes the construction of the simulation block.

The last line of the block generation file contains the search radius, R_s , for constructing the table of neighbors. In order to accelerate the energy and force calculations, the program constructs a table of neighbors of each atom and keeps this table in the memory during the simulation. Neighbors are identified as atoms contained within a sphere of radius R_s around an atom. When calculating the energy and forces, the program only considers interactions between tabulated neighbors.

There is no recipe for finding the optimum value of R_s . For each particular type of simulations, the optimum value of R_s making a compromise between memory, speed, and reliability can be found by experimenting. Obviously, R_s must be larger than the cutoff radius R_c of atomic interactions, otherwise some interactions will be missed. The difference $R_s - R_c$ gives an estimate of the maximum atomic displacement such that the calculated energy and forces will still be correct. If the displacement of at least one atom exceeds $R_s - R_c$, there is a danger that some atoms that are not listed as neighbors will be separated by a distance smaller than R_c , which may affect the calculated energy and forces. This issue is handled differently in different simulation methods. In the energy minimization modes 5 and 50 (see Section 8), the program calculates the maximum atomic displacement R_m after at each iteration and updates the table of neighbors if R_m exceeds $R_s - R_c$. In all other minimization modes the table of neighbors is not updated. However, the program reports, in the output file, the maximum atomic displacement R_m due to the entire relaxation process. If the reported R_m is small enough, you can afford

decreasing R_s in subsequent runs with the benefit of faster calculations and a smaller memory usage. In other cases you may need to increase R_s , which will make the program slower and the memory larger. The maximum possible number of neighbors of an atom is controlled by the parameter `maxnb` in the file `source/parameter.f`. In the rare event that the relaxation is so extensive that R_s goes beyond reasonable limits, make the program stop after a small number of iterations (see Section 8), then start a new run using the previously saved configuration, and so on. That way you can make sure that each run has a small enough value of R_m . In the MD and MC modes, the program updates the neighbor table automatically as discussed below.

6 Boundary conditions

The program offers a choice between a periodic and fixed boundary condition in each individual direction x , y and z . If a fixed boundary condition is chosen in a given direction (say, y), the lattice continues beyond the simulation block in the positive and negative directions parallel to the y axis to create two lattice slabs on either side of the block (Fig. 1). Each slab has the thickness of twice the search radius R_s and is divided into a buffer region and a fixed region, each having the thickness of R_s . The part of the simulation block sandwiched between the buffer/fixed slabs is called the inner block. Atoms of the inner block are referred to as free atoms because they are allowed to move each individually during the energy minimization, MD or MC simulations. The buffer and fixed atoms are “frozen” in their perfect lattice positions relative to one another, but the slabs can experience rigid-body translations in some minimization and MC modes. By construction, free atoms are able to interact with buffer atoms but not with fixed atoms. However, the interaction between free and buffer atoms depends on the electron densities on buffer atoms, which cannot be calculated properly without including fixed atoms into the calculation. This reflects the well known fact that force calculations in the EAM model involve atoms separated by twice the cutoff radius of the potential functions. The energy calculated and minimized by the program is the total energy of free and buffer atoms together. In the case of periodic boundary conditions in all three directions (“supercell” geometry), this energy obviously coincides with the total energy of all atoms in the block.

The choice of boundary conditions depends on the problem at hand. For a perfect crystal or a single point defect, the supercell geometry is the best choice. For a grain boundary, surface, or any other planar defect, it is best to combine periodic boundary conditions in the x and z directions parallel to defect plane with the fixed boundary condition in the y direction normal to the defect plane [Fig. 1(left)]. In this case the buffer and fixed atoms simulate lattice regions away from the defect. In some cases, fixed boundary conditions are applied in two or all three directions [Fig. 1(right)].

7 Lattice defects

The program offers a flexible mechanism of introducing lattice defects into the simulation block. The relevant part of the program is controlled by a defect file, see for example `cu.def` and `ni.al.def`. Open an example defect file as you read this Section.

Figure 1: Geometry of the simulation block created by the SOLD program. Left: block with a fixed boundary condition in the y direction and periodic boundary conditions in the x and z directions. Right: block with fixed boundary conditions in the x and y directions. The z axis is normal to the paper.

