FEPX Reference Manual

The documentation for FEPX 1.2.0
A finite element software package for polycrystal plasticity

11 June 2021

ACME Lab

Table of Contents

Conditions of Use

Copying Conditions

FEPX is "free software"; this means that everyone is free to use it and to redistribute it on a free basis. FEPX is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of FEPX that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of FEPX, that you receive source code or else can get it if you want it, that you can change FEPX or use pieces of FEPX in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of FEPX, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for FEPX. If FEPX is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the license for FEPX are found in the General Public License that accompanies the source code (see Appendix C [GNU General Public License], page 55). Further information about this license is available from the GNU Project webpage `http://www.gnu.org/copyleft/gpl-faq.html`.

User Guidelines

If you use FEPX for your own work, please, mention it explicity in your reports and cite the following publication:

- *P.R. Dawson, D.E. Boyce. FEPX – Finite Element Polycrystals: Theory, Finite Element Formulation, Numerical Implementation and Illustrative Examples. arXiv:1504.03296*

1 Introduction

1.1 Description

FEPX is a finite element software package for polycrystal plasticity. It is well-suited to model the global and local mechanical behavior of large polycrystalline solids as aggregates of grains as well as associated microstructural evolution through a highly scalable parallel framework. Each grain is discretized by finite elements whose local behavior corresponds accordingly to the local behavior of a sub-volume of a crystal. These behaviors include:

- Nonlinear kinematics capable of resolving large (or finite) strains and large rotations,
- Anisotropic elasticity based on crystal symmetry,
- Anisotropic plasticity based on rate-dependent slip restricted to dominant slip systems,
- Appropriate state variable evolution for crystal lattice orientation and slip system strengths.

FEPX strives to be a user-friendly, efficient, and robust tool. All of the input data are prescribed non-interactively, using ASCII files. FEPX works hand in hand with Neper (`https://neper.info`), which can be used for both generating the input polycrystal mesh and post-processing the simulation results. Currently, FEPX is designed to provide solutions on rectangular prismatic domains (or right cuboids).

1.2 Resources and Support

Several complementary resources describing FEPX are available:

- The FEPX reference manual, the document you are reading, provides a detailed overview of all of FEPX's capabilities. Specific chapters are dedicated to simulation input and output, running simulations, and various example simulations.
- The FEPX theory manual, written by Paul Dawson and Donald Boyce, provides in depth details on the underlying mechanical theory and finite element methods utilized in FEPX. It is available at `https://arxiv.org/abs/1504.03296`.[1]
- The FEPX website, `https://fepx.info`, gives a general introduction to FEPX with illustrative examples.
- The FEPX GitHub repository, `https://github.com/acmelab-ua/FEPX`, is where the latest version is available and where user interactions take place:
  - To get and keep up-to-date with the latest version, clone the repository using

    ```
    $ git clone https://github.com/acmelab-ua/FEPX.git
    ```

    which gives access to the latest stable development release on the default, `main` branch. To update your local repository, run `git pull` from within the repository.
  - To report bugs, use the issue tracker, `https://github.com/acmelab-ua/FEPX/issues`. When reporting bugs, please provide a minimal working example and the FEPX terminal output.
  - To ask questions, share comments or request new features, use discussions, `https://github.com/acmelab-ua/FEPX/discussions`.

Resources for Neper can be accessed from `https://neper.info`.

[1] Please note that the descriptions of simulation input and output provided in the FEPX theory manual are no longer up-to-date and the user is instead recommended to utilize the descriptions provided in the FEPX reference manual.

1.3 Installing FEPX

FEPX is written in Fortran, and it can run on any Unix-like system (including MacOS). Parallelization of the code is achieved via OpenMPI. Compilation is performed via CMake:

- Create a `build` directory, for instance as a subdirectory of FEPX's `src` directory

  ```
  $ mkdir build
  ```

- Run CMake from within the `build` directory, pointing to FEPX's `src` directory

  ```
  $ cd build
  $ cmake ..
  ```

- Build FEPX

  ```
  $ make
  ```

- Install FEPX on your system (as root)

  ```
  $ make install
  ```

This procedure uses the default configuration options and should work out-of-the-box if you have a Fortran compiler, OpenMPI, and CMake installed. Testing is performed on GFortran version 6 and greater, and OpenMPI version 2 and greater (other Fortran compilers and MPI distributions may also work, though they are not explicitly supported or tested by ACME Lab). A minimum version of CMake version 3.0 is required to utilize the build system.

1.4 Getting Started

To run a serial simulation on a local computer, the '`fepx`' binary must be run in a terminal,

```
$ fepx
```

or, for parallel simulations,

```
$ mpirun -np N fepx
```

where N refers to the number of MPI processes (typically equal to or less than the number of cores on the local machine). The '`fepx`' binary should always be run from within a simulation directory that contains the necessary simulation input files (see Chapter 2 [Simulation Input], page 7).

To perform simulations across multiple computational nodes on an HPC cluster, a submission script that conforms to the specific job scheduling program is necessary. Examples of generic scripts for common job scheduling programs are detailed in Chapter 4 [Running a Simulation], page 37.

During a simulation run, FEPX returns real-time messages in the terminal and, upon successful completion, prints requested output data in ASCII files.

1.5 Reading this Manual

This manual is maintained as a Texinfo manual. Here are the writing conventions used in the document:

- A command that can be typed in a terminal is printed like `this`, or, in the case of a major command, like

  ```
  $ this ;
  ```

- A program (or command) option is printed like `this`;
- The name of a variable is printed like `this`;
- A meta-syntactic variable (i.e. something that stands for another piece of text) is printed like *this*;
- Literal examples are printed like '`this`';

- File names are printed like `this`.

Additionally, hereinafter a *core* will explicitly refer to a processor (or CPU) of a computer. This terminology is also consistent with file name formatting for parallel simulation output by FEPX.

1.6 Development History

The development of FEPX began in the late 1990s and was lead by Paul Dawson, and involved many members of the Deformation Process Laboratory (DPLab) at Cornell University, until early 2020. An extended development history contributed by Paul Dawson, the lead investigator of the DPLab, can be found in Appendix A [Development History], page 49. Ongoing development has since been lead by Matthew Kasemer, and involved other members of the Advanced Computational Materials Engineering Laboratory (ACME Lab) at The University of Alabama.

2 Simulation Input

By default, a minimum of two files is necessary to completely define a simulation. These files are the configuration file, `simulation.config`, and the mesh file, `simulation.msh`. The configuration file defines the material parameters, control of the simulation (i.e., boundary conditions and loading history), printing of output files, and various optional input. The mesh file contains the polycrystal finite element mesh information (grain morphologies, phases and crystal orientations) as well as simulation-related information on the domain faces and the mesh partitions used for parallel simulations.

2.1 The Configuration File (`simulation.config`)

This file contains the necessary definitions to run a simulation. It is structured in several, successive *blocks* that define different aspects of the simulation. Each of these blocks is headed by a line starting by '`##`', which provides a short description of the block. The structure for the `simulation.config` file is as follows:

```
## Optional Input
    key_phrase value
    ...

## Material Parameters
    key_phrase value
    ...

## Deformation History
    key_phrase value
    ...

## Boundary Conditions
    key_phrase value
    ...

## Printing Results
    key_phrase value
    ...
```

Configuration options within a *block* may generally be provided in any order; however, the material parameters and deformation history must follow specific orders. It is recommended that the overall structure of the `simulation.config` file follow the example structure above. The '`Optional Input`' block should always precede any others.

Any piece of text that is preceded by a '`#`' is assumed to be a comment and is ignored (which also makes block headers optional), while '*`key_phrase value`*' is the input structure of the file, where *`key_phrase`* is the input command and *`value`* are the associated parameters for the input command (as will be defined in the following sections). A single line in the file should only ever pertain to a single *`key_phrase`*/*`value`* pairing. All strings are interpreted literally and should be lowercase except where otherwise stated.

2.1.1 Material Description

The model is described in Section 2.1.1.1 [Model Description], page 8, and the corresponding input parameters are made clear and written in fixed font, most often as in 'γ (`gamma`)'. The

specification of the input parameters in the `simulation.config` file is detailed in Section 2.1.1.2 [Input Parameters], page 10.

2.1.1.1 Model Description

The material is described via an elastic response (Hooke's law) and a plastic response (rate dependent plastic flow and hardening).

The stress, σ, is related to the elastic strain, ϵ, via Hooke's law:

$$\sigma = \mathcal{C}\epsilon,$$

where $\mathcal{C}$ is the stiffness tensor. Written in Voigt notation, the above equation is expanded for cubic materials:

$$\begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{Bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{12} & & & \\ C_{12} & C_{11} & C_{12} & & & \\ C_{12} & C_{12} & C_{11} & & & \\ & & & C_{44} & & \\ & & & & C_{44} & \\ & & & & & C_{44} \end{bmatrix} \begin{Bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{Bmatrix}.$$

Here, the strength of materials convention is utilized, where the shear factors of 2 are written in the strain vector. Special attention must be paid to ensure that the correct stiffness values are chosen, to align with the input convention used here.

For this convention, the Zener anisotropy ratio for cubic materials (which quantifies the level of elastic anisotropy, with 1 being perfectly isotropic) would be written as:

$$A = \frac{2C_{44}}{C_{11} - C_{12}}.$$

For example, Tungsten (W) is a nearly perfectly elastically isotropic cubic (BCC) material, with $C_{11} = 522.4$ GPa, $C_{12} = 204.4$ GPa, and $C_{44} = 160.8$ GPa. This would yield a Zener ratio of 1.01.

The elastic constitutive relation may also be expanded for hexagonal materials:

$$\begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{Bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & & & \\ C_{12} & C_{11} & C_{13} & & & \\ C_{13} & C_{13} & C_{11} & & & \\ & & & C_{44} & & \\ & & & & C_{44} & \\ & & & & & (C_{11} - C_{12})/2 \end{bmatrix} \begin{Bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{Bmatrix}.$$

Note here that there is no parameter for C_{33}, as the hydrostatic and deviatoric portions of the motions are decoupled. This requires that: $C_{11} + C_{12} = C_{13} + C_{33}$.

Overall, the stiffness tensor is thus defined by the input parameters `c11`, `c12`, and `c44` for cubic materials, and `c11`, `c12`, `c13`, and `c44` for hexagonal materials.

The kinematics of slip are described by a power law:

$$\dot{\gamma}^{\alpha} = \dot{\gamma}_0 \left(\frac{|\tau^{\alpha}|}{g^{\alpha}} \right)^{1/m} \operatorname{sgn}(\tau^{\alpha}),$$

where $\dot{\gamma}_0$ (`gammadot_0`) is the fixed-rate strain rate scaling coefficient (expressed in [force/area][1]), and m (`m`) is the rate sensitivity exponent.

For an isotropic hardening assumption[2], slip system strength evolution (hardening) is modeled by:

$$\dot{g^\alpha} = h_0 \left(\frac{g_{s0} - g^\alpha}{g_{s0} - g_0} \right)^n \dot{\gamma},$$

where h_0 (`h_0`) is the fixed-state hardening rate scaling coefficient, where g_{s0} (`g_s0`) is the initial slip system saturation strength (expressed in [force/area]), g_0 (`g_0`) is the initial slip system strength (expressed in [force/area]), and n (`n`) is the non-linear Voce hardening exponent. In the above equation, $\dot{\gamma}$ is calculated as:

$$\dot{\gamma} = \sum_\alpha \left| \dot{\gamma}^\alpha \right| .$$

The slip system saturation strength may be evolved as a function of the slip activity. In this case, the hardening expression takes the form:

$$\dot{g^\alpha} = h_0 \left(\frac{g_s(\dot{\gamma}) - g^\alpha}{g_s(\dot{\gamma}) - g_0} \right)^n \dot{\gamma},$$

where $g_s(\dot{\gamma})$ is the function for the saturation strength, which evolves via:

$$g_s(\dot{\gamma}) = g_{s0} \left(\frac{\dot{\gamma}}{\dot{\gamma}_{s0}} \right)^{m'},$$

where g_{s0} (`g_s0`) is the initial slip system saturation strength (expressed in [force/area]), m' (`m_prime`) is the saturation strength rate scaling exponent, and $\dot{\gamma}_{s0}$ (`gammadot_s0`) is the initial saturation slip system shear rate. Again, in the above two equations, $\dot{\gamma}$ is calculated as the sum of the absolute value of the individual slip system shear rates, as defined above.

For a cyclic hardening assumption, the slip system strength evolution (hardening) is modeled by:

$$\dot{g^\alpha} = h_0 \left(\frac{g_s(\dot{\gamma}) - g^\alpha}{g_s(\dot{\gamma}) - g_0} \right)^n f,$$

[1] All variables presented in this chapter (if applicable) are detailed by their dimensions instead of any specific unit. No unit system is inherently assumed by FEPX and the chosen unit system and value magnitudes should be consistent with the chosen length scale for the domain. For example, if it is assumed that the length scale is '`mm`' and SI units are to be used, then [force/area] will be understood to be [MPa]. The unit for time, however, is always assumed to be seconds [s].

[2] Note that for HCP materials, the implementation of an isotropic hardening assumption is such that the shape of the single crystal yield surface is maintained. That is, the slip rates on the basal, prismatic, and pyramidal slip systems will harden such that the ratios of slip strengths remains constant. The consequence of this is that for HCP materials, this is not a true isotropic assumption, as the different slip families may harden at different rates (depending on the ratios of slip system strengths).

where f is calculated as:

$$f = \sum_{\beta=0}^{n_a} \left| \dot{\gamma}^{\beta} \right| .$$

A slip system that contributes to hardening (n_a total systems contributing to hardening) is that which has a change in shear greater than a critical value:

$$\Delta\gamma_{crit} = a \left[g/g_s(\dot{\gamma}) \right]^c ,$$

where the material parameters here are a (`cyclic_a`) and c (`cyclic_c`). A more complete description can be found in Turkmen *et al.* [22]. Note that minor differences exist between the implemented model described above and the formulation described in the paper.

For an anisotropic hardening assumption, slip system strength evolution (hardening) is modeled by:

$$\dot{g^{\alpha}} = h_0 \left(\frac{g_s(\dot{\gamma}) - g^{\alpha}}{g_s(\dot{\gamma}) - g_0} \right)^n \dot{\gamma} h_{\alpha\beta},$$

where the model parameters are the same as the isotropic case described above, with the addition of $h_{\alpha\beta}$, the slip interaction matrix. The slip interaction matrix only allows for interactions from direct and coplanar slip families. The slip interaction matrix is defined by the diagonal entry, d, and the off-diagonal entries, $h_1, \ldots, h_n$. These input parameters are defined by `diag`, `h1`, `h2`, `h3`, and `h4` for FCC materials, `diag`, `h1`, `h2`, `h3`, `h4`, `h5`, and `h6` for BCC materials, and `diag`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, and `h7` for HCP materials. A more complete description can be found in Carson *et al.* [21].

For a hexagonal material, the c/a ratio (`c_over_a`) must also be defined. The initial slip strengths (`g_0`) must be provided as a set of three values. The order of the values for `g_0` is assumed to be: basal slip family strength, prismatic slip family strength, and pyramidal slip family strength.

