PRISMS-Fatigue: Python script installation and execution guide

Krzysztof S. Stopka

Version 1.0, November 2020

Introduction

This document describes the five Python scripts associated with the PRISMS-Fatigue toolkit. The user is required to download Python on their local computer. Several common libraries are required which are listed and imported at the top of each script (e.g., NumPy, os, pickle, pandas, etc.). The majority of these should come preinstalled with a standard Python installation package such as Anaconda or equivalent. A recent version of the open-source microstructure instantiation software DREAM.3D is also required [1]. ParaView is used for visualization.

There are several publications that should be read thoroughly by aspiring Prisms-Fatigue users [2-4]. Users should watch the PRISMS-Plasticity tutorial videos located at: https://www.youtube.com/playlist?list=PL4yBCojM4Swqy4FRteqxHWSiM1uiOOesj. There are also a set of tutorial videos associated with these scripts located at https://www.youtube.com/playlist?list=PL4yBCojM4Swo3CvlA57syFrzk3p1mugP5 to supplement this guide. The five Python scripts are listed below and described subsequently in this document. Users are strongly encouraged to examine each script in detailed to gain a fundamental understanding of the workflows!

1. *generate_microstructures.py*
2. *calculate_FIPs.py*
3. *volume_average_FIPs.py*
4. *compule_and_plot_FIPs.py*
5. *gamma_plane.py*

The raw data associated with the PRISMS-Fatigue manuscript is available for download from Materials Commons at: https://doi.org/10.13011/m3-rcyy-gx13. The results and figures in the PRISMS-Fatigue manuscript can be entirely replicated by downloading this raw data and executing the last four Python scripts. The video tutorials do just this, so users are encouraged to go through this document as well as the online videos.

Python installation

Python can be downloaded and installed at https://www.anaconda.com/products/individual for several operating systems as listed below, but the user is free to install Python in whichever way is most convenient. Various Python packages are required to execute the scripts that are straightforward to install. The scripts are backwards compatible with older 2.X versions of Python.

Fig. 1. Python installation options at https://www.anaconda.com/products/individual.

ParaView

The open-source software ParaView is used to visualize microstructures and PRISMS-Plasticity simulation results. It is available at https://www.paraview.org/download/ for users to download.

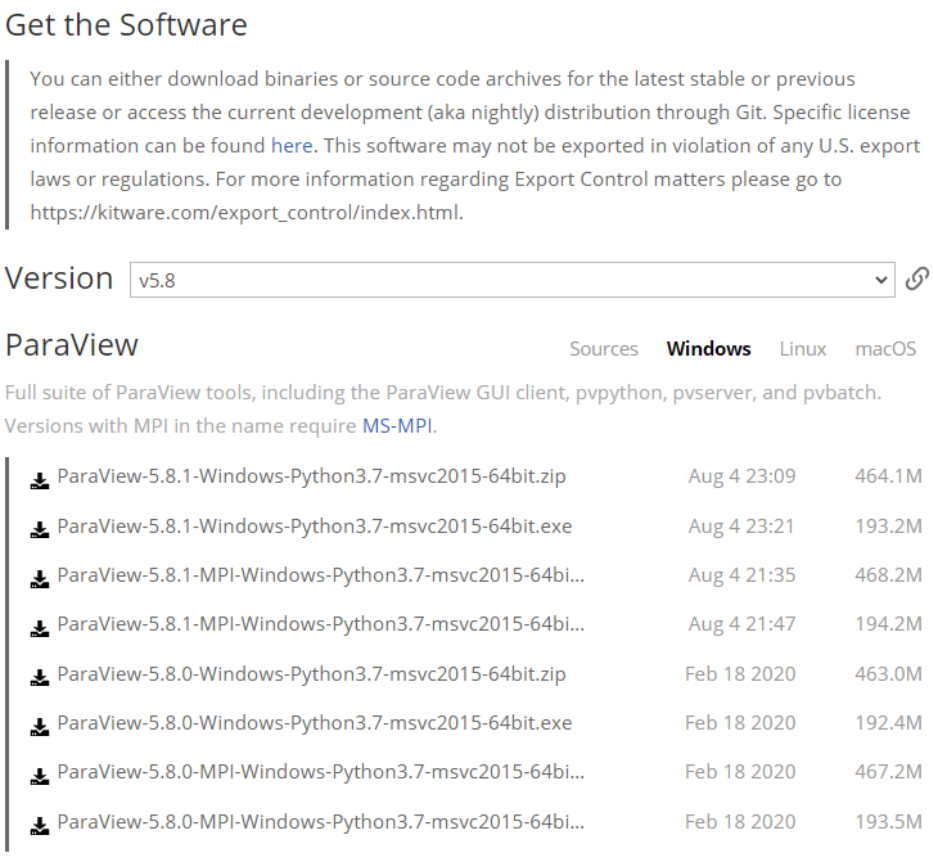


Fig. 2. ParaView installation options at https://www.paraview.org/download/.

DREAM.3D installation

DREAM.3D is a powerful, open-source software with many capabilities including (but not limited to) generation of synthetic microstructures, microstructure reconstruction using data sets such as electron back scatter diffraction (EBSD) or high energy x-ray diffraction microscopy (HEDM), and analysis of statistical information [1]. It is available for download at http://dream3d.bluequartz.net/. Prospective PRISMS-Fatigue users should read the work by Groeber and Jackson [1] to understand the structure of the program, which is briefly reviewed here. In addition to the PRISMS-Fatigue video tutorials, DREAM.3D specific tutorials for more advanced users are available at https://www.youtube.com/channel/UCjeF8pFMzET5ZN3vsBHATpg. A screenshot of the program files and folders is shown in Fig. 3. The executable *DREAM3D.exe* opens a blank Graphical User Interface (GUI) as shown in Fig. 4.