Any non-zero value in the first line signals that you want to create a symmetrical tilt grain boundary. The boundary is created by a 180° rotation of one half-crystal ($y > 0$) relative to the other ($y \leq 0$). This transformation is typically applied in combination with fixed boundary conditions in the y direction [Fig. 1(left)], but can also be applied to supercells. It is preceded by a lattice rotation so that to align the desired crystallographic plane parallel to the x - z plane and the appropriate crystallographic direction parallel to the tilt axis z . A zero number in the first line suppresses the grain boundary generation.

In the next line, indicate whether you want to create a planar fault. Zero means no, a nonzero number means yes. In the latter case, open a new line and type Cartesian coordinates of the translation vector (in Angstroms) in any format. The coordinates must be separated by at least one space. The program will translate one half-crystal ($y > 0$) relative to the other ($y \leq 0$) by this vector, which will result in a planar fault on the $y = 0$ plane. Again, this construction is typically applied in combination with a fixed boundary condition in the y direction [Fig. 1(left)] or with the supercell geometry. It is preceded by a lattice rotation aligning the appropriate crystallographic plane parallel to the x - z plane. For example, a stacking fault can be created by applying a translation parallel to the x - z plane. Two free surfaces can be created by applying a translation parallel to the y direction by a distance larger than twice the cutoff radius of atomic interaction. The planar fault generation is often combined with the creation of a grain boundary at the previous step. This gives you an opportunity to model rigid-body translations parallel to the GB.

After the creation of a grain boundary and/or a planar fault, the program restores the initial

rectangular shape of the simulation block if it was affected. To this end, the program sends atoms that left the simulation block back to the block by applying appropriate translations parallel to periodic directions (i.e., by exchanging atoms with their periodic images).

The next line creates vacancies. You can type zero if you do not want any vacancies, or a non-zero number of vacancies that you want to create. In the latter case, open a new line per every vacancy and type the number of the atom that you want to remove. For example, the following lines create three vacancies by removing atoms 8, 25 and 47:

```
3 - if not zero, open new lines with sites where to place vac's
8
25
47
```

If you know only the location where you want to put a vacancy but do not know the atom number, skip the vacancy part in the defect file, skip the minimization process (minimization mode 0), run the program, and save the final configuration in a file. Then open the configuration file, identify the desired atom by its coordinates, modify the defect file accordingly, and repeat the run. It is useful to know that the program does not actually remove an atom when creating a vacancy. It simply assigns that atom a factitious chemical sort "0". When calculating the energy and forces, atoms of this sort are simply ignored, which creates the effect of a missing atom.

The next line introduces antisite defects if your crystal is an alloy or a compound. Again, you can type zero to skip this step or a non-zero number of antisites that you want to create. In the latter case, open a new line per every antisite and type the number of the atom whose chemical sort you want to change followed by its new chemical sort. Obviously, if the new and old chemical sorts coincide, no defect is introduced. For example, the following lines assign chemical sort 1 to atom 9 and chemical sort 2 to atom 35:

```
2 - if not zero, open lines w. sites where to place antisites
9 1
35 2
```

If the initial chemical sorts of atoms 9 and 35 are 2 and 1, respectively, then we introduce two antisite defects, or simply switch the two atoms. This way you can effectively move atoms around the simulation block, or introduce any number of substitutional impurities into the crystal. Again, since we address atoms by their numbers, such numbers must be looked up in the previously saved configuration.

The last line of the defect file gives you an opportunity to introduce interstitial atoms. Type zero if you want to skip this step, or the number of interstitials if you want to create them. In the latter case, open a new line per every interstitial atom and type its Cartesian coordinates (in Angstroms) and its chemical sort. Use any format you like and separate the numbers by at least one space. For example, the following lines introduce two interstitials: an atom of sort 1 at point (1.0, -1.0, 1.0) and an atom of sort 2 at point (-4.0, -5.5, 8.2):

```
2 - if not zero, open new lines with sites where to place int's
1 -1 1 1
-4 -5.5 8.2 2
```

If the block consists of atoms of sort 1, then the first atom is a self-interstitial while the second atom is an interstitial impurity. The maximum number of new atoms that can be introduced this way is controlled by the parameter `maxint` in the parameter file `parameter.f`.