2.1.1.2 Input Parameters

The material (as defined in the mesh file) can include one or several phases (to which grains are assigned), and the mechanical behavior of these phases must be defined accordingly. The number of phases must first be provided:

```
number_of_phases value
```

The material parameters for a particular phase should be defined entirely for said phase before parameters for any subsequent phases are defined.

Each phase requires the specification of a consistent set of single-crystal material parameters, prefaced by

```
phase value
```

where *value* is the phase identification number (ranging from 1 to *nphases*).

First, the crystal symmetry is defined by:

```
crystal_type type_string
```

where *type_string* is the crystal symmetry type. Options for *type_string* are: '`fcc`', '`bcc`', and '`hcp`', for face-centered cubic, body-centered cubic, and hexagonal close-packed, respectively.

Then, the single-crystal elastic and plastic material parameters of the phase must be defined. Depending on the crystal symmetry, the total number of required parameters varies. Below, all *value* inputs are expected to be real numerical values.

Anisotropic elastic constants are defined using the strength of materials convention, as described previously in Section 2.1.1.1 [Model Description], page 8. The input is, for '`fcc`' and '`bcc`' crystal structures:

```
c11 value
c12 value
c44 value
```

and, for an '`hcp`' crystal structure:

```
c11 value
c12 value
c13 value
c44 value
```

where `c11-c44` are expressed in [force/area]. For '`hcp`' materials, the C_{33} elastic constant is constrained by the other four nonzero moduli and is not required as direct input.

For the '`hcp`' crystal structure, an additional crystal parameter needs to be provided:

```
c_over_a value
```

Crystallographic slip (plasticity) parameters are defined as:

```
m value(s)
gammadot_0 value
h_0 value
g_0 value(s)
g_s0 value
n value
```

For hexagonal materials, multiple values may be provided for the rate sensitivity exponent, '`m`', and the initial slip system strengths, `g_0`. If a single value is provided for `m`, a constant rate sensitivity exponent is assumed across all slip families. If three values are provided for `m`, varying rate sensitivity exponents are applied on a per-slip family basis. Additionally, `g_0` must be defined by three unique values. The order of the values for `m` and `g_0` is assumed to be: basal slip family, prismatic slip family, and pyramidal slip family.

Saturation strength evolution (optional) is by default disabled, and may be enabled by defining both of the necessary parameters for saturation strength evolution. These parameters do not need to be defined if the saturation strength is not intended to evolve. To enable saturation strength evolution, define:

```
m_prime value
gammadot_s0 value
```

Cyclic hardening (optional) is by default disabled. It may be enabled via:

```
hard_type cyclic_isotropic
```

If *cyclic* hardening is enabled, each phase requires the definition of two additional parameters by:

```
cyclic_a value
cyclic_c value
```

where both `cyclic_*` values are model parameters for a critical value of accumulated shear strain used to modify the form of the Voce hardening law [22].

Anisotropic hardening (optional) is by default disabled. Anisotropic hardening may be enabled via:

```
hard_type anisotropic
```

If *anisotropic* hardening is enabled, each phase requires the definition of slip interaction matrix values which vary based on crystal symmetry.

For an '`fcc`' crystal structure:

```
latent_parameters diag h1 h2 h3 h4
```

For an '`bcc`' crystal structure:

```
latent_parameters diag h1 h2 h3 h4 h5 h6
```

For an '`hcp`' crystal structure:

```
latent_parameters diag h1 h2 h3 h4 h5 h6 h7
```

where *diag* is the diagonal coefficient and *h1-h7* are the slip family coefficients [21].

2.1.2 Deformation History

A variety of deformation modes are available that are capable of reproducing various mechanical loading configurations. A deformation history is defined by both steps and increments, where steps are made of one or several increments. Steps define the strain or load targets that are to be reached during the simulation, and results can only be printed at the end of steps. One or several increments occur within each step to reach the prescribed step target while ensuring numerical stability. A step target can be expressed in terms of *strain* or *load*. The *strain* refers to the engineering strain as computed from the displacement of the loading surface and the initial sample length along the loading direction, and the *load* refers to the total force on the loading surface. Of course, the relative order of the steps defined for the deformation history matters and should be written in an ascending manner.

Deformation histories are divided into uniaxial loading and multiaxial loading. In general, the multiaxial loading definition is technically triaxial in nature; however, biaxial loading may be performed by zeroing out one of the load columns accordingly. The available deformation history configuration options follow.

2.1.2.1 Uniaxial

Uniaxial loading is always strain controlled (internally); however, either specific strain targets or specific load targets may be prescribed. For strain targeting, the number of increments for a given step must be provided as opposed to a time-step value. For load targeting, the bounds

on the time-step value are provided in order to control both the accuracy and, indirectly, the number of increments taken per step. These time-step values should be defined relative to the `strain_rate` value (see Section 2.1.3 [Boundary Conditions], page 15).

Strain targeting allows the definition of loading to specific uniaxial strain states. This deformation history is defined as follows:

```
def_control_by uniaxial_strain_target
number_of_strain_steps nsteps
target_strain target_val n_incr print_flag
...
```

where *nsteps* is the number of strain steps that are defined in the file after this line, *target_val* is the desired strain value to be reached, *n_incr* is the number of increments to be performed in order to complete the step, and *print_flag* allows for the printing (or not) of specific steps. The options available for *print_flag* are: '`print_data`' or '`suppress_data`'.

Load targeting allows the definition of loading to specific uniaxial load states. This deformation history is defined as follows:

```
def_control_by uniaxial_load_target
number_of_load_steps nsteps
target_load target_val dt_max dt_min print_flag
...
```

where *nsteps* is the number of load steps that are defined in the file after this line, *target_val* is the desired load value to be reached, *dt_max* is the maximum time-step value to be used for a given increment, *dt_min* is the minimum time-step value to be used for a given increment, and *print_flag* allows for the printing (or not) of specific steps. The options available for *print_flag* are: '`print_data`' or '`suppress_data`'.

Strain rate jumps are also available for both uniaxial deformation modes and are defined by adding the following input to the block:

```
number_of_strain_rate_jumps njumps
strain_rate_jump target_step new_strain_rate
...
```

where *njumps* is the number of strain rate jumps defined in the file after this line, *target_step* defines which `target_strain` step is assigned a new strain rate, and *new_strain_rate* is the new strain rate to be assigned and has units of [1/s]. In general, and for numerical stability, the strain rate jumps should be of a similar magnitude to the `strain_rate` defined previously.

2.1.2.2 Multiaxial

Multiaxial loading is always strain controlled (internally) and operates at either a constant engineering strain rate or constant load rate; however, only specific load targets may be prescribed. The principal loading directions must be aligned with the coordinate axes of the mesh and the surface face normals should likewise be coincident with the coordinate axis of the mesh. Symmetry boundary conditions (zero normal velocities) are enforced on the three faces of minimal coordinates ('**0*'), and, in the general case, non-zero normal velocities are applied to the faces of maximal coordinates ('**1*'). The velocity on the primary control surface is held constant through the simulation (except during a strain rate jump).

Multiaxial loading with a constant strain rate (CSR) is defined as follows:

```
def_control_by triaxial_constant_strain_rate
```

```
number_of_csr_load_steps nsteps
target_csr_load load_x load_y load_z dt_max dt_min print_flag
...
```

where *nsteps* is the number of CSR load steps that are defined in the file after this line, *load_x* is the desired load value to be reached in the 'x' coordinate axis direction, *load_y* is the desired load value to be reached in the 'y' coordinate axis direction, *load_z* is the desired load value to be reached in the 'z' coordinate axis direction, *dt_max* is the maximum time-step value to be used for a given increment, *dt_min* is the minimum time-step value to be used for a given increment, and *print_flag* allows for the printing (or not) of specific steps. The options available for *print_flag* are: 'print_data' or 'suppress_data'.

Strain rate jumps are available for this deformation mode and are defined by adding the following input to the block:

```
number_of_strain_rate_jumps njumps
strain_rate_jump target_step new_strain_rate
...
```

where *njumps* is the number of strain rate jumps defined in the file after this line, *target_step* defines which target_csr_load step is assigned a new strain rate, and *new_strain_rate* is the new strain rate to be assigned and has units of [1/s].

Multiaxial loading with a constant load rate (CLR) is defined as follows:

```
def_control_by triaxial_constant_load_rate
number_of_clr_load_steps nsteps
target_clr_load load_x load_y load_z target_time_incr print_flag
...
```

where *nsteps* is the number of CLR load steps that are defined in the file after this line, *load_x* is the desired load value to be reached in the 'x' coordinate axis direction, *load_y* is the desired load value to be reached in the 'y' coordinate axis direction, *load_z* is the desired load value to be reached in the 'z' coordinate axis direction, *target_time_incr* is the physical time increment to be reached for the given target_clr_load steps for a given load rate, and *print_flag* allows for the printing (or not) of specific steps. The options available for *print_flag* are: 'print_data' or 'suppress_data'.

Load rate jumps and dwell episodes are available for this deformation mode. A dwell episode maintains the macroscopic loads of the step in which it is defined, but holds the ramp rate at zero for the amount of time defined by *dwell_time*. These options are defined as follows:

For load rate jumps:

```
number_of_load_rate_jumps njumps
load_rate_jump target_step new_ramp_rate
...
```

where *njumps* is the number of load rate jumps defined in the file after this line, *target_step* defines which target_clr_load step is assigned a new load rate, and *new_load_rate* is the new load rate to be assigned and has units of [force/s].

For dwell episodes:

```
number_of_dwell_episodes nepisodes
dwell_episode target_step dwell_time target_time_incr print_flag
...
```

where *nepisodes* is the number of dwell episodes defined in the file after this line, *target_step* defines which target_clr_load step is assigned to dwell, *dwell_time* is the physical amount of time in [s] for a given dwell episode, *target_time_incr* is the physical time increment to be reached for the given dwell episode, and *print_flag* allows for the printing (or not) of specific steps. The options available for *print_flag* are: 'print_data' or 'suppress_data'.

2.1.3 Boundary Conditions

Standard, simple boundary conditions are available for automatic definition with minimal input and are computed internally for each simulation based on the definitions in the simulation.config file. This ensures that standard boundary conditions are consistently defined for all simulations and increases the portability of the simulation.config file. Alternatively, custom boundary conditions can be defined, as described separately, in Section 2.1.4 [Externally Defined Boundary Conditions (Optional)], page 17.

2.1.3.1 Uniaxial

Uniaxial definitions are available for three different constraint configurations. The available uniaxial constraint configuration options follow.

Grip boundary conditions fully constrain two opposite faces in the spatial domain. The first face is fully fixed in all sample directions while the second face has a strain rate applied in the face normal direction while the other two sample directions are fully fixed. All other faces are unconstrained.

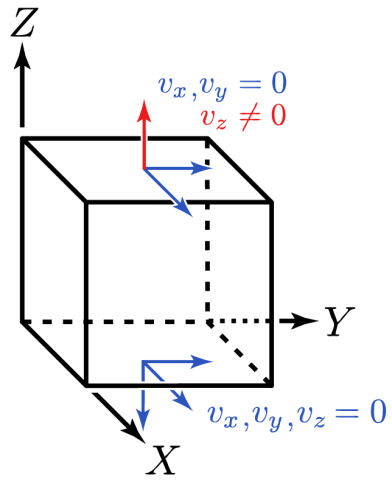

Figure 1. Simplified schematic of the applied velocities for grip boundary conditions. The loading face is 'z1' and the sample is being loading in the '+Z' direction.

Grip boundary conditions are defined as follows:

```
boundary_conditions uniaxial_grip
loading_direction sample_dir
loading_face face_label
strain_rate value
```

where *sample_dir* is the direction along the a positive sample axis in which the sample is loaded, *face_label* is the face on which the loading is applied (the opposing face is fully fixed), and *value* is the strain rate value in units of [1/s].

Symmetry boundary conditions constrains four faces in the spatial domain. The three **0* faces are fixed in the face normal directions and unconstrained in the other two sample directions. The fourth **1* face has a strain rate applied in the face normal direction while the other two

sample directions are fully fixed. The selection of the **1* face is based on the defined `loading_direction`.

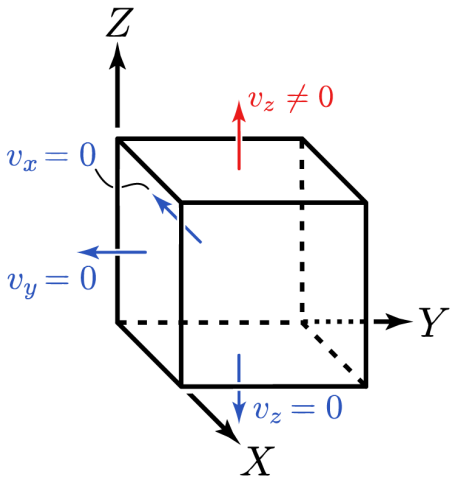

Figure 2. Simplified schematic of the applied velocities for symmetry boundary conditions. The sample is being loaded in the '`+Z`' direction.

Symmetry boundary conditions are defined as follows:

```
boundary_conditions uniaxial_symmetry
loading_direction sample_dir
strain_rate value
```

where *`sample_dir`* is the direction along the a positive sample axis in which the sample is loaded, and *`value`* is the strain rate value in units of [1/s].

Minimal boundary conditions are a modification of grip boundary conditions that only constrain two opposite faces in the face normal directions and two corner nodes in the spatial domain. The selection of the constrained faces is based on the defined `loading_direction`. The first node is always fully fixed where the **0* faces converge. The second node is defined relative to the defined `loading_direction` and is constrained to prevent rigid body rotation about the `loading_direction` axis.

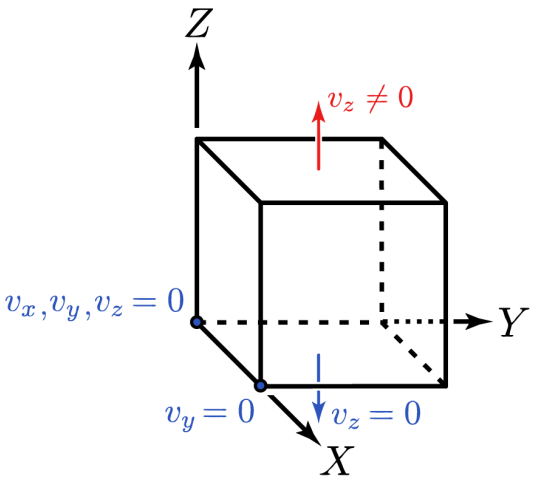

Figure 3. Simplified schematic of the applied velocities for minimal boundary conditions. The sample is being loaded in the '`+Z`' direction. The two blue corner nodes are constrained to prevent rigid body translation and motion.

Minimal boundary conditions are defined as follows:

```
boundary_conditions uniaxial_minimal
loading_direction sample_dir
strain_rate value
```

where *sample_dir* is the direction along the a positive sample axis in which the sample is loaded, and *value* is the strain rate value in units of [1/s].

2.1.3.2 Multiaxial

Multiaxial boundary conditions are generally consistent across modes, however, the input rate type varies depending on the mode. For both modes, the `loading_direction` defines the primary control direction in which the normal velocities are held constant throughout the simulation.