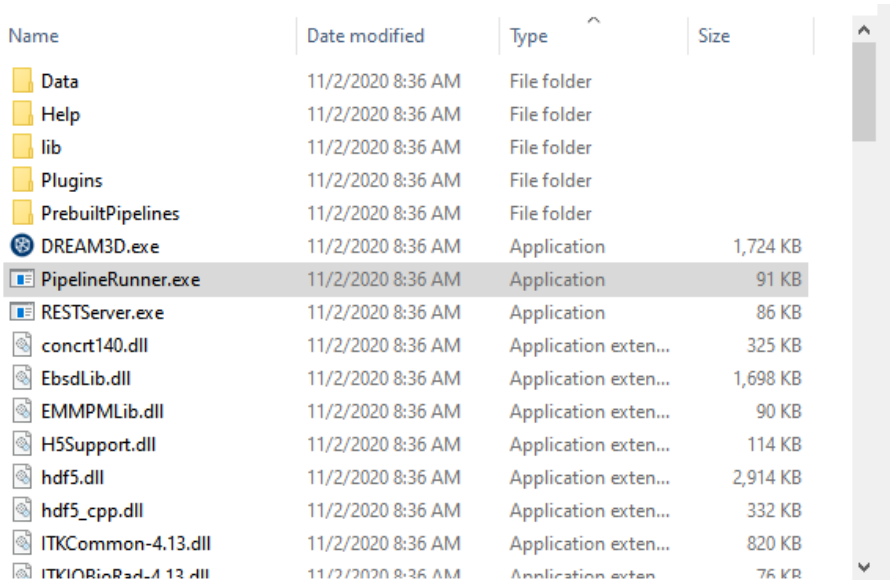


Fig. 3. DREAM.3D program folder (version 6.5.141 downloaded November 2020). The path of the *PipelineRunner.exe* executable is an input in the first Python script.

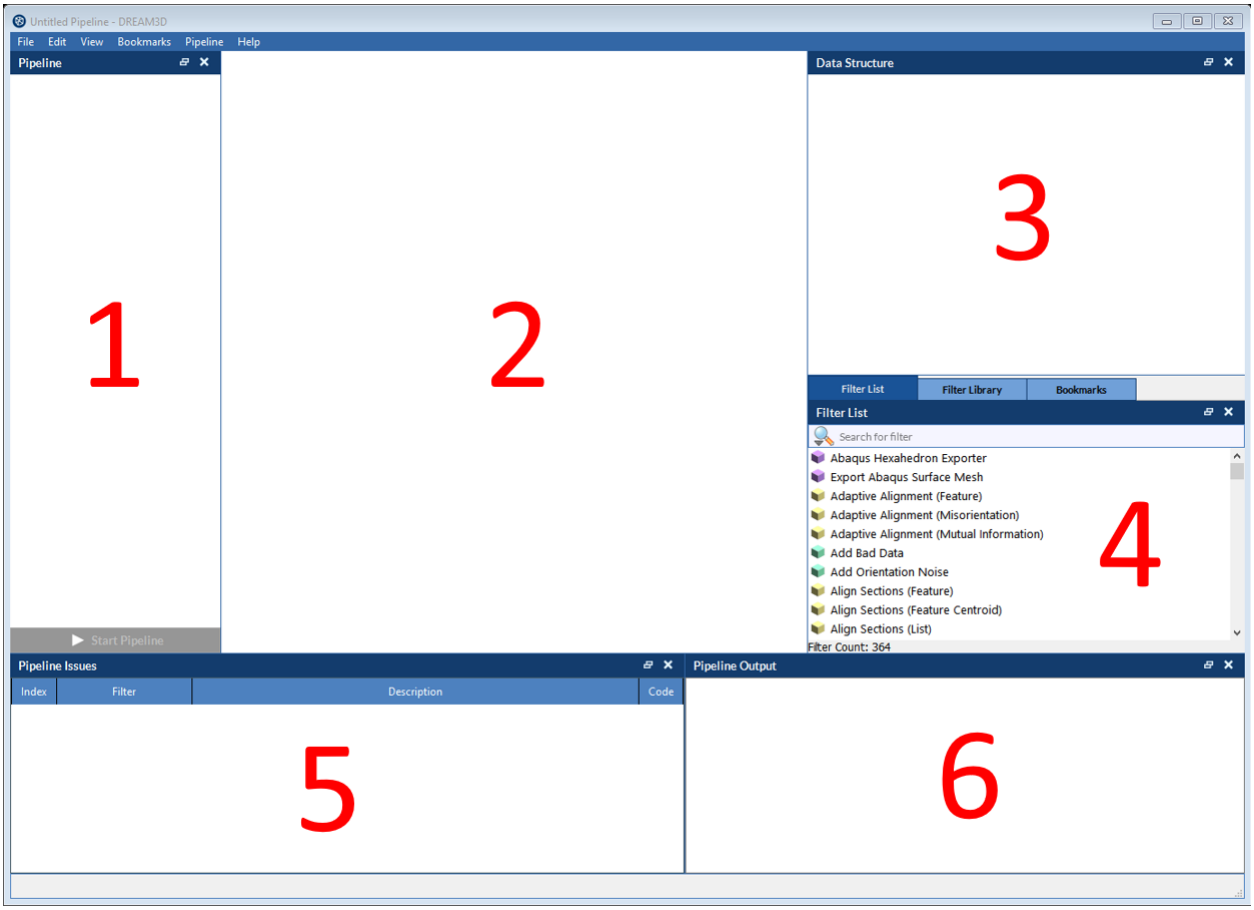


Fig. 4. Empty DREAM.3D GUI. Filters from region 4 are "dragged and dropped" into region 1 for sequential execution. Region 2 contains unique settings for each filter. The structure of the data is shown in region 3. Regions 5 and 6 display pipeline issues/warnings/errors and pipeline output, respectively.

Fig. 5 shows a screenshot of what will be referred to as the microstructure instantiation pipeline "input file." This file consists of only two filters. The first specifies the desired microstructure statistics that include crystal symmetry, grain size distribution, crystallographic texture, grain morphology, phase volume fraction, etc. Once these options are set, the user must click on "Create Data"! The second filter simply writes this information to a ".dream3d" type file. At this point, only the desired *statistics* are stored in this file. As part of the initial PRISMS-Fatigue release, six input files are available for face centered cubic microstructures. These include equiaxed and elongated/rolled grain morphologies, and cubic, random, and rolled target crystallographic textures. Sample microstructure instantiations are depicted in Fig. 6. There are several other DREAM.3D files available as part of PRISMS-Plasticity.

An important note is that any changes made to the DREAM.3D "input" file shown in Fig. 5 **cannot** be simply saved by clicking "File → save"! The user must instead click on "Create Data" in the middle of the screen on the "StatsGenerator" options and then click "Start Pipeline" to execute these two filters in sequence. The name of the resultant .dream3d file in the second filter should also be updated.

