MDStressLab++ User Manual Version 1.0.0

N. C. Admal[b], M. Shi[a], E. B. Tadmor[a]

[a]*Department of Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, MN 55455, USA*
[b]*Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA*

1. Introduction

This is the user manual for *MDStressLab++*, a computer program written in C++ for post processing molecular statics (MS) and molecular dynamics (MD) results to obtain Hardy stress fields. The latest version of the code and this user manual are available online at `http://mdstresslab.org`. When making use of this code, please cite the following articles upon which it is based:

1. Admal, N. C., Tadmor, E. B., 2010. A unified interpretation of stress in molecular systems. Journal of Elasticity 100, 63–143

2. Admal, N. C., Tadmor, E. B., 2011. Stress and heat flux for arbitrary multibody potentials: A unified framework. The Journal of Chemical Physics 134, 184106

3. Admal, N. C., Tadmor, E. B., 2016a. Material fields in atomistics as pull-backs of spatial distributions. Journal of the Mechanics and Physics of Solids 89, 59–76

4. Admal, N. C., Tadmor, E. B., 2016b. The non-uniqueness of the atomistic stress tensor and its relationship to the generalized Beltrami representation. Journal of the Mechanics and Physics of Solids 93, 72–92

5. Shi, M., Admal, N., Tadmor, E., 2020. Noise filtering in atomistic stress calculations for crystalline materials. Journal of the Mechanics and Physics of Solids 144, 104083

1.1. Atomistic stress field definitions

The most commonly used atomistic stress field definitions are the Hardy (1982) stress tensors. For a given choice of weighting function w, the Hardy stress $\boldsymbol{\sigma}_w$ at a point $\boldsymbol{x} \in \mathbb{R}^3$ and time t is given by

$$\boldsymbol{\sigma}_w(\boldsymbol{x}) = \boldsymbol{\sigma}_{w,\mathrm{v}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{w,\mathrm{k}}(\boldsymbol{x}), \tag{1}$$

$$\boldsymbol{\sigma}_{w,\mathrm{v}}(\boldsymbol{x}) = \frac{1}{2}\sum_{\substack{\alpha,\beta \\ \alpha\neq\beta}} \int_{s=0}^{1} [-\boldsymbol{f}_{\alpha\beta}\, w((1-s)\boldsymbol{x}_\alpha + s\boldsymbol{x}_\beta - \boldsymbol{x}) \otimes (\boldsymbol{x}_\alpha - \boldsymbol{x}_\beta)]\, ds, \tag{2}$$

$$\boldsymbol{\sigma}_{w,\mathrm{k}}(\boldsymbol{x}) = -\sum_{\alpha} m_\alpha (\boldsymbol{v}_\alpha^{\mathrm{rel}} \otimes \boldsymbol{v}_\alpha^{\mathrm{rel}}) w(\boldsymbol{x}_\alpha - \boldsymbol{x}), \tag{3}$$

where $\boldsymbol{x}_\alpha \in \mathbb{R}^3$ and $\boldsymbol{v}_\alpha \in \mathbb{R}^3$ denote the position and velocity of particle α in the deformed configuration, $\boldsymbol{f}_{\alpha\beta}$ is the interatomic force between particles α and β, and $\boldsymbol{v}_\alpha^{\mathrm{rel}}$ is the relative velocity of particle α with respect to the particles in its neighborhood.[1] The summations in (2) and (3) are over all particles, and the summation in (2) is a double summation.

The atomistic stress tensors described above are defined on the deformed configuration of particles and correspond to the spatial Cauchy stress (or true stress) defined in continuum mechanics. To make this clear, in the following the prefix "Cauchy" is used when referring to spatial stress measures.

At the same time, in continuum mechanics the stress tensor also has a material description in terms of the first Piola–Kirchhoff stress. In the absence of an explicit deformation mapping, it is uncommon to give a material description of the atomistic stress tensor. However, in Admal and Tadmor (2016a), we have shown that there exists a material description of the atomistic stress in the form of an atomistic Piola–Kirchhoff stress. In particular, for a given reference

[1] See Admal and Tadmor (2016a) and Admal and Tadmor (2016b) for the definition of $\boldsymbol{v}_\alpha^{\mathrm{rel}}$.