Multiaxial loading with a constant strain rate (CSR) is defined as follows:

```
boundary_conditions triaxial
loading_direction sample_dir
strain_rate value
```

where *sample_dir* is the direction along the a positive sample axis in which the sample is loaded, and *value* is the strain rate value in units of [1/s].

Multiaxial loading with a constant load rate (CLR) is defined as follows:

```
boundary_conditions triaxial
loading_direction sample_dir
load_rate value
```

where *sample_dir* is the direction along the a positive sample axis in which the sample is loaded, and *value* is the loading rate value in units of [force/(area-s)].

2.1.4 Externally Defined Boundary Conditions (Optional)

Boundary conditions different from the ones defined in Section 2.1.3 [Boundary Conditions], page 15, can be applied using an external `simulation.bcs` containing per-node constraints. Of course, this file should be generated for a singular `simulation.msh` file (if the finite element mesh in the `simulation.msh` file changes, then the associated `simulation.bcs` file will need to be generated anew).

To read in external boundary conditions, the following line must be added to the `simulation.config` file:

```
read_bcs_from_file
```

The per-node constraints are defined as follows, in the `simulation.bcs` file:

```
node_id coord_index vel
...
```

where *node_id* is a unique 1-indexed identification number, *coord_index* defines the sample axis the constraint is applied to, and *vel* is the velocity being applied to the node in the constraint direction. The options for *coord_index* are: '`x`', '`y`', or '`z`'. A singular *coord_index*/*vel* pair should be defined per-line for a given *node_id*. The velocities should be prescribed relative to the mesh dimensions and time-step size in order to produce expected strain rates.

2.1.5 Externally Defined Phases and Orientations (Optional)

Crystallographic phase and orientations different from the ones defined in the `simulation.msh` file can be defined in external files by adding appropriate commands to the configuration file.

To read in external orientations, the following line must be added to the `simulation.config` file:

```
read_ori_from_file
```

More detailed information on the structure of this external file can be found in Section 2.2.1 [External Orientation Assignment (Optional)], page 21.

To read in external grain/phase assignments, the following line must be added to the `simulation.config` file:

```
read_phase_from_file
```

More detailed information on the structure of this external file can be found in Section 2.2.2 [External Crystallographic Phase Assignment (Optional)], page 23.

2.1.6 Printing Results

Each field variable file to be output from a simulation must be individually defined. This includes nodal output, elemental output, simulation restart information, fiber-averaging output, and other miscellaneous output (see Chapter 3 [Simulation Output], page 25, for a complete description of all output). The printing of a given field variable file is defined as follows:

```
print output_file_name
```

where *`output_file_name`* is the particular field variable file to be output. The available options for *`output_file_name`* are:

- '`coo`' - Nodal coordinates
- '`crss`' - Critical resolved shear stress
- '`defrate`' - Deformation rate tensor
- '`defrate-eq`' - Equivalent deformation rate
- '`defrate-pl`' - Plastic deformation rate tensor
- '`defrate-pl-eq`' - Equivalent plastic deformation rate
- '`disp`' - Nodal displacements
- '`elt-vol`' - Elemental volume
- '`ori`' - Crystallographic orientations
- '`slip`' - Slip system shear
- '`sliprate`' - Slip system shear rate
- '`spinrate`' - Plastic spin rate tensor
- '`strain`' - Total strain tensor
- '`strain-eq`' - Equivalent total strain
- '`strain-el`' - Elastic strain tensor
- '`strain-el-eq`' - Equivalent elastic strain
- '`strain-pl`' - Plastic strain tensor
- '`strain-pl-eq`' - Equivalent plastic strain
- '`stress`' - Stress tensor
- '`stress-eq`' - Equivalent stress

- '`vel`' - Nodal velocity
- '`velgrad`' - Velocity gradient tensor
- '`work`' - Work
- '`work-pl`' - Plastic work
- '`workrate`' - Work Rate
- '`workrate-pl`' - Plastic work rate
- '`forces`' - Surface forces
- '`convergence`' - Simulation convergence statistics
- '`restart`' - Simulation restart data

A full description of each output variable can be found in Chapter 3 [Simulation Output], page 25.

2.1.7 Fiber Averaging of Output Variables (Optional)

The average and standard deviation of certain output variables may be calculated depending on whether or not they belong to a user-defined crystallographic fiber. To activate this option, the following line must be added to the `simulation.config` file:

```
run_fiber_average
```

More detailed information on the structure of this external file can be found in Section 2.3 [The Fiber Averaging File (Optional)], page 23.

2.1.8 Optional Input Parameters

These options may pertain to specific deformation modes or control standard simulation behavior. All possible inputs presented in this section have default values already defined.

- `max_incr` *`integer`* specifies the maximum number of increments (default: '`50000`').
- `max_total_time` *`real`* specifies the maximum deformation time (default: '`12000.0`').
- `check_necking` `on/off` specifies whether or not to terminate simulation when specimen begins to neck (default: '`off`').
- `load_tol` *`positive_real`* is the the target load tolerance. A small positive load tolerance (e.g. 0.1 x control surface area) improves load control while reducing the number of small steps near target loads (default: '`0.0`').
- `dtime_factor` *`real`* is a number greater than or equal to 1 which is used when calculating time increments near target loads (default: '`1.001`').
- `hard_type` `isotropic/anisotropiccyclic_isotropic` specifies the hardening model to use (default: '`isotropic`').
- `max_bc_iter` *`integer`* specifies the maximum number of boundary condition iterations (default: '`10`').
- `min_pert_frac` *`real`* is the minimum fraction of the control velocity by which the secondary and tertiary surface velocities are perturbed during boundary condition iterations (default: '`0.001`').
- `load_tol_abs` *`real`* is the absolute tolerance on the secondary and tertiary loads. The absolute load criterion is that both loads are within the absolute load tolerance of the ideal load. Loads are considered to be within tolerance if either the absolute or relative criterion is satisfied (default: '`0.1`').
- `load_tol_rel` *`real`* is the relative load tolerance on the secondary and tertiary loads. It represents a fraction of the load in the control direction. The relative load criterion is

that the difference between the load and ideal load, normalized by the load in the control direction, is less than the relative load tolerance. Loads are considered to be within tolerance if either the absolute or relative criterion is satisfied (default: '`0.001`').

- `max_strain_incr` *`real`* specifies the maximum strain increment for dwell episodes (default: '`0.001`').
- `max_strain` *`real`* specifies the maximum allowable macroscopic strain (default: '`0.2`').
- `max_eqstrain` *`real`* specifies the maximum allowable macroscopic equivalent strain (default: '`0.2`').
- `max_iter_hard_limit` *`integer`* specifies the maximum allowable iterations on the Backward Euler approximation used to update hardnesses (default: '`10`').

Additional input commands related to fiber-averaging routines, restart capabilities and external file read-in are presented separately, see Section 2.3 [The Fiber Averaging File (Optional)], page 23, Section 4.2 [Restarting a Simulation], page 38, Section 2.2.1 [External Orientation Assignment (Optional)], page 21, and Section 2.2.2 [External Crystallographic Phase Assignment (Optional)], page 23, respectively.

2.1.9 Optional Convergence Parameters

These options modify the tolerances and general behavior of the solution algorithms and should only be modified by those who know what they are doing. All possible inputs presented in this section have default values already defined.

2.1.9.1 Velocity Convergence

The velocity solver employs a hybrid successive-approximation/Newton-Raphson algorithm. Convergence of the velocity solution is based on a convergence parameter, which unless otherwise noted, is defined as the norm of the change in the velocity field, divided by the norm of the velocity field, $||\Delta u||/||u||$. Other parameters are also used to assess the convergence of the velocity solution. The following parameters pertain to the convergence of the velocity solver:

- `nl_max_iters` *`integer`* specifies the maximum allowable number of iterations of the nonlinear velocity solver (default: '`50`').
- `nl_tol_strict` *`real`* specifies the desired tolerance on the elasto-viscoplastic velocity solution (default: '`5e-4`').
- `nl_tol_loose` *`real`* specifies an acceptable level of convergence if the desired level of convergence cannot be reached via '`nl_tol_strict`' (default: '`5e-4`').
- `nl_tol_min` *`real`* is the tolerance on the norm of the change in velocity, divided by the number of degrees of freedom, ($||u||$/max(ndof)). This parameter is useful for assessing convergence when the macroscopic velocity is near zero (default: '`1e-10`').
- `nl_tol_switch_ref` *`real`* is the value of the convergence parameter at which the solution algorithm switches from successive-approximation to Newton-Raphson. To only use successive-approximations, set the value of '`nr_tol_switch_ref`' equal to the value of '`nl_tol_strict`' (default: '`1e-2`').
- `nl_tol_conv` *`real`* is a parameter between 0 and 1 that is used to assess whether the Newton-Raphson algorithm is converging slowly (default: '`0.2`').

2.1.9.2 Conjugate Gradient Convergence

The solution of the linear system of equations $[K]\{\Delta u\} = -\{R\}$ is performed using a conjugate gradient solver. The following parameters pertain to the convergence of the conjugate gradient solver:

- `cg_max_iters` *`integer`* specifies the maximum allowable number of iterations of the conjugate gradient solver (default: '`16000`').
- `cg_tol` *`real`* specifies the desired tolerance on the conjugate gradient solver (default: '`1e-8`').

2.1.9.3 Material State Convergence

The convergence of the material stress state for both the viscoplastic and elasto-viscoplastic solutions is assessed by the following parameters:

- `sx_max_iters_state` *`integer`* specifies the maximum number of iterations on material state (default: '`100`').
- `sx_max_iters_newton` *`integer`* specifies the maximum number of iterations of the Newton algorithm used to solve for crystal stress (default: '`100`').
- `sx_tol` *`real`* specifies the tolerance on the stress solution (default: '`1e-4`').

2.2 The Mesh File (`simulation.msh`)

This file contains the finite element mesh information along with phase assignments and crystal orientations. The mesh file is generally generated by Neper and not directly modified. A brief description is provided below, which a more complete description can be found in the Neper reference manual, `https://neper.info/docs/neper.pdf`. The file can be opened by Gmsh for interactive visualization.

The file is structured in several, successive *fields* that define different aspects of the mesh. Each of these fields is wrapped by '`$`*`Field`*`/$End`*`Field`*' lines, where '*`Field`*' is a short description of the information stored within the block. A typical `simulation.msh` file will contain the following information:

- Mesh Format ('`$MeshFormat`'),
- Mesh Version ('`$MeshVersion`'),
- Nodes ('`$Nodes`'),
- Elements ('`$Elements`'),
- Surface Element Sets ('`$Fasets`'),
- Crystal Orientations ('`$ElsetOrientations`' or '`$ElementOrientations`'),
- Grain/Phase Assignments ('`$Groups`').

Additionally, the `simulation.msh` file may also include fields with partition information for both the nodes and elements if the domain is decomposed for parallel execution and surface node sets ('`$NSets`').

Embedded microstructural information (phases and orientations) with the `simulation.msh` may be overridden by external files, `simulation.ori` and `simulation.phase`, if the appropriate commands are added to the `simulation.config` file (see Section 2.1.5 [Externally Defined Orientations or Phase Assignments (Optional)], page 18).

2.2.1 External Orientation Assignment (Optional, `simulation.ori`)

The embedded orientation assignments within the `simulation.msh` may be overridden via an external `simulation.ori` file. This file contains formatting identical to the associated fields in the mesh file and is defined as:

For per-grain (or '`Elset`') orientations:

```
$ElsetOrientations
number_of_ori_entities orientation_descriptor:orientation_convention
```

```
entity_id ori_des1 ...
...
$EndElsetOrientations
```

For per-element orientations:

```
$ElementOrientations
number_of_ori_entities orientation_descriptor:orientation_convention
entity_id ori_des1 ...
...
$EndElementOrientations
```

where *number_of_ori_entities* is the number of unique orientations defined in the section, *orientation_descriptor* is the parameterization for the orientations (see options below), *orientation_convention* describes the basis transformation route for the orientations provided, *entity_id* is a unique 1-indexed identification number, and *ori_des** are the components of the unique orientation. Available options for *orientation_convention* are: '`active`' or '`passive`'. Following the usual terminology, an *active* orientation assumes that which describes a basis transformation from the sample basis to the crystal basis (sample-to-crystal), while a *passive* orientation convention assumes that which describes a basis transformation from the crystal basis to the sample basis ("crystal-to-sample").

The following *orientation_descriptor* types are available (associated per-line formats are also described):

- For '`rodrigues`', each orientation is described by r_1, r_2, r_3, where the Rodrigues vector is $\mathbf{r} = \mathbf{t}\tan(\omega/2)$. The per-line format is:

  ```
  entity_id r_1 r_2 r_3
  ```

- For '`euler-bunge`', each orientation is described by ϕ_1, θ, ϕ_2, where ϕ_1 is the rotation about the z axis, θ is the rotation about the z' axis, and ϕ_2 is the rotation about the z'' axis, all in degrees). The per-line format is:

  ```
  entity_id phi_1 Phi phi_2
  ```

- For '`euler-kocks`', each orientation is described by Ψ, Θ, ϕ, where Ψ is the rotation about the z axis, Θ is the rotation about the y' axis, and ϕ is the rotation about the z'' axis, all in degrees). The per-line format is:

  ```
  entity_id Psi Theta phi
  ```

- For '`axis-angle`', each orientation is described by t_1, t_2, t_3, ω, where $\mathbf{t}$ is the normalized axis of rotation and ω is the angle of rotation about said axis, in degrees. The per-line format is:

  ```
  entity_id t_1 t_2 t_3 omega
  ```

- For '`quaternion`', each orientation is described by q_0, q_1, q_2, q_3, where $q_0 = \cos(\omega/2)$ and $q_i = t_i \sin(\omega/2)$ for $i = 1, 2, 3$. The per-line format is:

  ```
  entity_id q_0 q_1 q_2 q_3
  ```

2.2.2 External Crystallographic Phase Assignment (Optional, `simulation.phase`)

The embedded grain/phase assignments within the `simulation.msh` may be overridden via an external `simulation.phase` file. This file contains formatting identical to the associated fields in the mesh file and is defined as:

```
$Groups
group_entity
number_of_group_entities
entity_id group
...
$EndGroups
```

where *group_entity* defines the phase assignment method and must always be defined as 'elset', *number_of_group_entities* is the number of grain/phase pairs defined in the field, *entity_id* is a unique 1-indexed identification number, and *group* is an 1-indexed value that defines the phase for a given grain.

2.3 The Fiber Averaging File (Optional, `simulation.fib`)

This file contains the definitions of crystallographic fibers. For each crystallographic fiber, the values of certain simulation variables are averaged (and the standard deviation calculated) for the elements which are oriented such that they belong to each fiber. Additionally, statistics on the elements that belong to each fiber are printed, such as the number of elements, the volume of the collective elements, and the list of elements that belong to a fiber. A full description of the fiber averaging output is discussed later (see Section 3.5 [Fiber Averaging Output], page 33, for a description of this output).

The structure of the `simulation.fib` is as follows:

```
h k l u v w phase ang_tol
...
```

Each line defines a single crystallographic fiber, where *h k l* are integer values for the crystallographic direction, *u v w* are real values for the sample direction, *phase* is an integer that specifies the crystallographic phase (beginning with 1), and *ang_tol* is a real value that defines the angular tolerance for which to accept a crystal as belonging to a fiber. All 8 values describing a fiber are expected to be separated by a space. The values provided for *phase* must correspond to the phase identification numbers defined in the `simulation.config` file.