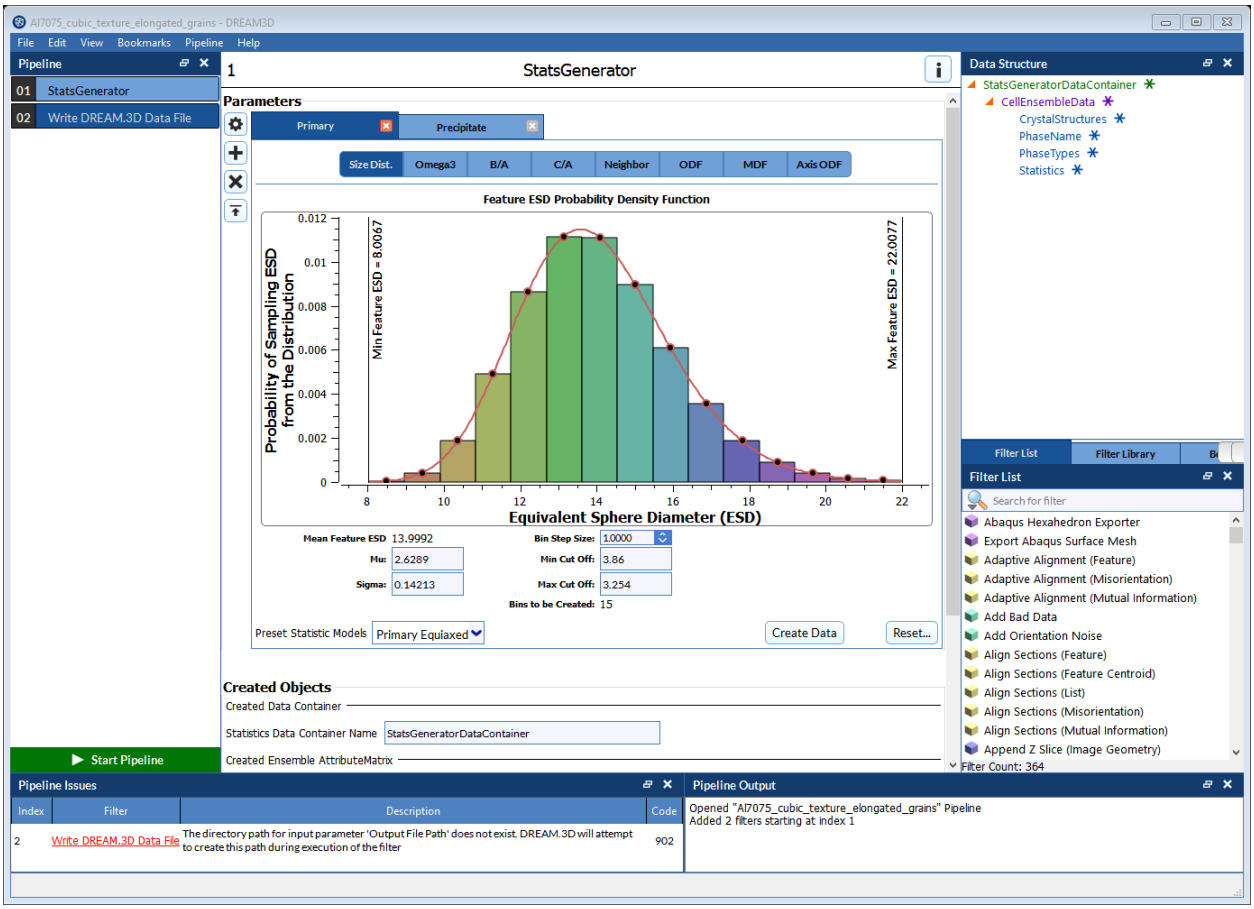


Fig. 5. DREAM.3D "input file" consisting of two filters.

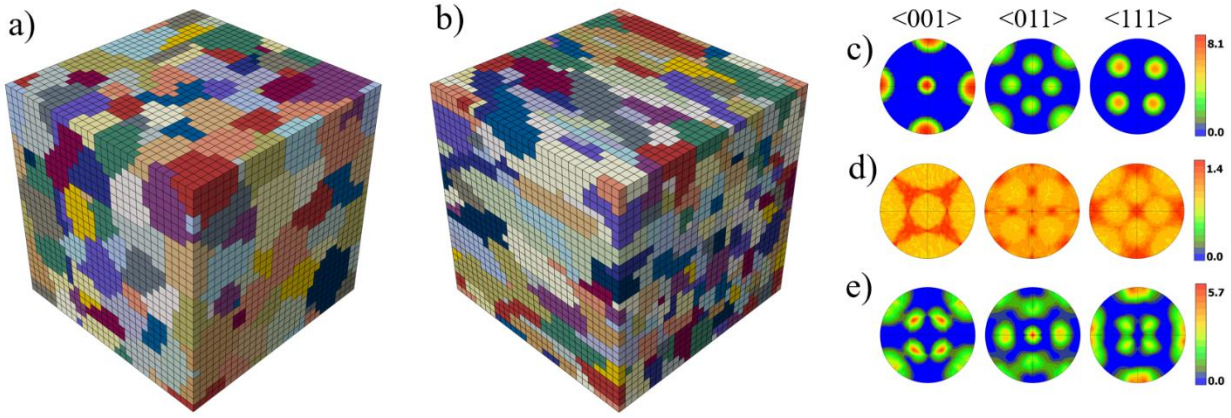


Fig. 6. Sample microstructure instantiations with (a) equiaxed and (b) elongated/rolled grain (grain elongation ratio of 5:1:1) morphology. The different target crystallographic textures include (c) cubic, (d) random, and (e) rolled.

The second DREAM.3D file is shown in Fig. 7 and is referred to as the DREAM.3D "pipeline." Note that the first filter in this pipeline is "Read DREAM.3D Data File" which reads in a ".dream3D" file, for instance the one created by the "input file" shown in Fig. 5. Although a single .json "pipeline" can be used to instantiate microstructures, the scripted workflows presented here split these into an "input" file and a "pipeline." The latter remains unchanged and contains the necessary filters to instantiate microstructures and create other necessary files whereas the former is a simple file that users can edit to change target microstructure statistics. The pipeline file has the .json extension and can be edited as a text file. In fact, the *generate_microstructures.py* script edits the .json pipeline file before execution with user inputs in the main() function.