and deformed configuration of particles, $\boldsymbol{X}_\alpha$ and $\boldsymbol{x}_\alpha$ respectively, we have shown that the Hardy version of the first Piola–Kirchhoff stress is given by

$$\boldsymbol{P}(\boldsymbol{X}) = \sum_{\substack{\alpha,\beta \\ \alpha<\beta}} \int_{s=0}^{1} [-\boldsymbol{f}_{\alpha\beta}\, w((1-s)\boldsymbol{X}_\alpha + s\boldsymbol{X}_\beta - \boldsymbol{X}) \otimes (\boldsymbol{X}_\alpha - \boldsymbol{X}_\beta)]\, ds, \tag{4}$$

where $\boldsymbol{f}_{\alpha\beta}$ is evaluated for a deformed configuration of particles. Some notable features of the first Piola–Kirchhoff atomistic stress is its non-symmetry and the absence of a kinetic contribution.

1.2. MDStressLab++ stress calculations and required input

The *MDStressLab++* code can calculate fields of the Cauchy and first Piola–Kirchhoff versions of the Hardy stress tensor for three-dimensional systems. The user defines a grid of points and the stress is evaluated at all grid points. Examining the definitions given in equations (1)–(4), we see that the stress tensor code requires the following input:

- Reference (initial) and deformed (final) configuration of particles.
- Velocities of particles.
- The species of each particle (used to identify its mass and used by the force calculations with the interatomic model).
- Details of the periodic boundary conditions (PBCs) and the initial and final size of the box used to prescribe the PBCs. In the present version, we restrict ourselves to orthogonal periodic boundary conditions.
- The potential representation used to evaluate the forces $\boldsymbol{f}_{\alpha\beta}$. This is an interatomic model compatible with the application programming interface (API) of the Knowledge base of Interatomic Models (OpenKIM) version 2 as described below.
- The weighting function used to evaluate the Hardy stress tensor. We recommend to use the lattice-dependent averaging domain (LDAD) to eliminate the noise issue in atomistic stress calculations described in Shi et al. (2020).

1.3. Units

The units used by the *MDStressLab++* program are:

- Distance: Å
- Energy: eV
- Time: ps
- Mass: $\mathrm{eV} \cdot \mathrm{ps}^2/\mathrm{Å}^2$

In the next section, the format of the input file to *MDStressLab++* is described.

2. Input file

Below is a sample C++ input code to *MDStressLab++*. This is followed by a detailed explanation of the codes that appear in it.

```
/*
 * main.cpp
 *
 *  Created on: Nov 9, 2020
 *      Author: Min Shi
 */
```

```
#include <string>
#include <iostream>
#include <tuple>
#include <fstream>
#include "Ldad.h"
#include "Constant.h"
#include "Trigonometric.h"
#include "BoxConfiguration.h"
#include "Sphere.h"
#include "calculateStress.h"
#include "Mls.h"
#include "Grid.h"
#include "typedef.h"
#include "../compareStress.cpp"

int main()
{

//        ----------------------------------------------------------------------
//        Input configuration and potential
//        ----------------------------------------------------------------------

        int numberOfParticles;
        int referenceAndFinal=true;
        std::string configFileName="config.data";
        std::string modelname=\
        "SW_StillingerWeber_1985_Si__MO_405512056662_005";

        std::ifstream file(configFileName);
        if(!file) \
        MY_ERROR("ERROR: config.data could not be opened for reading!");

        file >> numberOfParticles;
        if (numberOfParticles < 0) \
        MY_ERROR("Error: Negative number of particles.\n");

        BoxConfiguration body{numberOfParticles,referenceAndFinal};
        body.read(configFileName,referenceAndFinal);

        Kim kim(modelname);

//        ----------------------------------------------------------------------
//        Create grid
//        ----------------------------------------------------------------------

        int ngrid_pk1 = 125;
        Grid<Reference> gridFromFile_ref(ngrid_pk1);
        gridFromFile_ref.read("grid_pk1.data");

        int ngrid_cauchy = 125;
        Grid<Current> gridFromFile_def(ngrid_cauchy);
        gridFromFile_def.read("grid_cauchy.data");

//        ----------------------------------------------------------------------
```

```
//        Calculate stress on the grid
//        ----------------------------------------------------------------

        Sphere hardy1(20);
        Stress<Sphere,Cauchy> \
        hardyStress1("hardy1",hardy1,&gridFromFile_def);

        calculateStress(body,kim,std::tie(),std::tie(hardyStress1));

        // Ldad stress
        Matrix3d ldadVectors_ref;

        ldadVectors_ref << 5.430949777364731, 0.0, 0.0,
                           0.0, 5.430949777364731, 0.0,
                           0.0,  0.0, 5.430949777364731;

        Ldad<Constant> ldad_Constant_ref(ldadVectors_ref);
        Ldad<Trigonometric> ldad_Trigonometric_ref(ldadVectors_ref);

        Stress<Ldad<Constant>,Piola> \
        ldad_Constant_Stress_ref("ldad_Constant_ref",\
        ldad_Constant_ref,&gridFromFile_ref);
        Stress<Ldad<Trigonometric>,Piola> \
        ldad_Trigonometric_Stress_ref("ldad_Trigonometric_ref",\
        ldad_Trigonometric_ref,&gridFromFile_ref);

        calculateStress(body,kim,std::tie(ldad_Constant_Stress_ref));
        calculateStress(body,kim,std::tie(ldad_Trigonometric_Stress_ref));

//        ----------------------------------------------------------------
//        Perform MLS pushing forward
//        ----------------------------------------------------------------

        double MlsRadius = 5.430949777364731 * 3.0;
        Mls testMls(body,&gridFromFile_ref,MlsRadius,\
        "ldad_Constant_Stress_ref");
        std::vector<Matrix3d> cauchyPushedField;
        testMls.pushToCauchy(ldad_Constant_Stress_ref.field,\
        cauchyPushedField);
        testMls.writeDeformationGradient();
        testMls.writeGridPushed();
        testMls.writePushedCauchyStress(cauchyPushedField);

        return 0;
}
```