3 Simulation Output

In this section the output files are described. In general, the output can be broken down into five types: nodal output (variables that are calculated and printed at the finite element nodes), elemental output (variables that are calculated and printed at the finite elements), restart output (which contains all variables necessary to restart a simulation at a specific load step), miscellaneous output (macroscopic forces and various simulation statistics), and fiber averaged output (fiber averaged values of certain variables and other statistics associated with the fiber average output option).

All desired output must be defined in the `simulation.config` file (see Section 2.1.6 [Printing Results], page 18, for a description of the print options). A (small) `post.report` file is systematically printed, which contains information necessary for post-processing with Neper. All output described in this section is *raw* simulation output and can be post-processed into a more human-readable format with Neper's '`-S`' module. Refer to the Neper reference manual, `https://neper.info/docs/neper.pdf`, for a more complete description of the '`-S`' module.

3.1 Nodal Output

Nodal output prints a single variable per finite element node. Raw output is printed on a per-core basis, and the general file name structure is `post.variable.core#`, where '`variable`' is the variable being printed, and '`#`' denotes the ID of the core on which the data is being printed. In general, the file structure is:

```
step_number core_start_dof core_end_dof
node n values
node n+1 values
node n+2 values
...
```

Here, a header line prints for each deformation step, which details the deformation step number, the initial degree of freedom that prints from that core, and the final degree of freedom that prints from that core. Note that the number of *nodes* per-core is one third of the number of degrees of freedoms per-core. For each load step, multiple values associated with a variable are printed per line. When multiple values are printed per line, values are space delimited.

Below, the specific nodal values available for printing are explained in detail.

`post.coo.core*` : `Coordinates`
: In this file, the coordinates are printed. Each nodal coordinate is described in the orthonormal Cartesian sample basis, and one coordinate is printed per line (3 values per line). The components of the coordinates are printed in the order:

  x, y, z

`post.disp.core*` : `Displacements`
: In this file, the displacements are printed. Each nodal displacement is described in the orthonormal Cartesian sample basis, and one displacement is printed per line (3 values per line). The components of the displacements are printed in the order:

  d_x, d_y, d_z

`post.vel.core*` : `Velocities`
: In this file, the velocities are printed. Each nodal velocity is described in the orthonormal Cartesian sample basis, and one velocity is printed per line (3 values per line). The components of the velocities are printed in the order:

  v_x, v_y, v_z

3.2 Elemental Output

Elemental output prints a single variable per finite element. Raw output is printed on a per-core basis, and the general file name structure is `post.variable.core#`, where '`variable`' is the variable being printed, and '`#`' denotes the ID of the core on which the data is being printed. In general, the file structure is:

```
step_number core_start_elt core_end_elt
element n value(s)
element n+1 value(s)
element n+2 value(s)
...
step_number core_start_elt core_end_elt
element n value(s)
element n+1 value(s)
element n+2 value(s)
...
```

Here, a header line prints for each deformation step, which details the deformation step number, the initial element that prints from that core, and the final element that prints from that core. For each load step, either a single value associated with a variable is printed per line (for variables that are printed singularly per element, such as scalars), or multiple values associated with a variable are printed per line (for variables that print multiple values per element, such as tensors). When multiple values are printed per line, values are space delimited.

FEPX calculates elemental quantities at each Gauss quadrature point within the element (15 total). However, only one value is printed – that associated with the quadrature point that falls at the element centroid.

Below, the specific nodal values available for printing are explained in detail.

`post.crss.core*` : Critical Resolved Shear Stress

In this file, the critical resolved shear stress is printed. For the isotropic hardening assumption (see Section 2.1.8 [Optional Input Parameters], page 19), one value is printed per element. For anisotropic hardening assumptions, the critical resolved shear stress is printed for each slip system per element, one line of values per element.

For body centered cubic crystal symmetry, values are printed in the order:
$(01\bar{1})[111]$, $(10\bar{1})[111]$, $(1\bar{1}0)[111]$, $(011)[11\bar{1}]$, $(101)[11\bar{1}]$, $(1\bar{1}0)[11\bar{1}]$,
$(011)[1\bar{1}1]$, $(10\bar{1})[1\bar{1}1]$, $(110)[1\bar{1}1]$, $(01\bar{1})[1\bar{1}\bar{1}]$, $(101)[1\bar{1}\bar{1}]$, $(110)[1\bar{1}\bar{1}]$.

For face centered cubic crystal symmetry, values are printed in the order:
$(111)[01\bar{1}]$, $(111)[10\bar{1}]$, $(111)[1\bar{1}0]$, $(11\bar{1})[011]$, $(11\bar{1})[101]$, $(11\bar{1})[1\bar{1}0]$,
$(1\bar{1}1)[011]$, $(1\bar{1}1)[10\bar{1}]$, $(1\bar{1}1)[110]$, $(1\bar{1}\bar{1})[01\bar{1}]$, $(1\bar{1}\bar{1})[101]$, $(1\bar{1}\bar{1})[110]$.

For hexagonal close packed crystal symmetry, values are printed in the order (corresponding to the 3 basal, 3 prismatic, and 12 pyramidal slip systems):
$(0001)[2\bar{1}\bar{1}0]$, $(0001)[\bar{1}2\bar{1}0]$, $(0001)[\bar{1}\bar{1}20]$, $(01\bar{1}0)[2\bar{1}\bar{1}0]$, $(\bar{1}010)[\bar{1}2\bar{1}0]$, $(1\bar{1}00)[\bar{1}\bar{1}20]$,
$(10\bar{1}1)[\bar{2}113]$, $(10\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[1\bar{2}13]$, $(\bar{1}101)[1\bar{2}13]$, $(\bar{1}101)[2\bar{1}\bar{1}3]$,
$(\bar{1}011)[2\bar{1}\bar{1}3]$, $(\bar{1}011)[11\bar{2}3]$, $(0\bar{1}11)[11\bar{2}3]$, $(0\bar{1}11)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{2}113]$.

`post.defrate.core*` : Deformation Rate Tensor

In this file, the deformation rate tensor is printed. Each tensor, $\mathbf{D}$, is printed in the sample basis. The independent components are printed, one tensor per line (6 values per line). The components, D_{ij}, are printed in the order:

D_{11}, D_{22}, D_{33}, D_{23}, D_{13}, D_{12}

`post.defrate-eq.core*` : Equivalent Deformation Rate

In this file, the equivalent deformation rate is printed. One scalar value is printed per element. The equivalent deformation rate, D, is calculated based on the deformation rate tensor, $\mathbf{D}$, via the tensor inner product:

$D = \sqrt{\frac{2}{3}\mathbf{D} : \mathbf{D}}$

`post.defrate-pl.core*` : Plastic Deformation Rate Tensor

In this file, the deviatoric plastic deformation rate tensor is printed. Each tensor, $\mathbf{D}^p$, is printed in the sample basis. The independent components are printed, one tensor per line (6 values per line). The components, D^p_{ij}, are printed in the order:

D^p_{11}, D^p_{22}, D^p_{33}, D^p_{23}, D^p_{13}, D^p_{12}

`post.defrate-pl-eq.core*` : Equivalent Plastic Deformation Rate

In this file, the equivalent plastic deformation rate is printed. One scalar value is printed per element. The equivalent plastic deformation rate, D^p, is calculated based on the plastic deformation rate tensor, $\mathbf{D}^p$, via the tensor inner product:

$D^p = \sqrt{\frac{2}{3}\mathbf{D}^p : \mathbf{D}^p}$

`post.elt-vol.core*` : Elemental Volume

In this file, the elemental volume is printed. One scalar value is printed per element. The elemental volume is calculated as the Gaussian integration of the determinant of the Jacobian matrix:

$V_{el} = \sum_{i=1}^{n_{qp}} det(J_i) w_i$

`post.ori.core*` : Crystallographic Orientation

In this file, the crystallographic orientation is printed. Depending on the orientation parameterization used as input, the orientation values may range from 3 values per element (when using Rodrigues vector, Euler-Bunge angles and Euler-Kocks angles parameterizations) or 4 values per element (when using axis-angle or quaternion parameterizations). One orientation is printed per line (3 or 4 values per line).

For Rodrigues: r_1, r_2, r_3, where the Rodrigues vector is $\mathbf{r} = \mathbf{t}\tan(\omega/2)$.

For Euler-Bunge: ϕ_1, θ, ϕ_2 (where ϕ_1 is the rotation about the z axis, θ is the rotation about the x' axis, and ϕ_2 is the rotation about the z'' axis, all in degrees).

For Euler-Kocks: Ψ, Θ, ϕ (where Ψ is the rotation about the z axis, Θ is the rotation about the y' axis, and ϕ is the rotation about the z'' axis, all in degrees).

For axis-angle: t_1, t_2, t_3, ω (where $\mathbf{t}$ is the normalized axis of rotation and ω is the angle of rotation about said axis, in degrees).

For quaternion: q_0, q_1, q_2, q_3, where $q_0 = \cos(\omega/2)$ and $q_i = t_i \sin(\omega/2)$ for $i = 1, 2, 3$.

`post.slip.core*` : Slip System Shear

In this file, the accumulated slip system shear is printed. The slip system shear is printed for each slip system per element, one line of values per element.

For body centered cubic crystal symmetry, values are printed in the order:

$(01\bar{1})[111]$, $(10\bar{1})[111]$, $(1\bar{1}0)[111]$, $(011)[11\bar{1}]$, $(101)[11\bar{1}]$, $(1\bar{1}0)[11\bar{1}]$,
$(011)[1\bar{1}1]$, $(10\bar{1})[1\bar{1}1]$, $(110)[1\bar{1}1]$, $(01\bar{1})[1\bar{1}\bar{1}]$, $(101)[1\bar{1}\bar{1}]$, $(110)[1\bar{1}\bar{1}]$.

For face centered cubic crystal symmetry, values are printed in the order:

$(111)[01\bar{1}]$, $(111)[10\bar{1}]$, $(111)[1\bar{1}0]$, $(11\bar{1})[011]$, $(11\bar{1})[101]$, $(11\bar{1})[1\bar{1}0]$,
$(1\bar{1}1)[011]$, $(1\bar{1}1)[10\bar{1}]$, $(1\bar{1}1)[110]$, $(1\bar{1}\bar{1})[01\bar{1}]$, $(1\bar{1}\bar{1})[101]$, $(1\bar{1}\bar{1})[110]$.

For hexagonal close packed crystal symmetry, values are printed in the order (corresponding to the 3 basal, 3 prismatic, and 12 pyramidal slip systems):

$(0001)[2\bar{1}\bar{1}0]$, $(0001)[\bar{1}2\bar{1}0]$, $(0001)[\bar{1}\bar{1}20]$, $(01\bar{1}0)[2\bar{1}\bar{1}0]$, $(\bar{1}010)[\bar{1}2\bar{1}0]$, $(1\bar{1}00)[\bar{1}\bar{1}20]$,
$(10\bar{1}1)[\bar{2}113]$, $(10\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[1\bar{2}13]$, $(\bar{1}101)[1\bar{2}13]$, $(\bar{1}101)[2\bar{1}\bar{1}3]$,
$(\bar{1}011)[2\bar{1}\bar{1}3]$, $(\bar{1}011)[11\bar{2}3]$, $(0\bar{1}11)[11\bar{2}3]$, $(0\bar{1}11)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{2}113]$.

`post.sliprate.core*` : Slip System Shear Rate

In this file, the slip system shear rate is printed. The slip system shear rate is printed for each slip system per element, one line of values per element.

For body centered cubic crystal symmetry, values are printed in the order:

$(01\bar{1})[111]$, $(10\bar{1})[111]$, $(1\bar{1}0)[111]$, $(011)[11\bar{1}]$, $(101)[11\bar{1}]$, $(1\bar{1}0)[11\bar{1}]$,
$(011)[1\bar{1}1]$, $(10\bar{1})[1\bar{1}1]$, $(110)[1\bar{1}1]$, $(01\bar{1})[1\bar{1}\bar{1}]$, $(101)[1\bar{1}\bar{1}]$, $(110)[1\bar{1}\bar{1}]$.

For face centered cubic crystal symmetry, values are printed in the order:

$(111)[01\bar{1}]$, $(111)[10\bar{1}]$, $(111)[1\bar{1}0]$, $(11\bar{1})[011]$, $(11\bar{1})[101]$, $(11\bar{1})[1\bar{1}0]$,
$(1\bar{1}1)[011]$, $(1\bar{1}1)[10\bar{1}]$, $(1\bar{1}1)[110]$, $(1\bar{1}\bar{1})[01\bar{1}]$, $(1\bar{1}\bar{1})[101]$, $(1\bar{1}\bar{1})[110]$.

For hexagonal close packed crystal symmetry, values are printed in the order (corresponding to the 3 basal, 3 prismatic, and 12 pyramidal slip systems):

$(0001)[2\bar{1}\bar{1}0]$, $(0001)[\bar{1}2\bar{1}0]$, $(0001)[\bar{1}\bar{1}20]$, $(01\bar{1}0)[2\bar{1}\bar{1}0]$, $(\bar{1}010)[\bar{1}2\bar{1}0]$, $(1\bar{1}00)[\bar{1}\bar{1}20]$,
$(10\bar{1}1)[\bar{2}113]$, $(10\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[1\bar{2}13]$, $(\bar{1}101)[1\bar{2}13]$, $(\bar{1}101)[2\bar{1}\bar{1}3]$,
$(\bar{1}011)[2\bar{1}\bar{1}3]$, $(\bar{1}011)[11\bar{2}3]$, $(0\bar{1}11)[11\bar{2}3]$, $(0\bar{1}11)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{2}113]$.