Clearly, by combining the last three steps of the defect file you can create virtually any number and combination of point defects, remove or add atoms to the block, or manipulate their chemical sorts. The block itself may contain a grain boundary, a surface, or a planar fault, which gives you an opportunity to study the interaction of point defects with extended defects, for example surface segregation. It should be pointed out that the table of neighbors is constructed *after* the introduction of all defects.

8 Energy minimization

8.1 Minimization modes

The program offers several minimization modes as specified below.

Mode=0 No energy minimization. The program only calculates and reports the block energy.

Mode=1 Minimization with respect to local displacements of free atoms.

Mode=2 Minimization with respect to local displacements of free atoms plus variation of the block volume by uniform isotropic deformation.

Mode=20 Minimization with respect to volume deformation only. No local atomic displacements.

Mode=3 Minimization with respect to local displacements of free atoms plus variation of the block size in the y direction by uniform deformation.

Mode=30 Minimization with respect to variations of the block size in the y direction by uniform deformation. No local atomic displacements.

Mode=4 Minimization with respect to local displacements of free atoms plus rigid-body translations of the buffer/fixed slabs in the y direction.

Mode=40 Minimization with respect to rigid-body translations of the buffer/fixed slabs in the y direction. No local atomic displacements.

Mode=5 Minimization with respect to local displacements of free atoms plus variations of the block dimensions in all three directions by uniform deformation of the block.

Mode=50 Minimization with respect to variations of the block dimensions in all three directions by uniform deformation of the block. No local atomic displacements.

The choice of the minimization mode obviously depends on the simulation task. Mode 1 is the most common one: in this mode we fix the block dimensions and only allow free atoms to move and minimize the total energy. Modes 2, 3, 5, 20, 30 and 50 are normally used in

combination with the supercell boundary conditions. Modes 2 and 20 are especially useful for cubic crystals, while modes 3 and 30 are suitable for tetragonal and hexagonal crystals. Modes 5 and 50 are used for working with orthorhombic structures or simulating complex point defects and defect clusters. One should be extra careful when applying this mode to mechanically unstable crystal structures. Modes 4 and 40 are designed for simulating grain boundaries, stacking faults, and other planar defects.

8.2 Minimization file

The choice of the minimization mode is made in the main input file, but further details regarding the minimization process are read from the minimization file, for example `cu.min` or `nial.min`. The first line of that file controls the output generated during the minimization. If you type zero, the program will only report the results of the minimization, such as the number of iterations, residual forces, etc. This is the normal regime for productive simulations. If you type any nonzero number, the program will be reporting the energy and maximum force after every iteration. This can be useful if you are testing the program or simply want to trace the minimization process.

The next line offers you to choose between two minimization methods: one is the well-known conjugate gradient method and the other is a special algorithm written in the spirit of damped molecular dynamics (MD). The choice depends on the simulation task. To our experience, the conjugate gradient algorithm is relatively slow but quite stable. The damped MD algorithm is a bit fussy: if the choice of the initial step is unfavorable, it may take forever to converge, but if it is favorable, the algorithm can converge 10 to 100 times faster than the conjugate gradient method. The recommended strategy is to use the conjugate gradient method in exploratory runs, but if you need to make many runs with similar initial configurations, it makes sense to adjust the initial step of the damped MD method and start using it. Note that the damped MD algorithm calculates forces only, whereas the conjugate gradient method also makes a significant number of energy evaluations.

The following line specifies the stopping criterion of minimization, namely, the maximum force (in eV/Å) over all atoms of the block at which the algorithm is deemed to be converged. We normally consider the minimization converged after the maximum residual force falls below 10^{-5} eV/Å.

The next line sets the maximum number of iterations. If the algorithm has not converged after this number of iterations, the minimization stops and the program reports the current value of the residual force.