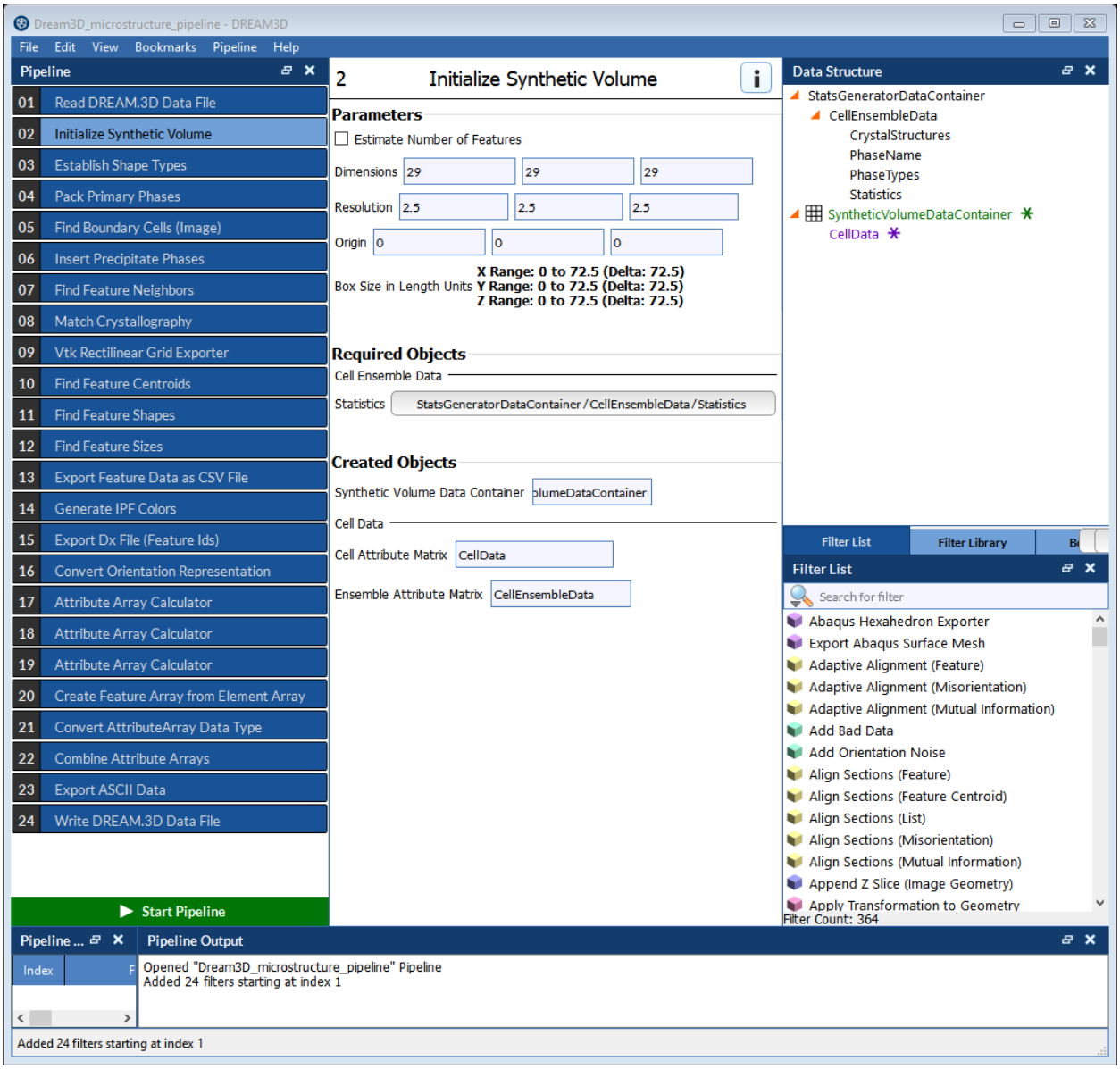


Fig. 7. DREAM.3D "pipeline file" with the filters necessary to instantiate microstructures for CPFE simulation.

generate_microstructure.py

The first script generates microstructures for CPFE simulation in PRISMS-Plasticity. A screenshot of the "main()" input function to be edited by the user is shown below. This and the other four Python scripts can be called either from a command prompt window as "python *generate_microstructure.py*" or using an interactive version of Python (i.e., executing the "ipython" command in command prompt and importing the script). The latter method is particularly useful because of its debugging capabilities. Fig. 9 and Fig. 10 depict both of these methods. The terms voxel and element are used interchangeably in this guide since the microstructure voxels directly represent the elements for CPFE simulations.

```
def main():
    # Run from command prompt

    # Directory where microstructure data should be instantiated and pre-processed
    # This command creates a directory in the same directory as the "PRISMS-Fatigue" directory with python scripts and
    DREAM.3D files
    # directory = os.path.dirname(DIR_LOC) + '\\PRISMS-Fatigue_tutorial\\test_json'

    # Alternatively, the directory can be expressed as an absolute path as:
    directory = r'C:\Users\stopk\Documents\GitHub\demo'

    # Location of DREAM.3D input file; should consist of only the "StatsGenerator" and "Write DREAM.3D Data File"
    # "StatsGenerator" inputs include grain size distribution, crystallographic texture, grain morphology, etc.
    # Six ".dream3d" files are included in PRISMS-Fatigue
    d3d_input_file = os.path.abspath(DIR_LOC) + '\\Al7075_random_texture_equiaxed_grains.dream3d'

    # Once again, this may be specified using an absolute path as:
    # d3d_input_file = r'C:\Users\stopk\Documents\GitHub\PRISMS-Fatigue\Al7075_cubic_texture_equiaxed_grains.dream3d'

    # Location of DREAM.3D .json pipeline
    # This can be modified by the user to include additional outputs
    d3d_pipeline_path = os.path.abspath(DIR_LOC) + '\\Dream3D_microstructure_pipeline.json'

    # Once again, this may be specified using an absolute path as:
    # d3d_pipeline_path = r'C:\Users\stopk\Documents\GitHub\PRISMS-Fatigue\Dream3D_microstructure_pipeline.json'

    # Location of DREAM.3D 'PipelineRunner.exe' file; this should be in the DREAM.3D program folder
    d3d_executable_path = r'C:\Users\stopk\Desktop\DREAM3D-6.5.141-Win64\PipelineRunner.exe'

    # Size of microstructure instantiations in millimeters, in the X, Y, and Z directions, respectively.
    size  = np.asarray([.0725,.0725,.0725])

    # Shape of microstructure instantiations, in the X, Y, and Z directions, respectively.
    # IMPORTANT: at this point, only CUBIC voxel functionality supported
    shape = np.asarray([29,29,29])

    # Number of elements in each sub-band region for volume averaing
    # Please see the references below for more information
    num_vox = 8

    # Number of crystallographic slip planes for FIP Volume averaging
    # There are four slip planes in the Al7075-T6 material system (see references below)
    num_planes = 4

    # Boundary conditions, either "periodic" or "free surface", for X,Y,Z directions
    # Three possible combinations: 1) all 'periodic', 2) all 'free', or 3) two 'periodic' + one 'free'
    face_bc = ['periodic', 'periodic', 'periodic']

    # Number of microstructure instantiations to generate using DREAM.3D
    num_instantiations = 1

    # Specify whether DREAM.3D was previously executed on these files
    # If set to false, the script will NOT generate new DREAM.3D microstructure and instead process the existing
    microstructures by reading the .csv and GrainID_#.txt files
    # Reasons to set this to False:
    #     1) Process the same set of microstrucutres with a different number of elemetns per sub-band regions, and store
    these in a separate folder
    #     2) Generate a set of ['periodic', 'periodic', 'periodic'] microstructures, and then reprocesses them with one
    set of faces set to non-periodic, i.e., ['periodic', 'free', 'periodic']
    generate_new_microstructure_files = True

    # Call to the main function
    gen_microstructures(directory, size, shape, face_bc, num_vox, num_planes, num_instantiations,
    generate_new_microstructure_files, d3d_input_file, d3d_pipeline_path, d3d_executable_path)

    # Print the parameters of this microstructure set to a text file
    print_params(directory, size, shape, face_bc, num_vox, num_planes, num_instantiations, d3d_input_file)
```