`post.spinrate.core*` : Plastic Spin Rate Tensor

In this file, the skew-symmetric plastic spin rate tensor is printed. Each tensor, $\mathbf{W}^p$, is printed in the sample basis. The independent components are printed, one tensor per line (3 values per line). The components, W^p_{ij}, are printed in the order:

$W^p_{12},\ W^p_{13},\ W^p_{23}$

`post.strain.core*` : Total Strain Tensor

In this file, the total strain tensor is printed. Each tensor, $\mathbf{E}$, is printed in the sample basis. The independent components are printed, one tensor per line (6 values per line). The components, E_{ij}, are printed in the order:

$E_{11},\ E_{22},\ E_{33},\ E_{23},\ E_{13},\ E_{12}$

`post.strain-eq.core*` : Equivalent Total Strain

In this file, the equivalent total strain is printed. One scalar value is printed per element. The equivalent total strain, E, is calculated based on the deviatoric portion of the total strain tensor, $\mathbf{E}'$. via the tensor inner product:

$E = \sqrt{\frac{2}{3}\mathbf{E}' : \mathbf{E}'}$

`post.strain-el.core*` : Elastic Strain Tensor

In this file, the elastic strain tensor is printed. Each tensor, $\mathbf{E^e}$, is printed in the sample basis. The independent components are printed, one tensor per line (6 values per line). The components, E^e_{ij}, are printed in the order:

$E^e_{11},\ E^e_{22},\ E^e_{33},\ E^e_{23},\ E^e_{13},\ E^e_{12}$

`post.strain-el-eq.core*` : Equivalent Elastic Strain

In this file, the equivalent elastic strain is printed. One scalar value is printed per element. The equivalent elastic strain, E^e, is calculated based on the deviatoric portion of the elastic strain tensor, $\mathbf{E}^{e\prime}$, via the tensor inner product:

$E^e = \sqrt{\frac{2}{3}\mathbf{E}^{e\prime} : \mathbf{E}^{e\prime}}$

`post.strain-pl.core*` : Plastic Strain Tensor

In this file, the plastic strain tensor is printed. Each tensor, $\mathbf{E^p}$, is printed in the sample basis. The independent components are printed, one tensor per line (6 values per line). The components, E^p_{ij}, are printed in the order:

$E^p_{11},\ E^p_{22},\ E^p_{33},\ E^p_{23},\ E^p_{13},\ E^p_{12}$

`post.strain-pl-eq.core*` : Equivalent Plastic Strain

In this file, the equivalent plastic strain is printed. One scalar value is printed per element. The equivalent plastic strain, E^p, is calculated based on the plastic strain tensor, $\mathbf{E}^p$, via the tensor inner product:

$E^p = \sqrt{\frac{2}{3}\mathbf{E}^p : \mathbf{E}^p}$

`post.stress.core*` : Stress Tensor

In this file, the symmetric stress tensor is printed. Each tensor, σ, is printed in the sample basis. The independent components are printed, one tensor per line (6 values per line). The components, σ_{ij}, are printed in the order:

$\sigma_{11},\ \sigma_{22},\ \sigma_{33},\ \sigma_{23},\ \sigma_{13},\ \sigma_{12}$

`post.stress-eq.core*` : Equivalent Stress

In this file, the equivalent stress is printed. One scalar value is printed per element. The equivalent stress, σ, is calculated based on the deviatoric portion of the stress tensor, σ', via the tensor inner product:

$\sigma = \sqrt{\frac{3}{2}\sigma' : \sigma'}$

`post.velgrad.core*` : Velocity Gradient Tensor

In this file, the velocity gradient tensor is printed. Each tensor, $\mathbf{L}$, is printed in the sample basis. One tensor is printed per line (9 values per line). The components, L_{ij}, are printed in the order:

$L_{11},\ L_{12},\ L_{13},\ L_{21},\ L_{22},\ L_{23},\ L_{31},\ L_{32},\ L_{33}$

`post.work.core*` : Work

In this file, the work is printed. One scalar value is printed per element. The work is calculated as the time integration of the tensor inner product of the deformation rate tensor and the Cauchy stress tensor:

$W = \int (\sigma : \mathbf{D}) \Delta t$

`post.work-pl.core*` : Plastic Work

In this file, the plastic work is printed. One scalar value is printed per element. The plastic work is calculated as the time integration of the tensor inner product of the plastic deformation rate tensor and the deviatoric portion of the Cauchy stress tensor:

$W^p = \int (\sigma' : \mathbf{D}^p) \Delta t$

`post.workrate.core*` : Work Rate

In this file, the work rate is printed. One scalar value is printed per element. The work rate is calculated as the tensor inner product of the deformation rate tensor and the Cauchy stress tensor:

$\dot{W} = \sigma : \mathbf{D}$

`post.workrate-pl.core*` : Plastic Work Rate

In this file, the plastic work rate is printed. One scalar value is printed per element. The plastic work rate is calculated as the tensor inner product of the plastic deformation rate tensor and the deviatoric portion of the Cauchy stress tensor:

$\dot{W}^p = \sigma' : \mathbf{D}^p$

3.3 Restart Output

If the '`print restart`' command is present in the `simulation.config` file, a set of additional restart files will be generated from the simulation. These files are written at the end of each prescribed step and contain necessary information to restart a given simulation (see

Section 4.2 [Restarting a Simulation], page 38, for information on how to restart a simulation). Two types of restart files are generated, a control file, '`rstN.control`', and per-core field files, '`rstN.field.core*`' (where '*N*' indicates which simulation the files describe, 0 indexing). Both file types are unformatted (or binary) files and are generally unmodifiable. The structures of the data stored within both files for the various deformation modes follow.

3.3.1 Uniaxial Restart Control

The `rstN.control` file for uniaxial loading modes contains the following data in the given order:

```
current_step integer
previous_load_array real_x real_y real_z
step_complete_flag logical
previous_timestep_value real
current_incr integer
current_time real
surface_1_loads real_x real_y real_z
...
surface_6_loads real_x real_y real_z
previous_prescribed_load real
current_surface_areas real_surf_1 ... real_surf_6
initial_surface_areas real_surf_1 ... real_surf_6
```

3.3.2 Multiaxial CSR Restart Control

The `rstN.control` file for multiaxial constant strain rate loading modes contains the following data in the given order:

```
current_step integer
current_load_array real_x real_y real_z
previous_load_array real_x real_y real_z
step_complete_flag logical
previous_timestep_value real
current_incr integer
current_time real
surface_1_loads real_x real_y real_z
...
surface_6_loads real_x real_y real_z
current_surface_areas real_surf_1 ... real_surf_6
initial_surface_areas real_surf_1 ... real_surf_6
current_mesh_lengths real_length_x real_length_y real_length_z
initial_mesh_lengths real_length_x real_length_y real_length_z
current_control_velocity real_vel_x real_vel_y real_vel_z
s_pert_mag real
t_pert_mag real
```

3.3.3 Multiaxial CLR Restart Control

The `rstN.control` file for multiaxial constant load rate loading modes contains the following data in the given order:

```
current_step integer
current_load_array real_x real_y real_z
previous_load_array real_x real_y real_z
first_incr_in_step logical
```

```
current_incr integer
current_time real
surface_1_loads real_x real_y real_z
...
surface_6_loads real_x real_y real_z
current_surface_areas real_surf_1 ... real_surf_6
initial_surface_areas real_surf_1 ... real_surf_6
current_mesh_lengths real_length_x real_length_y real_length_z
initial_mesh_lengths real_length_x real_length_y real_length_z
current_control_velocity real_vel_x real_vel_y real_vel_z
previous_control_action integer
current_control_action integer
initial_load_dwell_velocity real_vel_x real_vel_y real_vel_z
initial_unload_dwell_velocity real_vel_x real_vel_y real_vel_z
```

3.3.4 Restart Field Data

All loading modes also write field data on a per-core basis to `rstN.field.core*` files. These files contain the necessary field variable information in order to spatially define the total state of the virtual sample at the time of printing. The following field data arrays are written to the files in the given order:

```
coords real
velocity real

c0_angs real
c_angs real
rstar real
rstar_n real
wts real
crss real
crss_n real

gela_kk_bar real
gsig_vec_n real
pela_kk_bar real
psig_vec_n real
e_elas_kk_bar real
sig_vec_n real

eqstrain real
eqplstrain real
gamma real

el_work_n real
el_workp_n real
el_work_rate_n real
el_workp_rate_n real
```

3.4 Miscellaneous Output

In addition to nodal and elemental variable printing, miscellaneous output is available for printing and include simulation convergence data, surface-integrated forcing data, and a simulation report file. The optional input commands and output file formats are described in this section.

3.4.1 Convergence Statistics Output

If the '`print convergence`' command is present in the `simulation.config` file, various convergence statistics for the performed simulation will be output with convergence values provided at each `increment`. This `post.conv` file is tabulated with the given structure:

```
incr iter nr r_norm rx_norm f_norm delu_norm delux_norm u_norm cg_iter
```

where *incr* is the total increment value being printed, *iter* is a sub-increment iteration, *nr* is a boolean that notifies if the given *iter* was a Newton-Raphson iteration, *r_norm* is a residual norm, *rx_norm* is the maximum absolute value of *r_norm*, *f_norm* is a force norm, *delu_norm* is the change in velocity norm, *delux_norm* is the maximum absolute value of *delu_norm*, *u_norm* is the velocity norm, and *cg_iter* is the number of iterations the conjugate gradient solver performed. All norms are computed as l^2-norms or the square root of the inner product of a vector by itself.

3.4.2 Surface Forces Output

If the '`print forces`' command is present in the `simulation.config` file, loads for all surfaces in the performed simulation will be output with load values provided at each `increment`. The `post.force.*` file names are constructed via the defined `faset_label` strings in the `simulation.msh` file. The default `faset_label` order is '`x0, x1, y0, y1, z0, z1`' which defines the six orthogonal and planar surfaces that bound a domain. For example, '`x0`' refers to the face where the nodal coordinate component values in the x-direction are minimum and the file `post.force.x0` would contain the surface-integrated forces on this face. These files are generally tabulated with the given structure:

```
step incr force_x force_y force_z surf_area current_time
```

where *step* is the prescribed load step, *incr* is the total increment value being printed, *force_x* is the surface-integrated force in the x-direction, *force_y* is the surface-integrated force in the y-direction, *force_z* is the surface-integrated force in the z-direction, *surf_area* is the current surface area of the given face in [area], and *current_time* is the total simulated time at the time of printing.

If multiaxial loading is utilized, an additional *length* column will be appended to the right of *current_time*. The *length* column contains the maximal coordinate values of the domain and these values are stored in their associated face files. For example, the maximal mesh coordinate value in the x-direction is stored in the `post.force.x0` and `post.force.x1` files accordingly.

3.4.3 Simulation Report File

The `post.report` file is always printed for a simulation. The report file is for utilization with Neper and contains the following information:

```
number_of_nodes value
number_of_elements value
number_of_partitions value
number_of_elements_byparition part1_num_elems ... partN_num_elems
number_of_nodes_byparition part1_num_nodes ... partN_num_nodes
number_of_slip_systems num_slip_systems_for_crystal_type
```

```
orientation_definition orientation_descriptor:orientation_convention
results_nodes nodal_output_files
results_elements elemental_output_files
number_of_steps number_of_completed_steps
```

3.5 Fiber Averaging Output

The optional fiber averaging routine (see Section 2.3 [The Fiber Averaging File (Optional)], page 23) has the ability to gather the average and standard deviation of certain simulation output variables for elements which belong to the user-defined crystallographic fibers. Currently, the output available from the fiber averaging routine is (note that all output is printed by default if fiber averaging is activated):

`post.fib.crss` : Critical Resolved Shear Stress

In this file, the fiber averaged critical resolved shear stress is printed. For either an isotropic or anisotropic hardening assumption, the output file is structured as follows:

```
step_number num_fibers
crss1_fib1_avg crss2_fib1_avg ... crss1_fib1_std crss2_fib1_std ...
crss1_fib2_avg crss2_fib2_avg ... crss1_fib2_std crss2_fib2_std ...
...
```

where *`crss*_fib*_avg`* refers to the average critical resolved shear stress over all of the elements that belong to that specific crystallographic fiber on a given slip system, and *`crss*_fib*_std`* refers to the standard deviation of the critical resolved shear stress over all of the elements that belong to that specific crystallographic fiber on a given slip system. One set of values is printed per fiber, one fiber per line. Each block is printed per '`step`'. If a multiphase material is fiber-averaged, the number of *`fib*`* columns printed for a given fiber is determined by the maximal number of slip systems for the considered material phase. As such, the number of columns in this file may not necessarily be fixed throughout.

For body centered cubic crystal symmetry, values are printed in the order:

$(01\bar{1})[111]$, $(10\bar{1})[111]$, $(1\bar{1}0)[111]$, $(011)[11\bar{1}]$, $(101)[11\bar{1}]$, $(1\bar{1}0)[11\bar{1}]$,
$(011)[1\bar{1}1]$, $(10\bar{1})[1\bar{1}1]$, $(110)[1\bar{1}1]$, $(01\bar{1})[1\bar{1}\bar{1}]$, $(101)[1\bar{1}\bar{1}]$, $(110)[1\bar{1}\bar{1}]$.

For face centered cubic crystal symmetry, values are printed in the order:

$(111)[01\bar{1}]$, $(111)[10\bar{1}]$, $(111)[1\bar{1}0]$, $(11\bar{1})[011]$, $(11\bar{1})[101]$, $(11\bar{1})[1\bar{1}0]$,
$(1\bar{1}1)[011]$, $(1\bar{1}1)[10\bar{1}]$, $(1\bar{1}1)[110]$, $(1\bar{1}\bar{1})[01\bar{1}]$, $(1\bar{1}\bar{1})[101]$, $(1\bar{1}\bar{1})[110]$.

For hexagonal close packed crystal symmetry, values are printed in the order (corresponding to the 3 basal, 3 prismatic, and 12 pyramidal slip systems):

$(0001)[2\bar{1}\bar{1}0]$, $(0001)[\bar{1}2\bar{1}0]$, $(0001)[\bar{1}\bar{1}20]$, $(01\bar{1}0)[2\bar{1}\bar{1}0]$, $(\bar{1}010)[\bar{1}2\bar{1}0]$, $(1\bar{1}00)[\bar{1}\bar{1}20]$,
$(10\bar{1}1)[\bar{2}113]$, $(10\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[\bar{1}\bar{1}23]$, $(01\bar{1}1)[1\bar{2}13]$, $(\bar{1}101)[1\bar{2}13]$, $(\bar{1}101)[2\bar{1}\bar{1}3]$,
$(\bar{1}011)[2\bar{1}\bar{1}3]$, $(\bar{1}011)[11\bar{2}3]$, $(0\bar{1}11)[11\bar{2}3]$, $(0\bar{1}11)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{1}2\bar{1}3]$, $(1\bar{1}01)[\bar{2}113]$.

`post.fib.defrate-pl-eq`: Equivalent Plastic Deformation Rate

In this file, the fiber averaged equivalent plastic deformation rate is printed. The output file is structured as follows:

```
step_number num_fibers
defrate-pl-eq_fib1_avg defrate-pl-eq_fib1_std
defrate-pl-eq_fib2_avg defrate-pl-eq_fib2_std
...
```

where *defrate-pl-eq_fib*_avg* refers to the average equivalent plastic deformation rate over all of the elements that belong to that specific crystallographic fiber, and *defrate-pl-eq_fib*_std* refers to the standard deviation of the equivalent plastic deformation rate over all of the elements that belong to that specific crystallographic fiber. One set of values is printed per fiber, one fiber per line. Each block is printed per '`step`'.

`post.fib.sliprate-sum`: Sum of the Slip System Shear Rates
In this file, the fiber averaged sum of the slip system shear rates rate is printed. The output file is structured as follows:

```
step_number num_fibers
sliprate-sum_fib1_avg sliprate-sum_fib1_std
sliprate-sum_fib2_avg sliprate-sum_fib2_std
...
```

where *sliprate-sum_fib*_avg* refers to the average sum of the slip system shear rates over all of the elements that belong to that specific crystallographic fiber, and *sliprate-sum_fib*_std* refers to the standard deviation of the sum of the slip system shear rates over all of the elements that belong to that specific crystallographic fiber. One set of values is printed per fiber, one fiber per line. Each block is printed per '`step`'.