The following line specifies the parameter s which we call the “initial step of minimization”. In the conjugate gradient algorithm, s is the length (in Angstroms) of the interval of the one-dimensional minimum search in the force direction. In the damped MD, s is the initial and maximum atomic displacement (also in Angstroms) during the minimization. If the (Verlet-type) iteration dictates an atomic displacement longer than s , it is cut down to s . The recommended trial values of s are 0.1 Å for the conjugate gradient method and 10^{-2} – 10^{-3} Å for the damped MD method, but they need to be adjusted to every particular type of simulations.

The last line of the file sets up the accuracy of one-dimensional minimization involved in some minimization modes. For example, in modes 2 and 20 this number gives the relative error

in calculating the equilibrium size of the simulation block in any direction. Recommended values lie between 10^{-4} and 10^{-7} , depending on the problem at hand.

If you type 'none' instead of the minimization file name in the main input file, the program will use the following default values of the minimization parameters:

```
0 - mute. if nonzero, print the energy after every iteration
1 - minimization method (1 - conjug. gradient, 2 - damped MD)
1.D-5 - requested max gradient at minimum
500 - max number of iterations
0.1D0 - initial step of minimization
1.D-5 - accuracy of 1D minimization
```

9 Molecular dynamics

9.1 MD input file

Parameters of MD simulations are set up in the MD file (*.md), for example:

```
1000.0 - temperature of simulations
2.0 - time step in femtoseconds (recommended 2-3)
2 - ensemble: 1 - NVE, 2 - NVT
100. - mass of thermostat (approx. 10-100)
500 - number of steps for equilibration
2500 - number of steps for production
10 - number of steps for averaging
50 - number of steps for updating neighbors
1.8 - displacement for updating neighbors
100 - number of steps to save snapshots
'nial.report' - report file
```

The first line of the file contains the desired temperature of the simulation (in K), the second line the MD time step in femtoseconds (recommended values are 2-3). The integration of the equations of motion is performed by the Verlet method. The third line offers you to choose between the micro-canonical ensemble NVE (choice 1) and the canonical NVT ensemble (choice 2). For the NVT ensemble, the constant temperature is controlled by a Nose-Hoover thermostat, therefore the next line specifies the mass of the thermostat. Reasonable values of this parameter range between 10 and a few hundred. If the thermostat mass is too small, the amplitude of temperature fluctuations can be too large; if the thermostat mass is too large, the characteristic period of temperature fluctuations may be too long for the chosen MD time.

The following two lines contain the number of MD steps for equilibration and production, respectively. The MD process itself is the same at both steps but the average temperature of the MD run is calculated by averaging over the production step only. If this is a new MD run and not a restart of a previous run, the program initially creates a Maxwellian distribution of atomic

velocities corresponding to the desired temperature. For a restarted run, the velocities are read from the configuration file.

The next line specifies the number of MD steps for local averaging. If, for example, you type 10 in this line, the program will average the kinetic energy and temperature after every 10 MD steps and will report these numbers in the report file. The next line contains the number of steps for a possible update of the neighbor table. For example, with 50 in this line the program checks if the table needs to be updated after every 50 MD steps. It is updated if the maximum atomic displacement over our system exceed the critical value given (in Angstroms) in the next line of the file (1.8 in the example above). If this number is set to be zero or negative, the program will use the difference $R_s - R_c$ instead. Each time the neighbor table is updated, the program saves a snapshot in a new file. Each line of the snapshot file contains the atom number, its instantaneous Cartesian coordinates, and its chemical sort. Snapshots are also saved at regular intervals regardless of atomic displacements and without updating neighbors. Such "forced" snapshots are with the periodicity specified in the last but one line of the MD file (after every 100 MD steps in the example above). Each time a snapshot is saved, the program writes a line in the output file, reporting the current number of MD steps and marking "forced" snapshots by an asterisk.