The input code is a C++ file which contains four stages: potential and configuration inputs, grid definitions, stress calculation and Moving Least Squares (MLS) pushing forward.

2.1. Stage 'potential and configuration inputs'

Let us look at the code in this stage.

```
//        ----------------------------------------------------------------
//        Input configuration and potential
```

```
//        ------------------------------------------------------------------

        int numberOfParticles;
        int referenceAndFinal=true;
        std::string configFileName="config.data";
        std::string modelname=\
        "SW_StillingerWeber_1985_Si__MO_405512056662_005";

        std::ifstream file(configFileName);
        if(!file) \
        MY_ERROR("ERROR: config.data could not be opened for reading!");

        file >> numberOfParticles;
        if (numberOfParticles < 0) \
        MY_ERROR("Error: Negative number of particles.\n");

        BoxConfiguration body{numberOfParticles,referenceAndFinal};
        body.read(configFileName,referenceAndFinal);

        Kim kim(modelname);
```

In this stage, the interatomic potential used to obtain the interatomic forces is specified. *MDStressLab++* is a OpenKIM-compliant program, which means that it works with interatomic models that are compatible with the OpenKIM Application Programming Interface (API) (see `https://openkim.org`). In order to use OpenKIM models, it is necessary to install the OpenKIM API framework. Moreover, OpenKIM version 2 needs to be installed. This is because for *MDStressLab++*, it only supports OpenKIM version 2, and does not support OpenKIM version 1. This is a prerequisite for using *MDStressLab++*. See the `INSTALL_OpenKIM` file for instructions on how to install the OpenKIM API version 2 and accompanying model files. Here, we couple *MDStressLab++* to a model with the extended OpenKIM ID "SW_StillingerWeber_1985_Si__MO_405512056662_005". This model (Singh, 2018) describes Silicon using a Stillinger-Weber potential.

In order to compute the Hardy atomistic stress tensor, *MDStressLab++* couples to a OpenKIM compliant model and outsources the computation of interatomic forces to the model. Since the atomistic stress tensor field depends on the derivative of the energy with respect to distances between particles, *MDStressLab++* can only couple to models that can compute this derivative. Within the framework of the OpenKIM API, this capability is described by "process_dEdr". Thus only models that support process_dEdr can couple to *MDStressLab++*.

Also, in this stage, the state of the system is read in as an input from the configuration file. The configuration file named "config.data" is read in as an *MDStressLab++* style configuration file. Here is a snippet of the *MDStressLab++* style configuration file "config.data":

```
8000
      54.3094977736473012 0.0 0.0
0.0       54.3094977736473012 0.0
0.0 0.0       54.3094977736473012
      54.3094977736473083 0.0 0.0
0.0       54.8525927513837814 0.0
0.0 0.0       54.3094977736473083
 1    1    1
Si  0.0000000000000000        0.0000000000000000        0.0000000000000000
    0.0000000000000000        0.0000000000000000        0.0000000000000000
    0.0000000000000000        0.0000000000000000        0.0000000000000000
Si  0.0000000000000000        2.7426296375691890        2.7154748886823654
    0.0000000000000000        0.0000000000000000        0.0000000000000000
    0.0000000000000000        2.7154748886823650        2.7154748886823650
Si  2.7154748886823654        0.0000000000000000        2.7154748886823654
```

```
    0.0000000000000000        0.0000000000000000        0.0000000000000000
    2.7154748886823650        0.0000000000000000        2.7154748886823650
```