`post.fib.strain-el-lat`: Elastic Lattice Strains
In this file, the fiber averaged elastic lattice strain is printed. The output file is structured as follows:

```
step_number num_fibers
strain-el-lat_fib1_avg strain-el-lat_fib1_std
strain-el-lat_fib2_avg strain-el-lat_fib2_std
...
```

where *strain-el-lat_fib*_avg* refers to the average elastic lattice strain over all of the elements that belong to that specific crystallographic fiber, and *strain-el-lat_fib*_std* refers to the standard deviation of the elastic lattice strain over all of the elements that belong to that specific crystallographic fiber. One set of values is printed per fiber, one fiber per line. Each block is printed per '`step`'.

`post.fib.elt-stats`: Element Statistics
In this file, the element statistics for each fiber are printed. The output file is structured as follows:

```
step_number num_fibers
num_elt_fib1 vol_elt_fib1
num_elt_fib2 vol_elt_fib2
...
```

where *num_elt_fib** refers to the number of elements that belong to that specific crystallographic fiber, and *vol_elt_fib** refers to the total volume of all of the elements that belong to that specific crystallographic fiber. One set of values is printed per fiber, one fiber per line. Each block is printed per '`step`'.

`post.fib.elt.core*`: Lists of Elements Belonging to Crystallographic Fibers
In this file, the list of local elements (that is, elements on a particular core) belonging to each crystallographic fiber are printed. The output file is structured as follows:

```
step_number fib_id num_fib_elts core_start_elt core_end_elt
```

```
elt_id_1
elt_id_2
...
```

where *fib_id* refers to the current fiber whose elements are being printed, *num_fib_elts* refers to the number of elements on the local core that meet the diffraction conditions, and *elt_id_** refer to 1-indexed element IDs of elements that belong to the current fiber. One list of values is printed per fiber, *num_fiber* lists are printed per block. Each block is printed per '`step`'.

4 Running a Simulation

First, a simulation may be run serially – that is, on a single core – by executing the binary from the terminal,

```
$ fepx
```

However, simulations will generally require more computational resources to run in reasonable time, which can be done using a parallel computer architecture. A simulation may be run in parallel utilizing MPI,

```
$ mpirun -np N fepx
```

where N refers to the number of MPI processes desired. Note that your local installation of MPI may not utilize '`mpirun`' and instead an alternative MPI command may be required.

4.1 Submitting FEPX to a Job Scheduling Program

Performing simulations on high performance computing clusters typically requires interfacing with a job scheduling program. These programs have a number of directives that are too numerous to define here. For sake of illustration, however, generic submission scripts for the Slurm and Torque job scheduling programs are provided. The generation of these scripts is highly dependent on the local configuration, and you are encouraged to work with the system administrator of your cluster if you are unsure on how to properly build a submission script. Both example scripts below are designed to submit a parallel job of FEPX to 4 nodes with 16 cores per node (i.e., 64 total tasks) to the '`main`' queue.

A generic submission script, `runslurm.sh`, for submitting to a Slurm scheduler:

```
#!/bin/bash
#SBATCH -J fepxjob
#SBATCH -q main
#SBATCH --ntasks 64
#SBATCH --ntasks-per-node 16
#SBATCH -o output.%A
#SBATCH -e errors.%A

srun --mpi=pmi2 fepx
```

would be run by entering `sbatch runslurm.sh` into the terminal from within the simulation directory.

A generic submission script, `runtorque.sh`, for submitting to a Torque scheduler:

```
#!/bin/bash
#PBS -N fepxjob
#PBS -q main
#PBS -l nodes=4:ppn=16
#PBS -k oe
#PBS -j oe

# Change to the current working directory
cd $PBS_O_WORKDIR
```

```
# Calculate the total number of cores requested
NP=‘cat $PBS_NODEFILE | wc -l‘

mpirun -np ${NP} fepx
```

would be run by entering `qsub runtorque.sh` into the terminal from within the simulation directory.

4.2 Restarting a Simulation

A simulation may be restarted only if the restart files were printed as simulation output on the previous run (see Section 3.3 [Restart Output], page 29). Printing restart files outputs a single ‘`rstN.control`’ file and a ‘`rstN.field.core#`’ file for each individual core, where ‘*N*’ refers to the restart ID (0 indexing), and ‘`#`’ denotes the ID of the core on which the data is being printed. These restart files must be included in the simulation directory along with the configuration file, the mesh file, and any external files included with the simulation.

A simulation may be restarted by adding the following line to the `simulation.config` file:

```
restart on
```

When a simulation restarts, it will attempt to find the simulation restart files with the highest index, *N*. It will write output variable data to a new set of files, ‘`post.var.rstN+1.core*`’, where ‘*var*’ is the requested output variable name and ‘`rstN+1`’ is the restart label applied to all new output variable files. Likewise, if restart files are again printed, their index will increase to *N*+2 (the previous restart’s files will not be overwritten).

A simulation restart must be performed with the same number of cores that were used to run the original simulation. Restart files are always written at the end of a successful step and not at individual increments.

Each restarted simulation is considered a new simulation, albeit with initialized field variables as written in the restart files. Step and increment indices, as well as the simulation time, are all reset to 0.

When restarting a simulation, the prescribed deformation history (see Section 2.1.2 [Deformation History], page 12) should include only additional steps. The restarted simulation will attempt to follow the entire deformation history as prescribed in the `simulation.config` file, and will not consider steps that were completed in the initial simulation.

5 Example Simulations

Several example simulations come pre-packaged to get you started with running simulations.These examples are reference cases to show how a simulation should be built and how FEPX can interface with Neper in order to prepare mesh files as well as post-process a simulation directory. All examples contain the necessary configuration (`simulation.config`) and mesh (`simulation.msh`) files, along with a shell script to generate the mesh file directly from Neper. In order to run the provided shell scripts, you must have a configured installation of Neper present on your system. In the following, visualizations of the undeformed and deformed mesh are generated with Neper while graphs are generated with Gnuplot.

All examples can be run either in serial or parallel (see Chapter 4 [Running a Simulation], page 37), but the included scripts are pre-set to run in parallel on 4 cores, and parallel execution with OpenMPI via `mpirun` is assumed.

A polycrystal containing 50 grains is generated via Voronoi tessellation for all examples. Each cell in the tessellation represents a discrete grain in the domain and all grains are volumetrically discretized into finite elements. Visualizations of the tessellated domain (morphology) and the associated finite element mesh are in Figure 5.1. Length units are assumed to be [mm], thus, all pressure units assumed to be [MPa] for the simulation (including input parameters).

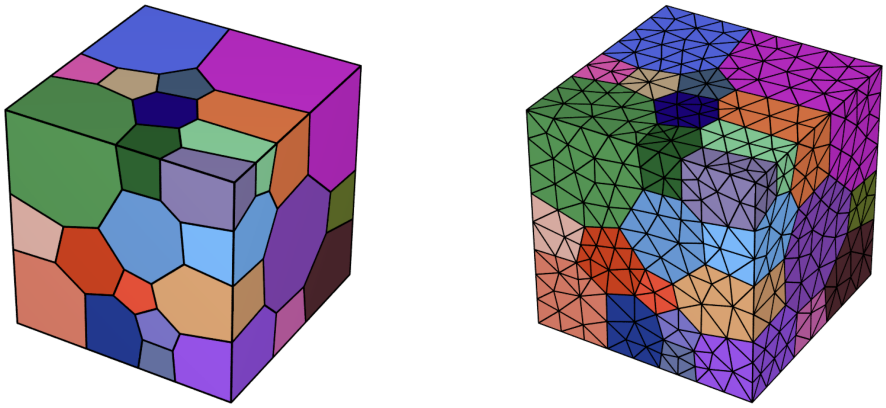

Figure 5.1: (left) 50 grain polycrystal used in all examples, colored by grain id, and (right) its finite element mesh.

5.1 Uniaxial Control (`examples/1_uniaxial`)

This example covers the uniaxial tensile deformation of an FCC polycrystal by strain targeting to 2% axial strain. Material parameters are of a generic a copper alloy, and are provided in Table 5.1 and Table 5.2. Two steps are defined with a variable number of increments for each step. Grip boundary conditions are applied to the sample with the primary loading face set as '`z1`' loaded in the `+`Z direction. Elemental stress and elastic strain tensors are output, along with the nodal coordinates and surface-integrated forces. Illustrations of the stress and elastic strain fields are provided in Figure 5.2. Material parameters are defined as:

Phase	**Type**	C_{11}	C_{12}	C_{44}
α	FCC	245.0×10^3	155.0×10^3	62.50×10^3

Table 5.1: Single crystal elastic constants.

Phase	m	$\dot{\gamma}_0$	h_0	g_0	g_{s0}	n
α	0.05	1.0	200.0	210.0	330.0	1.0

Table 5.2: Initial slip system strengths and other plasticity parameters

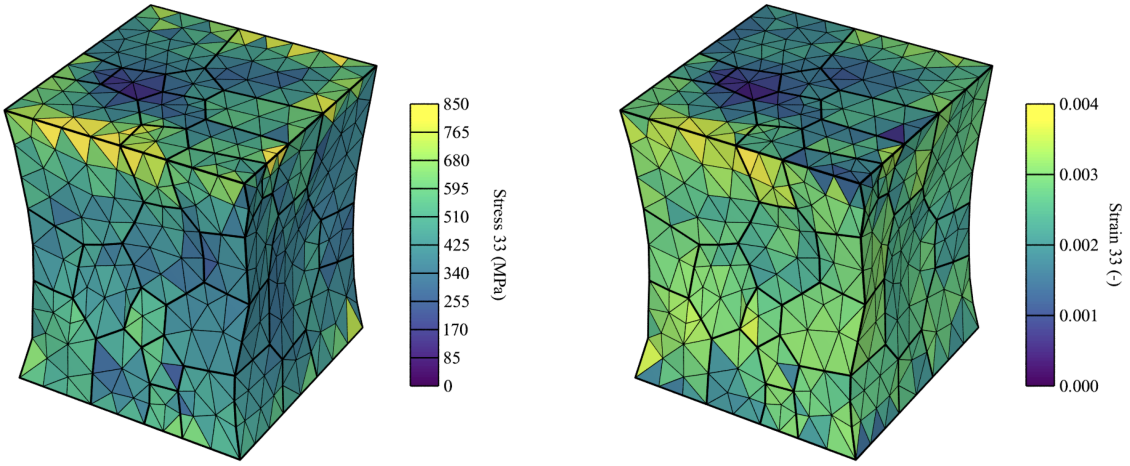

Figure 5.2: Deformed sample at 2% axial strain (deformation field is exaggerated 10x for illustrative purposes), colored by (left) stress (σ_{33}), and (right) elastic strain (ϵ_{33}).

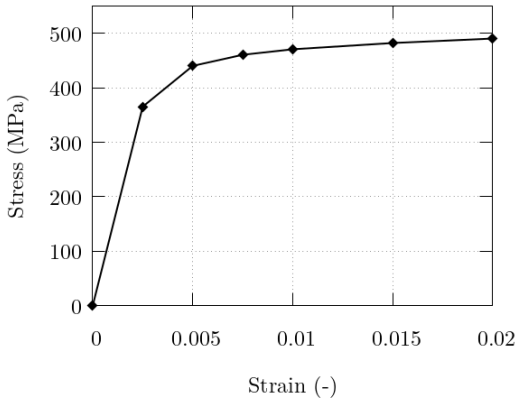

Figure 5.3: Macroscopic stress-strain curve recovered from the surface-integrated forces in `post.force.z1`. Markers denote values output at each increment.

5.2 Multiaxial Control with Constant Strain Rate (`examples/2_triaxCSR`)

This example covers the biaxial deformation of an HCP polycrystal at a constant strain rate. Loads are applied normal to the surface, maintaining proportional macroscopic load ratios of -1:0:1 for the X:Y:Z directions, respectively. Load tolerance options are prescribed and latent hardening is enabled, as well as saturation strength evolution. The primary loading direction is set to be in the X direction, and the strain rate is doubled on the second step. Elemental values for the equivalent plastic strain and plastic work are output, along with the nodal coordinates and surface-integrated forces. Material parameters are those for the α-phase of Ti-6Al-4V and are provided in Table 5.3, Table 5.4 and Table 5.5. The latent parameters are input values to the hardening interaction matrix [21]. Illustrations of the results are provided in Figure 5.4 and Figure 5.5. Material parameters are defined as:

Phase	**Type**	C_{11}	C_{12}	C_{13}	C_{44}
α	HCP	1.6966×10^5	0.8866×10^5	0.6166×10^5	0.4250×10^5

Table 5.3: Single crystal elastic constants

Phase	m	$\dot{\gamma}_0$	h_0	g_{s0}	m'	$\dot{\gamma}_s$	n	c/a
α	0.010	1.0	190.0	530.0	1.1	1.0	1.0	1.587

Table 5.4: Crystallographic slip (plasticity) parameters

Phase	$g_{0,basal}$	$g_{0,prismatic}$	$g_{0,pyramidal}$	h_{diag}	h_1–h_7
α	390.0	468.0	663.0	1.0	1.4

Table 5.5: Initial slip system strengths and hardening parameters

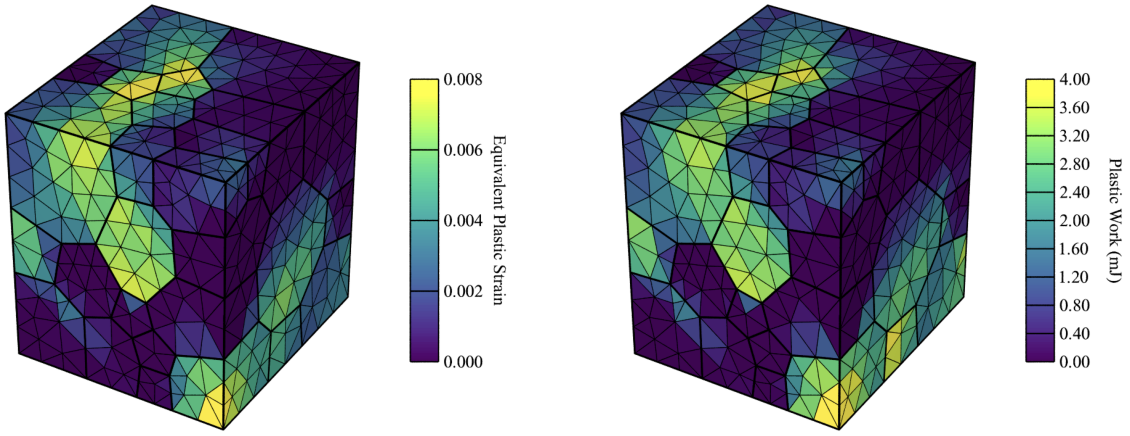

Figure 5.4: Deformed sample at the end of the second load step (deformation field is exaggerated 10x for illustrative purposes), colored by (left) plastic work (W^p) and (right) equivalent plastic strain ($\bar{\epsilon}^P$). Note that, unlike the deformed sample in Figure 5.2, a multiaxial simulation will maintain the orthogonal, planar surfaces throughout the simulation.