You should have in the working directory a file `snap.names` containing names of snapshot files to be used by the program, with one line per file-name. Here is an example of a `snap.names` file:

```
'snap/snap1.plt'  
'snap/snap2.plt'  
'snap/snap3.plt'  
'snap/snap4.plt'  
'snap/snap5.plt'  
'snap/snap6.plt'  
'snap/snap7.plt'  
'snap/snap8.plt'  
'snap/snap9.plt'  
'snap/snap10.plt'
```

If the program tries to save more snapshots than you have names in the `snap.names` file, all excessive snapshots are saved into a file called `SNAP_DUMP`, over-writing each other. This rather conservative procedure is designed to provide a strong defense against overfilling your hard drive by saving more snapshots files than you expect.

Finally, the last line gives the name of the MD report file, with 'none' if the report is not required. The structure of the report file is explained below.

9.2 MD report file

Each line of the MD report file contains the number of MD steps made, the local-averaged temperature and kinetic energy, the instantaneous potential energy, and the total energy (local-averaged kinetic energy plus instantaneous potential energy). In the case of an NVT run, the

line contains another number, namely, the Hamiltonian of the extended system, which should be practically constant if the MD parameters are set up correctly. Remember that the number of steps for calculating local averages is set up the MD file.

10 Monte Carlo simulation

10.1 General information

The program performs a Metropolis Monte Carlo simulation at a desired temperature. Trial MC moves include (but are not limited to) choosing a free atom at random and displacing it by a random amount in a random direction, with a possible change in its chemical sort. We say that the program has made *one MC step* after it has made N trial moves of individual atoms (N being the number of free atoms in the block) plus has made a trial change in the block volume/shape if this is prescribed by the simulation mode. Thus, at each MC step the program implements the following trial moves:

1. Displace a randomly picked free atom by a random amount in a random direction
2. Simultaneously with displacing the atom, switch its chemical sort by choosing the new sort at random from all sorts present in the system.
3. After N individual atomic displacements/switches, change the shape and/or volume of the simulation block.

Moves 1 are implemented in any MC simulation mode, while moves 2 and 3 are optional. Each move is accepted or rejected according to the Metropolis criterion [4].

If the chemical sorts of atoms are fixed, the program simulates a canonical ensemble. If they are subject to change, it simulates a grand-canonical ensemble. More accurately, it then simulates a constrained grand-canonical ensemble, called also a permutational ensemble, in which the total number of atoms in the simulation block is fixed while their chemical sorts can vary. In the latter case, it is the differences between the chemical potentials of the elements that are held fixed during the simulation, and not each chemical potential individually. (The latter would be the case in a general grand-canonic ensemble involving the removal or addition of atoms to the system [4].) In terms of applications, this method is only good for simulating substitutional alloys as well as compounds disordered by the antisite mechanism.

The program divides the MC simulation into several stages and allows the user to change the ensemble from stage to stage, while keeping temperature fixed. The first stage (or stages) usually serve to bring the system to thermodynamic equilibrium. The last but one stage is devoted to production, i.e., calculation of average quantities such as the average volume of the block, its average chemical composition, etc. The last stage is intended for evaluating mean squared deviations of thermodynamic quantities from their average values determined at the previous stage. The maximum number of MC stages is specified in the `parameter.f` file.

10.2 MC input file

Parameters of the MC simulation are set in the MC file (*.mc), for example:

```
1200. - temperature
1 - if nonzero, switch atomic sorts using chem. potentials below
0
1.1
3 - number of stages of the simulations (>1); describe them below
1000 0.1 2 1
1000 0.1 2 1
300 0.1 2 1
20 - number of steps to update neighbors (if necessary)
200 - number of steps to calibrate the total energy
1 - if nonzero, create a history file below
'history.nial'
500 - number of steps to save a snapshot
1 - if nonzero, create an occupational file below file below
'occup'
```

The first line specifies the temperature of the simulation (in K). Zero in the second line indicates that chemical sorts of atoms will not be switched at any stage of the simulation, a non-zero value indicates that they may be switched. In the latter case, open as many new lines as you have chemical species in the system, and in each line introduce the chemical potential of the species (in eV) following in order. Because only chemical potential differences are important for the simulation, one arbitrary chemical potential can be set to zero. Even though simulations of any multicomponent systems are possible with this code, it has been extensively tested for binary systems only.