Fig. 8. Screenshot of the main() function in *generate_microstructure.py*.

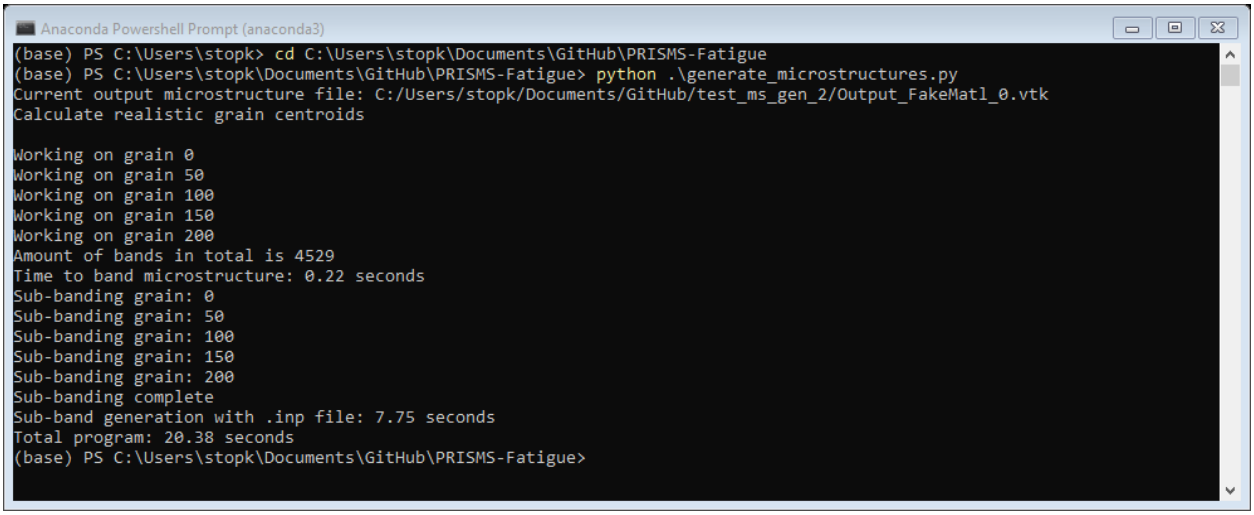


Fig. 9. Command prompt execution of the *generate_microstructure.py* script.

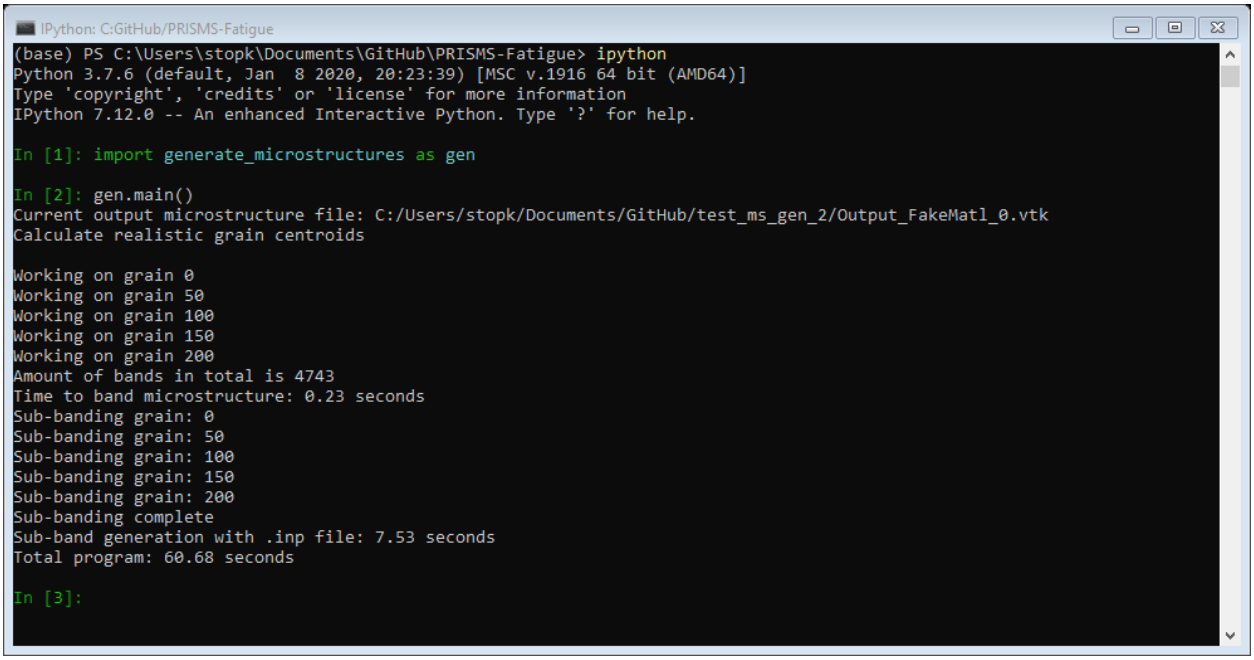


Fig. 10. Command prompt initiation of interactive Python session (i.e., “ipython”) to execute Python scripts.

The main() function contains all the necessary parameters for the user to edit to generate microstructures, as well as descriptions of each variable. This script edits the .json pipeline based on user inputs for the size and shape of instantiations, the number of instantiations to generate, and the locations of the input, pipeline, and executable DREAM.3D files, and then calls DREAM.3D [1] as a subprocess. This and the other Python scripts contain many other functions that the user is encouraged to inspect for a more fundamental understanding of the scripts. A more detailed explanation for each variable is included in Table 1 below. Table 2 describes the generated files for each instantiation.