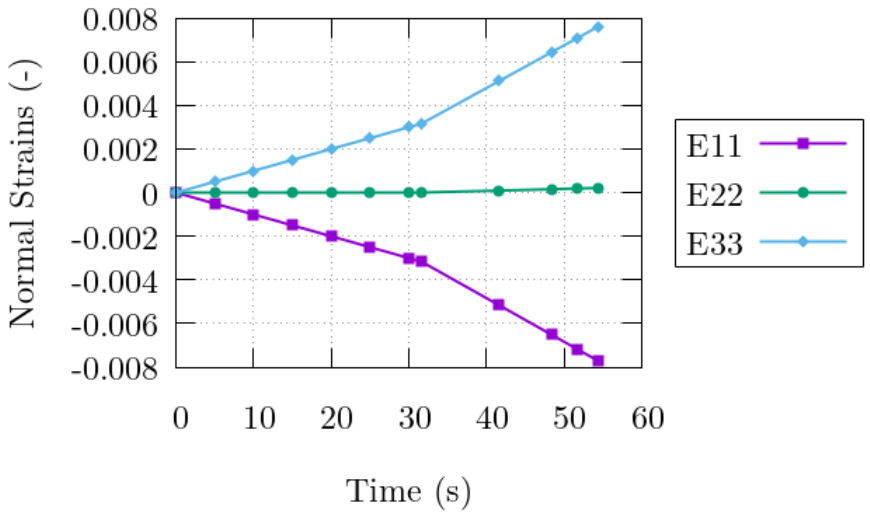

Figure 5.5: Evolution of the macroscopic normal strains. Note the strain rate increase corresponding to the strain-rate jump defined for step 2.

5.3 Multiaxial Control with Constant Load Rate (`examples/3_triaxCLR`)

This example covers the triaxial deformation of a dual phase FCC/BCC polycrystal (phase map shown on Figure 5.6(left)) at a constant load rate followed by a dwell episode and subsequent unloading. Loads are applied normal to the surface, maintaining proportional macroscopic load

ratios of -0.375:-0.625:1 for the X:Y:Z directions, respectively. Load tolerance options are prescribed. The primary loading direction is set to be in the Z direction, and a dwell episode is initiated on the second step. Elemental critical resolved shear stresses and equivalent strains are output, along with the nodal coordinates and surface-integrated forces. Material parameters are those for the austenitic (γ) and ferritic (α) phases of an LDX-2101 steel, and are provided in Table 5.6, Table 5.7 and Table 5.10. Illustrations of the results are provided in Figure 5.6 and Figure 5.7. Material parameters are defined as:

Phase	**Type**	C_{11}	C_{12}	C_{44}
γ	FCC	204.6×10^3	137.7×10^3	126.2×10^3
α	BCC	236.9×10^3	140.6×10^3	116.0×10^3

Table 5.6: Single crystal elastic constants

Phase	m	$\dot{\gamma}_0$	h_0	g_0	g_{s0}	n
γ/α	0.02	1.0	391.9	200.0	335.0	1.0

Table 5.7: Initial slip system strengths and other plasticity parameters

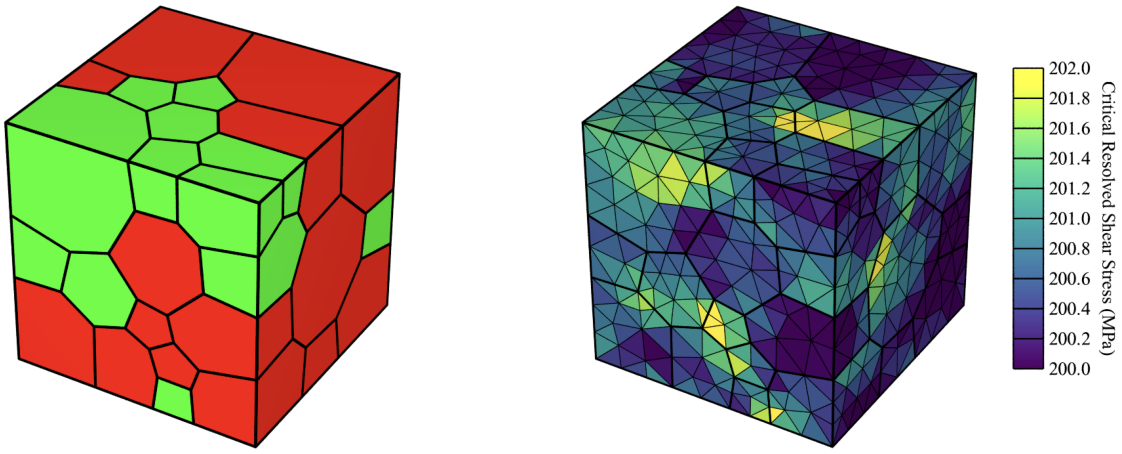

Figure 5.6: (left) Grain and phase assignment distribution in the virtual sample. Red-colored grains are γ-phase and green-colored grains are α-phase. (right) Elastically unloaded sample colored by critical resolved shear stress.

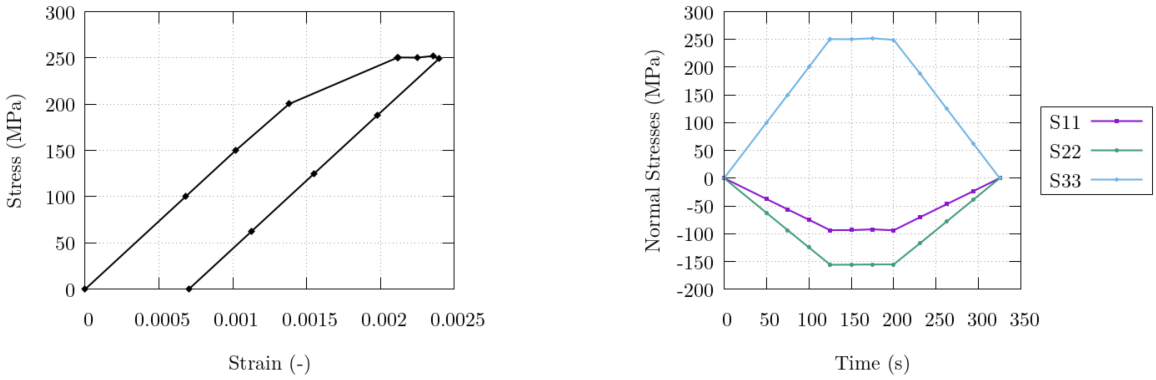

Figure 5.7: (left) Evolution of the macroscopic normal strains, and (right) evolution of the macroscopic stress and strain on the Z surface. Note that the load rate in the Z direction is always held constant during the simulation (except during the dwell episode) while the other two are automatically modified to maintain load proportionality throughout the simulation.

5.4 Restarting a Simulation with Appended Load Steps (`examples/4_restart`)

An example use case for the restart capabilities are to append additional loading steps to a completed simulation. This example covers the cyclic triaxial deformation of an FCC polycrystal at a constant strain rate. Each restart simulation adds an addition load-unload cycle. Loads are applied normal to the surface, maintaining proportional macroscopic load ratios of 0:0:1 for the X:Y:Z directions, respectively. Load tolerance options are prescribed and cyclic hardening is activated. The primary loading direction is set to be in the Z direction. Elemental equivalent plastic deformation rate, slip system shears, slip system shear rates, nodal coordinates, and restart files are output. Restart files are only printed on the first cycle. Material parameters are those for a AL6XN steel and are provided in Table 5.8 and Table 5.9. Illustrations of the results are provided in Figure 5.8.

The included shell script will run the initial simulation in `examples/4_restart/cycle1` which runs 2 load steps (a single load-unload cycle) and prints the files necessary to restart the simulation from the final state (using print option `print restart`). After successful completion of the first cycle, the mesh file and restart files are copied into the secondary directory (`examples/4_restart/cycle2`) and the simulation is performed again for another 2 load steps (a second load-unload cycle). The configuration file for the second cycle (`simulation_cycle2.config`) contains the following input to allow for simulating additional load steps to those already completed:

```
restart on
```

along with the additional load steps in the '`Deformation History`' section. The restarted simulation will continue with the load steps as defined in `simulation_cycle2.config`. Restart control information will print to the console upon the execution of the second cycle to briefly assess the state of the sample when the simulation is restarted. Material parameters are defined as:

Phase	**Type**	C_{11}	C_{12}	C_{44}
α	FCC	204.6×10^3	137.7×10^3	126.2×10^3

Table 5.8: Single crystal elastic constants

Phase	m	$\dot{\gamma}_0$	h_0	g_0	g_{s0}	n
α	0.02	1.0	375.0	160.0	1000.0	1.0

Table 5.9: Initial slip system strengths and other plasticity parameters

Phase	a	c
α	0.05	3.50

Table 5.10: Cyclic hardening parameters

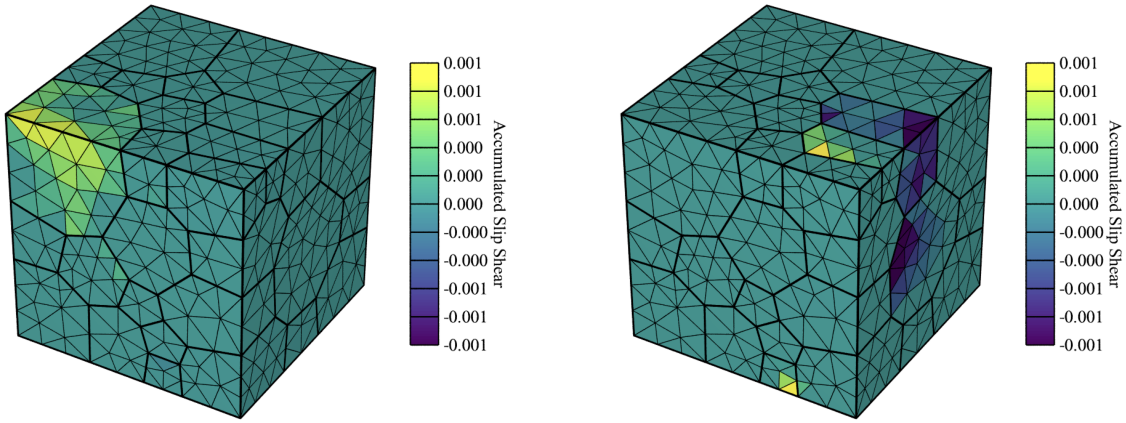

Figure 5.8: Sample after the second cycle is completed, (left) colored by accumulated slip shear on the $(1\bar{1}1)[011]$ system and (right) colored by accumulated slip shear on the $(1\bar{1}1)[10\bar{1}]$ system.

5.5 Running a Simulation with External Definition Files (examples/5_external)

Certain simulation input may be supplied from external files in order to allow for a static configuration and mesh file to be used for multiple simulations while certain microstructure information is varied. This example covers the uniaxial loading and unloading (via load targeting) of a dual phase FCC/BCC polycrystal including in-grain orientation distributions. Two steps are defined with a standard time-step value of 0.1 s and a minimum time-step value of 0.01 s. Minimal boundary conditions are applied to the sample with the primary loading face set as '`x1`' loaded in the +X direction. Elemental orientations and grain/phase assignments are supplied from the external files `simulation.ori` and `simulation.phase`, respectively. Elemental equivalent deformation rate, work, orientations, surface-integrated forces, and nodal coordinates are output. Fiber-averaging processing is enabled and data for three fibers for the first phase is generated per-step. Material parameters are those for the austenitic (γ) and ferritic (α) phases of a LDX-2101 steel and were previously provided in Table 5.6 and Table 5.7. Illustrations of the results are provided in Figure 5.10 and Figure 5.11.

Phases are assigned to grains in `simulation.phase` based on grain ID, resulting in a 50%/50% phase distribution, by:

```
$Groups
elset
50
1 1
2 1
...
26 2
...
50 2
$EndGroups
```

Orientations are assigned to individual elements in the mesh by:

```
$ElementOrientations
5455 rodrigues:active
1 -0.251740631650 -0.214936324918 0.002481866070
2 -0.263893837934 -0.212836283407 0.021747296015
...
5454 -0.062896691423 0.070800028157 0.312930553247
5455 -0.055849086418 0.072826013322 0.294773397825
$EndElementOrientations
```

Note that even though '`$ElsetOrientations`' are also present in `simulation.ori` as generated via Neper, the presence of '`$ElementOrientations`' will always override any other orientations present in `simulation.ori` or in the mesh file (`simulation.msh`).

The definitions of the crystallographic fibers for the fiber averaging routine are defined in `simulation.fib` by:

```
1 1 1 -1.000000  0.000000  0.000000 1 5
1 1 1 -0.993884 -0.110432  0.000000 1 5
...
2 2 0  0.993884  0.110432  0.000000 1 5
2 2 0  1.000000  0.000000  0.000000 1 5
```

Three unique fibers (‘`hkl`’ triplets) are defined in the file. The associated sample directions (‘`uvw`’ triplets) define unique points on a coarse mesh of a pole figure. The selection of sample directions in this manner allows for the fiber averaged output data to be visualized as a pole figure as seen in Figure 5.12.

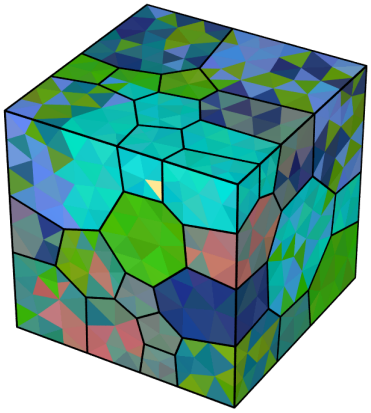

Figure 5.9: Undeformed sample colored by orientation. Per-element orientations are assigned to the sample allowing for the prescription of initial misorientation within grains. Per-element orientations are generated from a 3-variate normal distribution with an average misorientation angle (with respect to a grain’s average orientation) of 5 degrees.

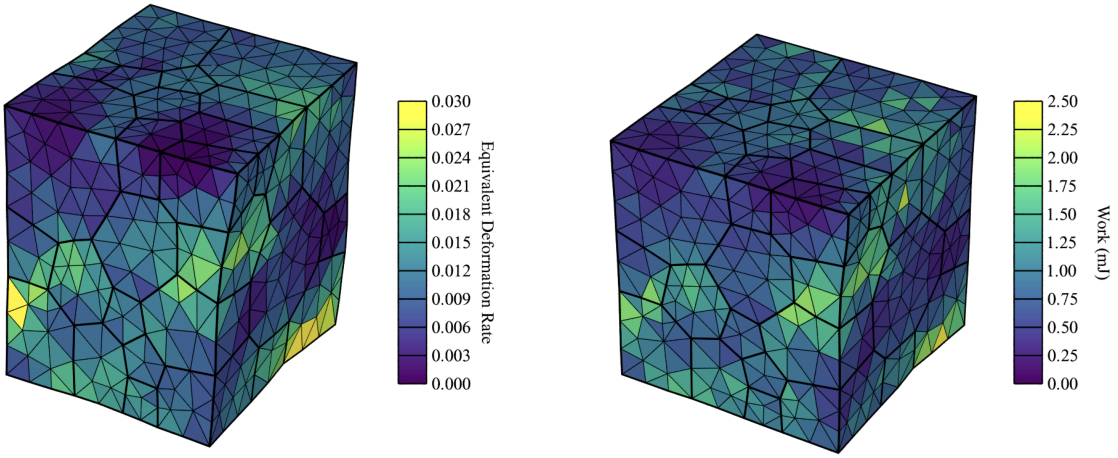

Figure 5.10: Deformed sample after the (left) first step and (right) second step are completed (deformation field is exaggerated 50x for illustrative purposes). The left figure is colored by equivalent deformation rate and the right figure is colored by work. Note that, unlike in Figure 5.2, the edges of the control surfaces need not remain constrained with the minimal boundary conditions.