The first line is the number of atoms in the configuration. Next comes with the initial box size and follows the final box size. After that, periodic boundary conditions (PBC) are specified with 1 meaning true and 0 meaning false. Then, each line specifies an atom in the configuration. It comes with species name, then the final coordinates (deformed configuration), the velocity, and if referenceAndFinal is true, comes with initial coordinates (reference configuration). Note that the value of the final box size (in all periodic directions) must be greater than twice the cutoff of the interatomic model.

we could also use dump2str++.exe which is compiled from dump2str++.cpp in the util directory to convert two LAMMPS dump files into an *MDStressLab++* style configuration file. The two LAMMPS dump files, each containing 1 timestep with the former LAMMPS dump file being served as initial configuration, while the latter LAMMPS dump file being served as the final configuration. However, the ordering rule applies to the two LAMMPS dump files when using the dump2str++.exe. You can add the command `atom_modify sort 0 0` into the lammps input file to explicitly turn off the reordering of the atoms. Furthermore, you can add the command `dump_modify sort id` when outputting the two LAMMPS dump files to explicitly sort the atom id in an ascending order. Moreover, we require that the sequence of the output quantity of the lammps dump file, if containing velocity, to be `id type x y z vx vy vz`. Here, `vx vy vz` are optional and if velocity terms are missing in the final configuration , then the velocity of all atoms will be set to zero.

2.2. Stage 'grid definitions'

Let us look at the code in this stage.

```
//        ------------------------------------------------------------------
//        Create grid
//        ------------------------------------------------------------------

        int ngrid_pk1 = 125;
        Grid<Reference> gridFromFile_ref(ngrid_pk1);
        gridFromFile_ref.read("grid_pk1.data");

        int ngrid_cauchy = 125;
        Grid<Current> gridFromFile_def(ngrid_cauchy);
        gridFromFile_def.read("grid_cauchy.data");
```

In this stage, the grid on which the atomistic stress will be evaluated is constructed in either initial configuration or final configuration. We specifies number of grid points and the grid information is read in from files named "grid_pk1.data" for grids on initial configuration and "grid_cauchy.data" for grids on final configuration. Here is a snippet of the grid file.

```
125

21.7237991136208386      21.7237991136208386      21.7237991136208386
21.7237991136208386      21.7237991136208386      23.0815365582221403
21.7237991136208386      21.7237991136208386      24.4392740028234456
```

The first line is the number of grid points, the second is left blank and starting from the third line follows the coordinates of the grid points.

2.3. Stage 'stress calculation'

```
//        ------------------------------------------------------------------
//        Calculate stress on the grid
//        ------------------------------------------------------------------
```

```
Sphere hardy1(20);
Stress<Sphere,Cauchy> \
hardyStress1("hardy1",hardy1,&gridFromFile_def);

calculateStress(body,kim,std::tie(),std::tie(hardyStress1));

// Ldad stress
Matrix3d ldadVectors_ref;

ldadVectors_ref << 5.430949777364731, 0.0, 0.0,
                   0.0, 5.430949777364731, 0.0,
                   0.0,  0.0, 5.430949777364731;

Ldad<Constant> ldad_Constant_ref(ldadVectors_ref);
Ldad<Trigonometric> ldad_Trigonometric_ref(ldadVectors_ref);

Stress<Ldad<Constant>,Piola> \
ldad_Constant_Stress_ref("ldad_Constant_ref",\
ldad_Constant_ref,&gridFromFile_ref);
Stress<Ldad<Trigonometric>,Piola> \
ldad_Trigonometric_Stress_ref("ldad_Trigonometric_ref",\
ldad_Trigonometric_ref,&gridFromFile_ref);

calculateStress(body,kim,std::tie(ldad_Constant_Stress_ref));
calculateStress(body,kim,std::tie(ldad_Trigonometric_Stress_ref));
```