The next line contains the number of stages of the simulation and is followed by lines each describing an individual stage. In each of those lines, the first number is the number of MC steps (see our definition of a MC step above), the amplitude of atomic displacements in Angstroms (0.1 Å in this example), the mode of shape/volume fluctuations of the block, and the signal of whether the chemical sorts of atoms should be changed at this stage (non-zero if yes, 0 is not). The magnitude of a trial atomic displacement is determined by multiplying the amplitude by a random number with a uniform distribution over the range [0,1]. The modes of shape/volume fluctuations are discussed below.

The line that follows specifies the frequency with which the program checks if the table of neighbors needs to be updated. It compares the current atomic coordinates with those stored after the previous check and calculates the maximum atomic displacement. If the latter exceeds $R_s - R_c$, the neighbor table is updated and the current configuration is stored for the next check. In the example above, such checks are made every 20 MC steps.

When dealing with individual atomic moves, the program calculates directly the energy change, δE , associated with each move, which is much faster than calculating the total energies before and after the move. The total energy of the block is then calculated by adding the δE value of a successful (accepted) move to the current energy. This can, in principle, result in some

accumulation of error in the total energy. To avoid this, after a certain number of atomic moves the program calculates the total energy directly (energy calibration). This number is specified in the next line of the input file (200 in our example). Experience shows that the optimum number in this line should be a few hundred to a thousand. More often energy calibrations only reduce the program speed without any benefit to the accuracy.

A non-zero value in the next line indicates that you want the program to create a history file. In that case, open a new line and type the name of the history file. The structure of the MC history file will be discussed later.

The number in the next line is the number of MC steps between snapshots. In our example, snapshots are saved after every 500 MC steps. The mechanism of saving snapshots and the structure of snapshot files are the same as in MD simulations (see Section 9).

The last line of the file gives you an option to create a so-called occupation file. Type any non-zero number and open a new line with the occupation file name, or type zero to suppress the creation of an occupation file. The occupation file contains the chemical composition of every atom averaged over the production stage of the simulation. Each line of the file contains the atom number followed by the probability of its occupation by every species present in the system. Of course, only free atoms are listed in the file. Working with this file only makes sense in a grand-canonical simulation. The occupation file is often used for studying grain boundary or surface segregation and other effects in non-uniform systems.

10.3 Shape and volume fluctuations

The program supports the following modes of shape/volume fluctuation in the MC regime:

Mode=1 The block dimensions are fixed

Mode=2 Volume fluctuations without changing the block shape

Mode=3 Fluctuations of the block size in the y direction only

Mode=4 Fluctuations of the block dimensions independently in the x and z directions

Mode=5 Fluctuations of the block dimensions independently in the x , y and z directions

Obviously, the last mode is the most general one and can be compared with mode 5 of the energy minimization (Section 8.1).

Given that each shape/volume fluctuation mode can be combined with either a canonical or grand-canonical simulation, the program is able to simulate the total of ten different thermodynamic ensembles. For example, by switching chemical sorts of atoms in Mode=2 we simulate the $\mu p T$ ensemble, i.e., grand-canonical ensemble under zero pressure. The easiness with which one can create and change thermodynamic ensembles in the course of one simulation is a well-known advantage of the MC method over MD [4].

10.4 MC history file

The head lines of the MC history file repeat the number of MC steps, the amplitude of atomic displacements, and the thermodynamic ensemble chosen for each stage of the MC simulation. The head lines start with symbol # to make them comment lines when plotting the history file with `gnuplot`. Each of the following lines contains the number of the MC steps, the total energy of the block relative to its energy E_0 before the start of the MC simulation, and three deformation factors of the block in the x , y and z directions relative to the block sizes before the start of the MC simulation. If the atomic sorts are being changed, each line also contains the total number of atoms of every chemical species at the current MC step.