Table 1. List of variables and settings in the *generate_microstructures.py* script.

Variable/setting (*type*)	Description
directory *path to folder*	Path to folder where all microstructure instantiation data should be stored. This is referred to as a single "batch simulation folder." If this folder does not exist, it will be created. Users will note two ways to define the directory as shown in Fig. 8. The default option uses the *DIR_LOC* variable which is the location of the folder that contains the Python scripts and DREAM.3D files. The top of each script defines this with the command below where *__file__* is the Python script file: *DIR_LOC = os.path.dirname(os.path.abspath(__file__))* **NOTE:** If this script is executed twice, the second instance of microstructure instantiations will overwrite the first set (see last variable in this table)!
d3d_input_file *path to file*	Location of the desired DREAM.3D "input" file, which contains two filters: "StatsGenerator" and "Write DREAM.3D Data File." This and the next variable can be specified in the manner above or by using an absolute path to the file location.
d3d_pipeline_path *path to file*	Location of the DREAM.3D .json pipeline that reads in the microstructure statistics defined by d3d_input_file and executes the pipeline.
d3d_executable_path *path to file*	Location of the DREAM.3D *PipelineRunner.exe* executable file (see Fig. 3) required for Python during the subprocess call.
size *float array*	Size of the microstructure instantiation in millimeters in the X, Y, and Z directions, respectively.
shape *Integer array*	Number of voxels in the X, Y, and Z directions, respectively. **NOTE:** At the time of this release, only CUBIC voxels are supported, so the user must ensure that the size divided by shape in each of the X, Y, and Z directions results in an identical voxel/element size!
num_vox *integer*	Number of voxels in each sub-band region for FIP volume averaging. Please see references [2-4] and the *volume_average_FIPs.py* section below for more information. In these references, the sub-band regions consist of 8 elements which is ~8-10% of the grain volume.
num_planes *integer*	Number of unique crystallographic slip planes in the material system under investigation. In the first PRISMS-Fatigue release, the face centered cubic (fcc) aluminum alloy Al 7075-T6 is simulated which has four crystallographic slip planes over which FIPs can be volume averaged.
face_bc *string array*	Defines the boundary conditions for microstructure periodicity in DREAM.3D and controls how FIP averaging volumes are determined. There are three options for users: 1) Set all face boundary conditions to "free": i. Microstructures are instantiated without periodicity in any direction. Afterwards, each grain is "banded" and then "sub-

	banded" to determine FIP volume averaging regions for band-averaged and sub-band averaged FIPs, respectively. This is the simplest type of instantiation. 2) Set all face boundary conditions to "periodic": i. Microstructures are instantiated with periodicity in all three directions, i.e., grain tessellations that reach the boundary of the synthetic volume will "wrap around" and appear on the other side of the instantiation. These instantiations are simulated with periodic boundary conditions in PRISMS-Plasticity. However, additional processing is required before FIPs may be volume averaged. In the process of "banding" and "sub-banding" grains in a microstructure, the grain center of mass (COM) is required. Grains split by instantiation boundaries will have erroneous COM calculations based on their voxel locations. The Python script detects grains split by SVE boundaries and adjusts grain COMs for proper banding and sub-banding. 3) Set any two face boundary conditions to "periodic" and set the third to "free": i. This boundary condition mimics a thin sheet of material [2-4]. Microstructures are instantiated with periodicity in all three directions. Grains split by one pair of parallel faces (determined by which of the face boundary conditions is set to "free") are detected and indexed as new grains. Grain periodicity is retained in the other two directions and so this emulates a thin sheet of material with a high surface area to volume ratio. **NOTE:** It is crucial that the subsequent CPFE simulations in PRISMS-Plasticity employ appropriate boundary conditions (i.e., with periodicity enforced along two directions)! DREAM.3D has the option to generate microstructures as either periodic or non-periodic. The third option described here provides users advanced control of subsequent FIP averaging. Fig. 11 depicts these three types of boundary conditions.
num_instantiations *integer*	Number of microstructure instantiations to generate in this batch simulation folder. Each unique microstructure file contains the suffix "_#" where # is the number of each instantiation and is indexed at 0 (see Fig. 12 and Table 2).
generate_new_ microstructure_files *Boolean*	Specify whether *new* microstructures should be instantiated (*True*) or whether previous _#.csv and GrainID_#.txt files should be read to band and sub-band microstructures (*False*).

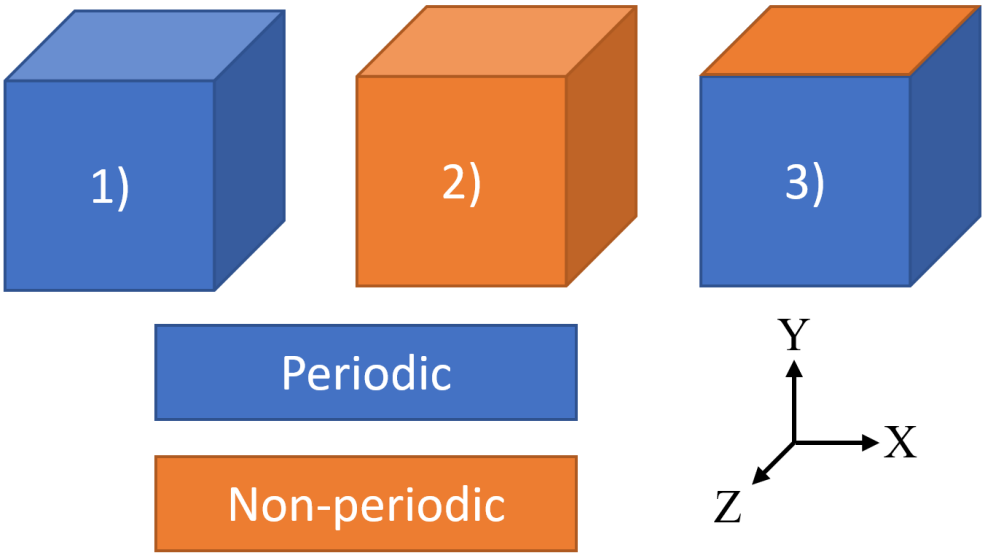


Fig. 11. Depiction of the three combinations of boundary conditions as described in Table 1.