In this stage, the weighting function defined on either the initial configuration or the final configuration required to compute the Hardy atomistic stress tensor are specified. Two kinds of averaging domain could be used, spherical averaging domain and lattice-dependent averaging domain. For spherical averaging domain, we offer the half linear mollifying weighting function, which has a core constant region for half of its compact support, then at the half compact support place, it decays linearly to zero to the end. It takes the following form as

$$w(r) = \begin{cases} c & \text{if } r < \frac{1}{2} r_{\text{cutoff}}, \\ 2c(1 - \frac{r}{r_{\text{cutoff}}}) & \text{if } \frac{1}{2} r_{\text{cutoff}} \leq r \leq r_{\text{cutoff}}, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

In the above example, we are using a spherical averaging domain of $r_{\text{cutoff}} = 20\text{Å}$ where the atomistic stress tensors are being evaluated on the deformed configuration. The output grid is written to a file named "hardy1.grid"and the output stress is written to a file named "hardy1.stress". The format of the output grid file is the same compared to the input grid file. The first line is the number of grid points. The second line is left blank and starting from the third line comes with the coordinates of the grid points. The format of the output stress file is as following. The first line is the number of the grid points. The second line is left blank and starting from the third line, they consist of 12 columns. The first three columns are the coordinates of the grid points, and the next nine columns are the components of the stress tensor in the following order: 11, 21, 31, 12, 22, 32, 13, 23, 33.

In order to solve the noise issue in atomistic stress calculation, lattice-dependent averaging domain (LDAD) could be used when evaluating the Hardy stress tensor. We offer the following two weighting function forms, constant weighting function and trigonometry weighting function which takes the following form as

$$w_{\text{t}}(\Delta \boldsymbol{X}) = \frac{1}{\Omega} \begin{cases} \prod_{i=1}^{3} (1 + \cos t_i \pi) & t_i \in [-1, 1], \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

$$\tag{7}$$

where $t_i = (\Delta \boldsymbol{X} \cdot \boldsymbol{A}_i)/\|\boldsymbol{A}_i\|^2$ and Ω is the unit cell volume.

In the above example, the lattice-dependent averaging domain defined is a cube on the initial configuration whose length of its edge is $5.43 \times 2 = 10.86$Å with a volume of $10.86^3 = 1281.50$Å^3 consisting of $2 \times 2 \times 2$ conventional unit cells. The grids are being outputted to files named "ldad_Constant_ref.grid" and "ldad_Trigonometric_ref.grid" respectively. The stress tensors are being outputted to files named "ldad_Constant_ref.stress" and "ldad_Trigonometric_ref.stress" respectively. The formats of the output files are the same as mentioned previously.

2.4. Stage 'Moving Least Squares (MLS) pushing forward'

```
//        ------------------------------------------------------------------
//        Perform MLS pushing forward
//        ------------------------------------------------------------------

        double MlsRadius = 5.430949777364731 * 3.0;
        Mls testMls(body,&gridFromFile_ref,MlsRadius,\
        "ldad_Constant_Stress_ref");
        std::vector<Matrix3d> cauchyPushedField;
        testMls.pushToCauchy(ldad_Constant_Stress_ref.field,\
        cauchyPushedField);
        testMls.writeDeformationGradient();
        testMls.writeGridPushed();
        testMls.writePushedCauchyStress(cauchyPushedField);
```

In order to get the clean stress on the final configuration, moving least square (MLS) method could be used to push the noise-free first PK-Hardy stress to the Cauchy-Hardy stress, which we denote the cauchy-pushed stress as a comparison to which we named the cauchy direct stress which is calculated directly on the final configuration. MLS interpolates the displacement field of the atomistic model and thus we could construct the deformation gradient using this information. The MLS-weighting function[2] used in the code is the cubic spline taking the following form as

$$w_{\mathrm{MLS}}(R) = \begin{cases} \frac{2}{3} - 4R^2 + 4R^3, & R \leq \frac{1}{2}, \\ \frac{4}{3} - 4R + 4R^2 - \frac{4R^3}{3}, & \frac{1}{2} < R \leq 1, \\ 0, & R > 1, \end{cases} \tag{8}$$

where $R = [X_1^2 + X_2^2 + X_3^2]^{1/2}/R_{\mathrm{cut}}^{\mathrm{MLS}}$.