11 Configuration file

The program saves the final configuration in a file whose name is set in the main input file. The format of the configuration file is convenient for plotting and allows you to read this file into the `SOLD` program and continue the simulation. The first nine lines of the file contain information about the boundary conditions, periods of the simulation block in periodic directions, numbers of free, buffer and fixed atoms, and so on. For example, the fifth line contains the number of atoms in the basis of the crystal lattice, the number of free atoms, the number of free and buffer atoms together, and the total number of atoms in the block. These nine head lines start with the symbol #, so that they are treated as comments by `gnuplot` and some other plotting programs. The head lines are followed by lines each defining an atom of the block. Free atoms are listed first, followed by buffer and then fixed atoms (if any). Each line contains the atom number, its Cartesian coordinates in Angstroms, and its chemical sort. Vacancies are listed with the fictitious chemical sort "0". Interstitials are added to the tail of the list of free atoms. Modifying the configuration file manually gives you another way of introducing lattice defects, but you should know well what you are doing.

Following the list of coordinates, there is a line containing either "0" or "1". If it is 1, then this line is followed by a list of all atomic velocities. Each velocity line contains the atom number and the Cartesian coordinates of the velocity vector. If it is 0, then the velocities are not listed or will be ignored by the program. The list of velocities is generated by a previous MD run and is used to restart an MD simulation. A static or MC run, on the other hand, does not produce or use atomic velocities.

12 Output file

The program generates an output file that echoes the input information and reports the calculation results. All messages appearing in the output file are self-explanatory, so we will only make a few comments.

After constructing the simulation block the program reports the lattice periodicity in each direction, the sizes of the inner block, the buffer region and the fixed region (in the number of planes and in Angstroms), as well as the number of atoms in each region. The program also calculates the electron density on atoms and the cohesive energy of the perfect lattice created

by the program. The cohesive energy ε_0 is calculated assuming zero energy of isolated atoms regardless of their chemical sorts.

The file contains all information about the defects introduced into the block and reports its final chemical composition, which can be different from the initial one. The file then reports the deformation applied to the block, if any. At this point the program constructs the table of neighbors and calculates the block energy. The latter is reported as “Initial energy of the defected block”.

Before starting the energy minimization process, the program announces the minimization algorithm and minimization mode together with its short description. After the minimization, the program reports the number of iterations made, the maximum residual force, the maximum atomic displacement during the relaxation process, and the minimum energy E of the simulation block (remember that the program operates with the total energy of free and buffer atoms together). If there are defects in the block, the program calculates and reports the defect energy. If the block contains only point defects, the defect energy is defined as $E - N\varepsilon_0$ (eV), where N is the total number of free and buffer atoms. If there is a grain boundary or a planar fault, the program divides the energy $E - N\varepsilon_0$ by the cross-sectional area of the block in the $x-z$ plane to obtain the defect energy per unit area (eV/Å²). If you prefer a different formula for the defect energy, all relevant information is available in the file.

The program also calculates the mechanical stress tensor averaged over all free atoms of the block. It reports the lower-diagonal part of this tensor (in eV/Å³) as well as its hydrostatic part p (pressure) and the product pV , where V is the block volume (in Å³). By examining these data you can get a better idea about the residual stresses in your system and the necessity of additional relaxation in other modes.

Next, the file reports output information relating to the MD step. It first echoes the input data (ensemble, MD time step, etc.) and then, in the course of the run, makes records of all neighbor table updates and snapshots saved. Upon completion of the run the program reports the average temperature, the final potential energy of the system, and other information.

In the MC regime, the program echoes the simulation temperature and repeats all information regarding the MC stages and the respective numbers of MC steps, ensembles, etc. (a copy of the headlines of the history file). It then reports the total energy before the MC simulation and starts the simulation. The completion of each MC stage is accompanied by a message like

```
Start stage 2....
                        done
```

Upon the completion of all stages, the program reports the average deformation factors of the block in all three directions, the root-mean-squared displacement of an atom, the average energy of the block, and its final energy. The average quantities are obtained by averaging over the production (last but one) stage of the simulation. The program then reports the acceptance probability of trial moves of atoms and the number of times the table of neighbors was updated.

Finally, the program reports the number of times the program calculated the energy, forces, and table of neighbors over the entire run. It also provides a list of wall-clock processor times.