Name	Date modified	Type	Size
band_plane_normals_0.txt	11/2/2020 12:41 PM	TXT File	55 KB
band_sets_0.txt	11/2/2020 12:41 PM	TXT File	812 KB
Dream3D_microstructure_pipeline.json	11/2/2020 12:41 PM	JSON File	26 KB
El_pos_0.p	11/2/2020 12:41 PM	MATLAB P-code	572 KB
element_band_sets_0.p	11/2/2020 12:41 PM	MATLAB P-code	485 KB
element_grain_sets_0.p	11/2/2020 12:41 PM	MATLAB P-code	204 KB
elements_per_band_0.txt	11/2/2020 12:41 PM	TXT File	78 KB
FeatureData_FakeMatl_0.csv	11/2/2020 12:41 PM	Microsoft Excel C...	91 KB
grainID_0.txt	11/2/2020 12:41 PM	TXT File	86 KB
orientations_0.txt	11/2/2020 12:41 PM	TXT File	8 KB
output.txt	11/2/2020 12:41 PM	TXT File	16 KB
Output_FakeMatl_0.dream3d	11/2/2020 12:41 PM	DREAM3D File	1,271 KB
Output_FakeMatl_0.vtk	11/2/2020 12:41 PM	VTK File	712 KB
Output_FakeMatl_0.xdmf	11/2/2020 12:41 PM	XDMF File	3 KB
sub_band_info_0.p	11/2/2020 12:41 PM	MATLAB P-code	5,340 KB
sub_band_regions_0.inp	11/2/2020 12:41 PM	INP File	11,664 KB

Fig. 12. Screenshot of the files generated for each microstructure instantiation using the *generate_microstructure.py*. Each file is described in Table 2.

Table 2. List and description of files generated for each microstructure instantiation using *generate_microstructures.py*. The "#" indicates the instantiation number and is indexed at 0. Files with the ".p" extension indicate "pickle" files that are easily written and read by Python.

File name	Description
band_plane_normals_#.txt	For each grain, the direction normal to each of the four crystallographic slip planes.
band_sets_#.txt	Complete list of which elements belong to each band, indexed at 1.
Dream3D_microstructure_pipeline.json	A copy of the DREAM.3D pipeline executed by *generate_microstructures.py*. Certain filters are modified within Python (i.e., size and shape of instantiations) based on user setting in the main() function.
elements_per_band_#.txt	List of number of elements in each slip band for each grain, slip plane, and band layer number.
element_band_sets_#.p	List of elements in each band. Read by the *volume_average_FIPs.py* script to average FIPs over bands.
element_grain_sets_#.p	List of elements in each grain. Read by the *volume_average_FIPs.py* script to average FIPs over grains.
El_pos_0.p	List of each element's centroid. Read by the *compile_and_plot_FIPs.py* script to plot the highest FIPs as a function of their distance to the free-surface (only for simulations with one of three "face_bc" boundary condition set to "free" as described in Table 1). Only one instance is generated because element centroids are identical across different microstructures in a single folder.
FeatureData_FakeMatl_#.csv	Detailed list of grain information generated in DREAM.3D including centroids, volumes, number of voxels/elements, Euler angles, aspect ratios, etc.
grainID_#.txt	Specifies to which grain each element belongs. One of the two microstructure files required for CPFE simulation in PRISMS-Plasticity.
orientations_#.txt	Specifies grain orientations using Rodrigues vectors. The second microstructure file for CPFE simulation.
output.txt	Prints the DREAM.3D pipeline output to a text file for the last instantiation (see Region 6 in Fig. 4).
Output_FakeMatl_#.dream3d	Microstructure instantiation data stored in the ".dream3d" file format. This file is not used by subsequent Python scripts and is not required for CPFE simulations but is versatile and can be read by other .json DREAM.3D pipelines for further analysis.

Output_FakeMatl_#.xdmf	Generated alongside the ".dream3d" microstructure instantiation file in the "Write DREAM.3D Data File" filter. It allows the user to view the associated microstructure instantiation when opened in ParaView.
Output_FakeMatl_#.vtk	This file also allows the user to view the microstructure instantiation in ParaView but the .json pipeline filter entitled "Vtk Rectilinear Grid Exporter" can be modified to export additional data for visualization.
sub_band_info_#.p	List of elements in each sub-band region. Read by the *volume_average_FIPs.py* script to average FIPs over sub-band regions.
sub_band_regions_#.inp	Complete list of which elements belong to each sub-band region, indexed at 1.