In the above example, we set the MLS-cutoff radius to be $R_{\mathrm{cut}}^{\mathrm{MLS}} = 16.292849332094193$Å and push forward the first PK-Hardy stress evaluated using the constant weighting function with LDAD. It is recommended to use $3 \times a_0$ to be the MLS-cutoff radius, where a_0 is the lattice constant of the underlying crystal. The stress tensors that getting pushed forward are being outputted to a file named "ldad_Constant_Stress_ref.CauchyPushed". It has the exact same format as mentioned before. The deformation gradient information is being outputted to a file named "ldad_Constant_Stress_ref.DeformationGradient", which has similar format compared to the stress tensors, with an additional column indicating the Jacobian of the deformation gradient. Also the grids that are pushed forward are being outputted to a file named "ldad_Constant_Stress_ref.pushedGrids". It has the same format compared to other grid files.

3. Examples

The above example named uniaxial strain located in the directory `unit_tests/testDocs` is included with the *MDStressLab++* code.

[2]This is not to be confused with the Murdoch–Hardy weighting function.

3.1. Uniaxial strain

An example of a silicon under uniaxial strain is provided with the *MDStressLab++* code. In this example, Hardy stress tensors are computed for a strained single crystal cube of silicon in the diamond cubic crystal structure such that the x, y and z axis are oriented along the $[100]$, $[010]$, and $[001]$ crystallographic directions. The system is deformed with a strain of 1% in the y direction, and zero strain in the x and z directions. The system is studied under zero temperature.

The intial and final configurations are obtained using LAMMPS (Plimpton, 1995) in the form of two dump files. The LAMMPS input script `in.uniaxialstrain` to generate the dump files is included with this example. Using LAMMPS, the reference configuration of the cube is obtained by stacking $10 \times 10 \times 10$ unit cells with a lattice parameter $a_0 = 5.430949777364731$Å under flexible periodic boundary conditions. The interatomic potential is a three-body Stillinger–Weber potential SW_StillingerWeber_1985_Si__MO_405512056662_005 archived in OpenKIM (Singh (2018); Wen (2018); Elliott and Tadmor (2011)). The reference and deformed configurations are generated in the files dump.initial.0 and dump.final.1, respectively. This example is located in the directory `unit_tests/testDocs`, which contains the following files:

- `README`: Contains instructions for how to run the example.
- `in.uniaxialstrain`: The LAMMPS input script file used to generate the reference and deformed configuration dump files.
- `dump.initial.0`: A LAMMPS generated dump configuration file that contains the positions of atoms corresponding to the reference configuration.
- `dump.final.1`: A LAMMPS generated dump configuration file that contains the positions of atoms corresponding to the deformed configuration.
- `testDocs.cpp`: The input file to *MDStressLab++* setting up the problem and defining what needs to be computed.
- `testDocs`: The executable file to *MDStressLab++* that can run directly.

Note that since the dump files are included in the directory, the user does not need to have LAMMPS installed to generate them.

References

Admal, N. C., Tadmor, E. B., 2010. A unified interpretation of stress in molecular systems. Journal of Elasticity 100, 63–143.

Admal, N. C., Tadmor, E. B., 2011. Stress and heat flux for arbitrary multibody potentials: A unified framework. The Journal of Chemical Physics 134, 184106.

Admal, N. C., Tadmor, E. B., 2016a. Material fields in atomistics as pull-backs of spatial distributions. Journal of the Mechanics and Physics of Solids 89, 59–76.

Admal, N. C., Tadmor, E. B., 2016b. The non-uniqueness of the atomistic stress tensor and its relationship to the generalized Beltrami representation. Journal of the Mechanics and Physics of Solids 93, 72–92.

Elliott, R. S., Tadmor, E. B., 2011. Knowledgebase of Interatomic Models (KIM) application programming interface (API). `https://openkim.org/kim-api`.

Hardy, R. J., 1982. Formulas for determining local properties in molecular dynamics simulation: Shock waves. The Journal of Chemical Physics 76 (01), 622–628.

Plimpton, S., 1995. Fast parallel algorithms for short-range molecular dynamics. Journal of computational physics 117 (1), 1–19.

Shi, M., Admal, N., Tadmor, E., 2020. Noise filtering in atomistic stress calculations for crystalline materials. Journal of the Mechanics and Physics of Solids 144, 104083.

Singh, A. K., 2018. Stillinger-Weber potential for Si due to Stillinger and Weber (1985) v005. OpenKIM, `https://doi.org/10.25950/c74b293f`.

Wen, M., 2018. Stillinger-Weber (SW) Model Driver v004. OpenKIM, `https://doi.org/10.25950/c74b293f`.