13 Examples

The SOLD program comes with three simple examples stored in the directories `example1/`, `example2/` and `example3/`. For running example 1, execute the command

```
sold < example1/sold-1.dat &
```

and similarly for examples 2 and 3. The output files are created in the respective example directories. Compare them with their “must be” versions to make sure that your installation is correct.

The first example demonstrates the calculation of the intrinsic stacking fault energy in FCC-Cu. The intrinsic stacking fault is obtained by shifting the FCC lattice above a (111) plane by the amount $a/\sqrt{6}$ in the $[\bar{2}11]$ direction. The program first rotates the FCC lattice to align the (111) plane parallel to the x - z plane and the $[\bar{2}11]$ direction parallel to the x axis. It then constructs a simulation block with a fixed boundary condition in the y direction and periodic boundary conditions in the x and z directions. The defect file creates a planar fault by applying a translation $(a/\sqrt{6}, 0, 0)$ to atoms with $y > 0$. The block energy is minimized in mode 4 using the damped MD algorithm, which results in the stacking fault energy of 0.277123×10^{-2} eV/Å² = 44.4 mJ/m². This value was reported in Ref. [2]. The block expansion in the y direction is zero within the accuracy of the calculation.

In the second example, we calculate the energy of a so-called exchange defect in NiAl, namely, two well separated antisites Ni_{Al} and Al_{Ni}. The program generates a 128-atom supercell, turns an Al atom 44 into a Ni atom and a Ni atom 127 into an Al atom, and then relaxes the block in mode 2 using the conjugate gradient method. As a result of the relaxation, the block dimensions increase by a factor of 1.00207 in each direction. The obtained defect energy equals 2.7501 eV. This energy is close to the more accurate value 2.765 eV calculated on a larger simulation block, see Table IX in Ref. [3].

The third example demonstrates the energy minimization in mode 5 by applying it to the D0₁₁ structure of the intermetallic compound Al₃Ni. The orthorhombic D0₁₁ structure is the experimentally observed structure of Al₃Ni under ambient conditions. The block generation file constructs this structure with experimental values of the lattice parameters a , b and c . After the relaxation in mode 5 these parameters change so that a and c slightly increase while b slightly decreases. The equilibrium cohesive energy -3.91 eV is obtained by dividing the minimum energy of the block by the number of free atoms (16). This result is close to the experimental cohesive energy of -4.02 eV [3].

For further details regarding atomistic modeling of lattice defects consult the review articles [5, 6, 7, 8].

14 Acknowledgments

The development of the SOLD package has been supported by the U.S. Department of Defense Common HPC Software Support Initiative (CHSSI).

15 Disclaimer

This software and any accompanying documentation are released “as is”. The U.S. Government makes no warranty of any kind, expressed or implied, concerning this software and any accompanying documentation, including without limitation, any warranties of merchantability or fitness for a particular purpose. In no event will the U.S. Government be liable for any damages, including any lost profits, lost savings, or other incidental or consequential damages arising out of the use, or inability of use, of this software or any accompanying documentation, even if informed in advance of the possibility of such damages.

References

- [1] M. S. Daw and M. I. Baskes, *Phys. Rev. B* **29**, 6443 (1984).
- [2] Y. Mishin, M. J. Mehl, D. A. Papaconstantopoulos, A. F. Voter, and J. D. Kress, *Phys. Rev. B* **63**, 224106 (2001).
- [3] Y. Mishin, M. J. Mehl, D. A. Papaconstantopoulos, *Phys. Rev. B* **65**, 224114 (2002).
- [4] D. Frenkel and B. Smit, *Understanding Molecular Simulation. From Algorithms to Applications*, Second Edition (Academic Press, San Diego, 2002).
- [5] J. H. Harding, *Rep. Prog. Phys.* **53**, 1403 (1990).
- [6] V. Vitek, *Encyclopedia of Advanced Materials* (Pergamon, Oxford, 1994), vol. 1, p. 153.
- [7] P. D. Bristowe, *Encyclopedia of Advanced Materials* (Pergamon, Oxford, 1994), vol. 1, p. 514.
- [8] C. R. A. Catlow, *New Trends in Materials Chemistry* (Kluwer: Dordrecht, 1997), p. 141.