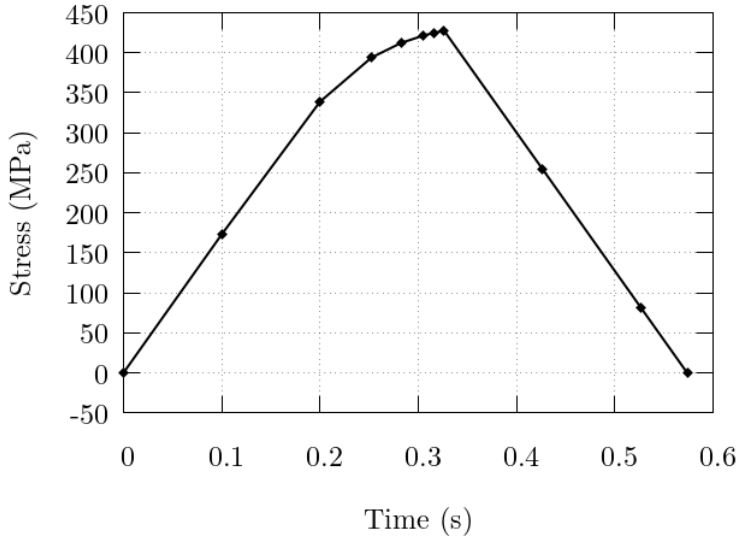

Figure 5.11: Evolution of the macroscopic normal stress. Note the increased point densities near to the load direction change exhibiting the simulation time-step value decreasing to accurately reach the load targets.

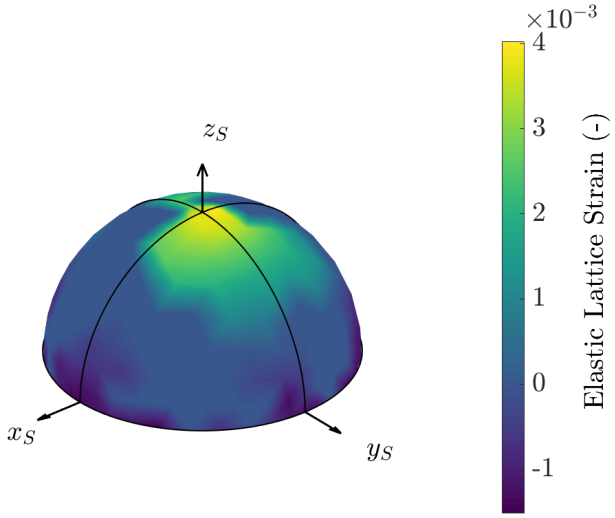

Figure 5.12: A lattice strain pole figure depicting the fiber-averaged elastic lattice strain for the {200} planes of the FCC phase (i.e., phase '1') at the end of step 1. Refinement of the pole figure would benefit from an increase in the number of grains in the simulated domain, as well as the inclusion of more unique fibers on the mesh of the pole figure.

Appendix A Development History

Milestones in the development of FEpX:

Development of present code began in the late 1990's. The impetus was to incorporate elasticity in the existing viscoplastic constitutive framework. A number of attributes of the mechanical response can be addressed only if elasticity is part of the total constitutive description, and several of these were important to the research goals at that time. The inclusion of elasticity, however, fundamentally alters the computational approach because elasticity is based on changes in configuration of differential volumes, whereas plasticity requires knowledge only of current configurations. Further, the stiff mathematical character of the resulting system of equations necessitates greater care be exercised in integrating the equations over time and motivates the use of implicit schemes. While introducing added complexity in this regard, the inclusion of elasticity removes the constraint of incompressibility, which is particularly difficult to enforce within a robust framework and limits options available for the computational methodologies. Given these factors, the intent of the effort was to develop a code to support research investigations with the following defining specifications:

- incorporates anisotropic elasto-viscoplastic behaviors, especially in materials with low rate sensitivity of yielding such as metals at low homologous temperature;
- embodies nonlinear kinematics, which are necessary for handling the large strains and large rotations inherent in plastic flow;
- utilizes state-based representation of properties, with attention to verifiable state descriptions at microscale as motivated by the inclusion of crystallographic texture (for yield surfaces) and evolution of texture (for strain induced anisotropy);
- exercises robust numerical methodologies, including implicit integration for stress, element types capable of large strain deformations, and general boundary conditions;
- features a data parallel implementation with good scaling characteristics; and,
- is an expansible code framework to facilitate testing of alternative plasticity models and numerical methodologies.

Incorporation of a code with elasticity and nonlinear kinematics (which are tightly coupled) was carried out principally by E. Marin [1, 2]. The starting point for this effort was a code that utilized a viscoplastic model implemented in parallel with a hybrid finite element formulation developed by A. Beaudoin [3] (the hybrid formulation was an effective approach for dealing with the incompressibility constraint in the presence of plastic anisotropy). This code had been migrated to a version using Fortran and MPI and had 10-node tetrahedral elements available for robust simulations of large plastic strains [4]. The approach taken by Marin followed a methodology developed for isotropic elasto-viscoplastic behavior developed about a decade earlier by Eggert [5]. Refinements were added later by N. Barton [6] for anisotropic elasticity.

Earlier development leading to the present version:

The use of finite elements with crystal plasticity models can be organized into two broad categories that are defined by the relative sizes of grains and elements [7]. One category is labeled, 'large scale', and is defined by the grains being much smaller than elements; the other is labeled, 'small scale', and is defined by the elements some part of a grain. In large scale applications, an ensemble of grains underlies spatial points at the continuum scale and defines the properties of the continuum at that point. In small scale applications, the volume within a finite element is entirely of one grain and the material exhibits properties of a single crystal. The code development leading eventually to FEpX began with a large scale implementation in which crystal plasticity was embedded in an Eulerian, viscoplastic formulation was devised by K. Mathur [8] to model steady-state metal forming applications. The potential capabilities of a code that incorporated

crystal plasticity was pursued because it permitted computing the evolving anisotropy associated with plastic yielding directly derived from crystallographic texture. Subsequent application of the approach for metal forming and geologic flow demonstrated that polycrystal models were viable for flow fields that could be idealized as steady and two-dimensional [9, 10, 11].

The desire to model transient processes, such as sheet forming, motivated the major effort to develop a data parallel code. The code developed in this effort employed a proprietary version of Fortran [12] that managed interprocessor communications, and enabled the simulation of fully three-dimensional forming processes [13, 14]. An interest in applying the approach to small scale as well as large scale problems subsequently led to development of the hybrid finite element formulation for polycrystals [3]. This milestone solidified the role of the finite element approach for investigating the role of grain interactions in polycrystal deformations and opened the door to investigating the strengths and limitations of various mean field assumptions (e.g. Taylor, Sachs, Relaxed Constraints, and Constrained Hybrid). Concurrently, the parallel computing landscape was rapidly evolving, and to take advantage of the introduction of new platforms, the code architecture was re-structured to employ Message Passing Interface (MPI) routines to conduct interprocessor communications. This Fortran/MPI version remained limited to purely viscoplastic behaviors, but was exploited to study texture evolution in polycrystals as well as development of intragrain misorientation distributions. This code was the starting point for development of FEpX and a re-focusing of the simulation priorities on small scale applications.

Application-driven expansion of capabilities:

FEpX development in the decade following the launch of the first version of the present code was centered on support of investigations related to mechanical behaviors of polycrystals. Improvements were made in numerical procedures to improve robustness (namely, the nonlinear solver, quadrature rules, and integration of state variables). Modifications were implemented to provide options in the loading histories that enabled better replication of experimental loading protocols. In particular, options to invoke more complex sequences of loading, unloading and reloading used in *in situ* loading, x-ray and neutron diffraction experiments were implemented. Capabilities for cyclic and multiaxial loading were added.

The extraction of data related to the orientation of the crystallographic lattice from the simulation was of paramount importance. Routines were implemented to identify elements of the mesh whose lattice orientations lie near crystallographic fibers, a process referred to as 'light-up' in analogy to diffraction measurements. The output of FEpX was coordinated with a number of ancillary capabilities for manipulation of orientation-dependent variables (ODFPF), representation of anisotropic yield surfaces, and execution of a virtual diffractometer.

Definition of the virtual polycrystals simulated with FEpX was initially limited to regular tessellations comprised of dodecahedral grains. Every grain was discretized with tetrahedral elements, typically numbering from 48 for a coarse representation to 1536 for more finely resolved grains. The use of other regular tessellations (cubic and truncated octahedral, in particular) were also explored [15]. The coupling of FEpX with Neper greatly improved the representation of virtual polycrystals by allowing for irregular Voronoi or Laguerre tesselations and facilitating re-meshing in simulations taken to large plastic strains [16].

Individuals contributing to these improvements include: R. Carson, D. Boyce, T. Han, M. Kasemer, T. Marin, A. Poshadel, R.Quey, and S.-L. Wong.

Source sharing and documentation:

By 2010 the use of FEpX over a decade in a variety of research projects motivated a push for standardization, version control, and sharing best done within a collaborative platform. A repository was established in 2012 together with documentation (users manual compiled by A. Mitch) for the needed input and possible outputs for FEpX. In concert with the establishing

the code repository, numerous improvements were made in the organization of input and output data. A full description of the underlying theory and finite element implementation was posted on arXiv in 2015 [17]. Individuals contributing to this effort include A. Poshadel and M. Kasemer.

Extensions of FEpX:

One of the specifications of FEpX was to provide an expansible code framework to facilitate testing of alternative plasticity models and numerical methodologies. Such efforts typically require substantial alterations to the code and are not intended to result in permanent changes to the baseline code. Examples of investigations of this nature include: a kinematic model with slip gradients [18]; a continuous intragrain lattice representation [19], and a kinematic framework for twinning [20].

Dr. Paul R. Dawson
Joseph C. Ford Professor of Engineering Emeritus
Sibley School of Mechanical and Aerospace Engineering
Cornell University

Appendix B References

1. E. B. Marin and P. R. Dawson. On modeling the elasto-viscoplastic response of metals using polycrystal plasticity. *Computer Methods in Applied Mechanics and Engineering*, 165:1-21, 1998.
2. E. B. Marin and P. R. Dawson. Elastoplastic finite element analysis of metal deformations using polycrystal constititive models. *Computer Methods in Applied Mechanics and Engineering*, 165:23-41, 1998.
3. A. J. Beaudoin, P. R. Dawson, K. K. Mathur, and U. F. Kocks. A hybrid finite element formulation for polycrystal plasticity with consideration of macrostructural and microstructural linking. *International Journal of Plasticity*, 11:501-521, 1995.
4. D. P. Mika and P. R. Dawson. Polycrystal Plasticity Modeling of Intracrystalline Boundary Textures. *Acta Materialia*, 47(4):1355-1369, 1999.
5. G. M. Eggert and P. R. Dawson. A viscoplastic formulation with elasticity for transient metal forming. *Computer Methods in Applied Mechanics and Engineering*, 70:165-190, 1988.
6. N. R. Barton, P. R. Dawson, and M. P. Miller. Yield strength asymmetry predictions from polycrystal plasticity. *Journal of Engineering Materials and Technology*, 121:230-239, 1999.
7. P. R. Dawson and E. B. Marin. Computational mechanics for metal deformation processes using polycrystal plasticity. In Erik van der Giessen and Theodore Y. Wu, editors, *Advances in Applied Mechanics*, 34:78-169. Academic Press, 1998.
8. K. K. Mathur and P. R. Dawson. On modeling the development of crystallographic texture in bulk forming processes. *International Journal of Plasticity*, 5:67-94, 1989.
9. K. K. Mathur and P. R. Dawson. Texture development during wire drawing. *Journal of Engineering Materials and Technology*, 112(3):292-297, 1990.
10. A. Kumar and P. R. Dawson. Polycrystal plasticity modeling of bulk forming with finite elements over orientation space. *Computational Mechanics*, 17:10-25, 1995.
11. P. R. Dawson and H.-R. Wenk. Texturing the upper mantle during convection. *Philosophical Magazine A*, 80(3):573-598, 2000.
12. K. K. Mathur. Parallel algorithms for large scale simulations in materials processing. In S. F. Shen and P. R. Dawson, editors, *Simulation of Materials Processing: Theory, Methods and Applications – NUMIFORM 95*, 109-114. A. A. Balkema, 1995.
13. A. J. Beaudoin, K. K. Mathur, P. R. Dawson, and G.C. Johnson. Three-dimensional deformation process simulation with explicit use of polycrystalline plasticity models. *International Journal of Plasticity*, 9:833-860, 1993.
14. A. J. Beaudoin, P. R. Dawson, K. K. Mathur, U. F. Kocks, and D. A. Korzekwa. Application of polycrystal plasticity to sheet forming. *Computer Methods in Applied Mechanics and Engineering*, 117:49-70, 1994.
15. H. Ritz and P. R. Dawson. Sensitivity to grain discretization of the simulated crystal stress distributions in fcc polycrystals. *Modeling and Simulation in Materials Science and Engineering*, 17:1-21, 2009.
16. R. Quey, P. R. Dawson, and F. Barbe. Large-scale 3-D random polycrystals for the finite element method: generation meshing and remeshing. *Computer Methods in Applied Mechanics and Engineering*, 200:1729-1745, 2011.
17. P. R. Dawson and D. E. Boyce. FEpX – Finite Element Polycrystals: Theory, finite element formulation, numerical implementation and illustrative examples. *arXiv:1504.03296 [cond-mat.mtrl-sci]*, 2015.

18. J. M. Gerken and P. R. Dawson. A finite element formulation to solve a non-local constitutive model with stresses and strains due to slip gradients. *Computer Methods in Applied Mechanics and Engineering*, 197:1343-1361, 2008.
19. R. A. Carson and P. R. Dawson. Formulation and Characterization of a Continuous Crystal Lattice Orientation Finite Element Method (LOFEM) and its Application to Dislocation Fields. *Journal of the Physics and Mechanics of Solids*, 126:1-26, 2019.
20. M. P. Kasemer and P. R. Dawson. A finite element methodology to incorporate kinematic activation of discrete deformation twins in a crystal plasticity framework. *Computer Methods in Applied Mechanics and Engineering*, 358:112653, 2020.
21. R. Carson, M. Obstalecki, M. Miller, and P. R. Dawson. Characterizing heterogeneous intragranular deformations in polycrystalline solids using diffraction-based and mechanics-based metrics. *Modelling and Simulation in Materials Science and Engineering*, 25:055008, 2017.
22. H. S. Turkmen, M. P. Miller, P. R. Dawson, and J. C. Moosbrugger. A slip-based model for strength evolution during cyclic loading. *Journal of Engineering Materials and Technology*, 126(4):329-338, 2004.

Appendix C GNU General Public License

GNU General Public License

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a devel copy. Propagation includes copying, distribution (with or without modification), making available to the distrib, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the distrib in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms

that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general distrib at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is distribly documented (and with an implementation available to the distrib in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for distribity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a distribly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

    If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's distrib statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRIT-

ING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the distrib, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
```

```
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.