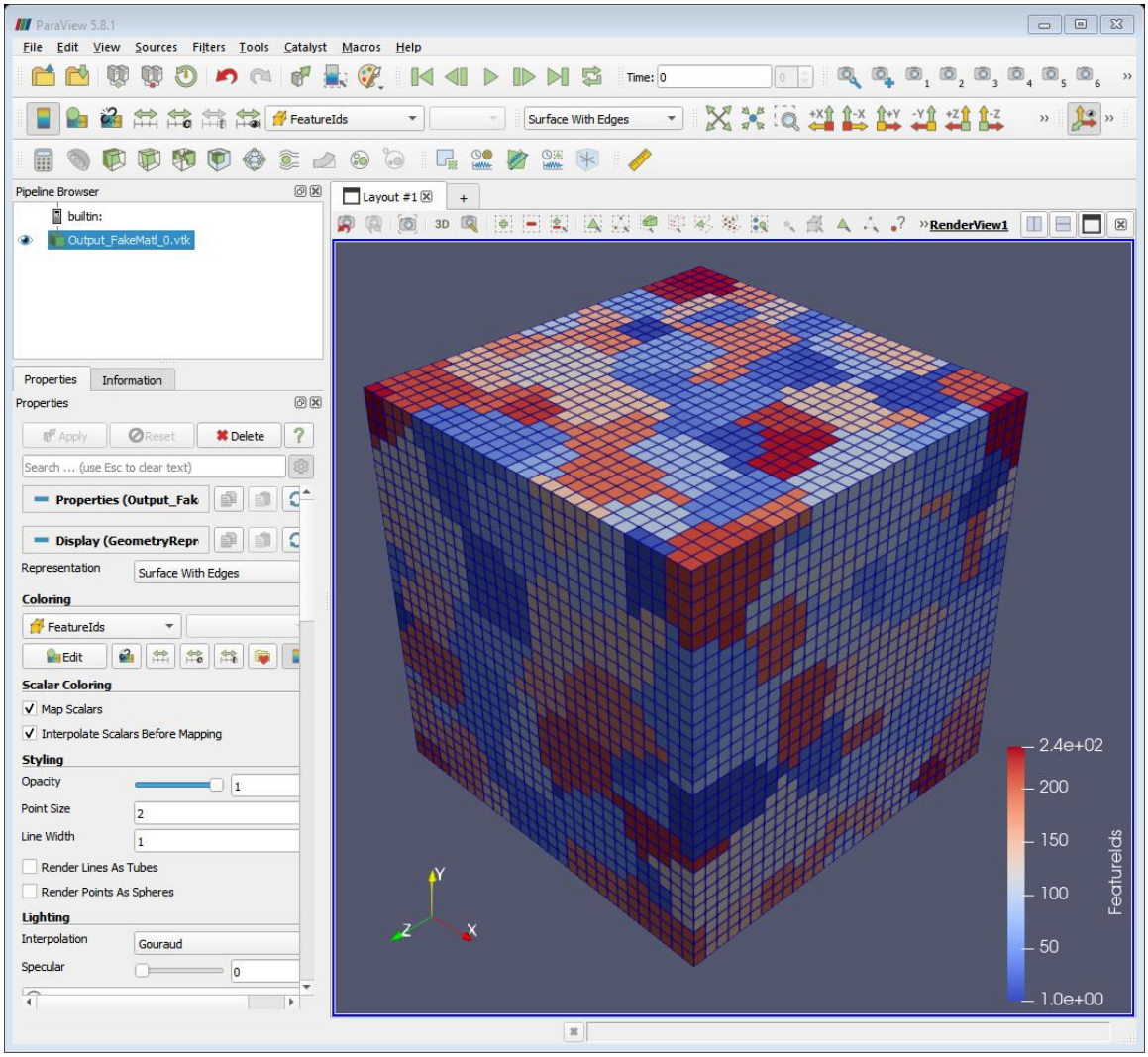


Fig. 13. Microstructure instantiation visualized in ParaView. Different colors indicate unique grains.

Crystal Plasticity Finite Element Method (CPFEM) Simulations

The files necessary to perform CPFEM simulations are available on the GitHub at this link: https://github.com/prisms-center/Fatigue/tree/main/src/MaterialModels. There are three .cc files specific to PRISMS-Fatigue that users must copy and paste into the PRISMS-Plasticity folders and then recompile PRISMS-Plasticity. More specifically, the three .cc files at the link above should replace the three .cc files with the same name in the PRISMS-Plasticity folder at: https://github.com/prisms-center/plasticity/tree/master/src/materialModels/crystalPlasticity, after which the package should be recompiled. This will ensure that the same quadrature outputs are written to the output .csv files and that users can arrive at identical results and figures.

Materials Commons

The entire dataset associated with the PRISMS-Fatigue manuscript is available for download at Materials Commons at this link: https://doi.org/10.13011/m3-rcyy-gx13. Users are encouraged to download these files to recreate the results and figures. There are four folders with microstructure and CPFE simulation results that correspond to the four applications described in the manuscript. Each folder contains the necessary prm_0.prm and other input files to perform the same CPFE simulations as described in the manuscript. The PRISMS-Fatigue videos describe this in great detail.

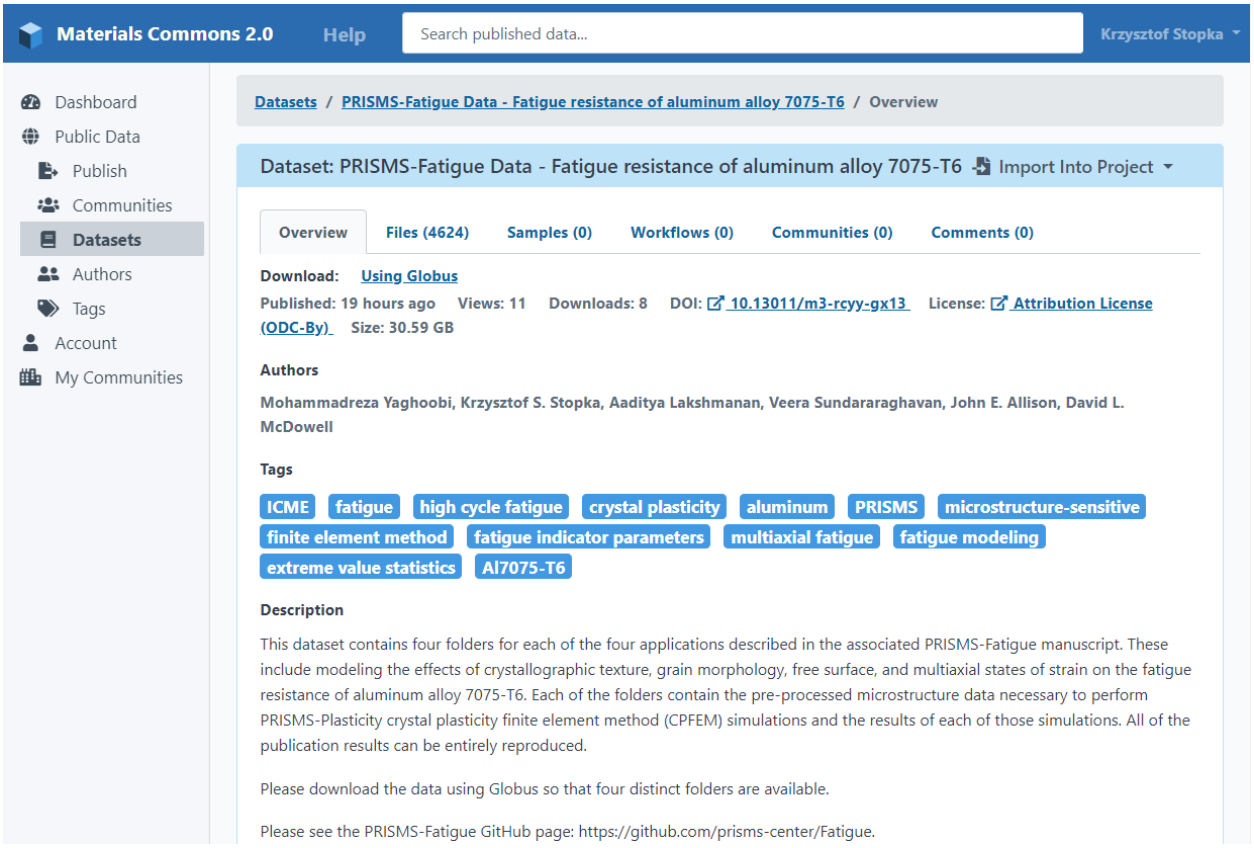


Fig. 14. Screenshot of PRISMS-Fatigue manuscript data available for public download.

calculate_FIPs.py

The second Python script calculates FIPs using simulation data exported from PRISMS-Plasticity. A screenshot of the main() function is shown in Fig. 15.

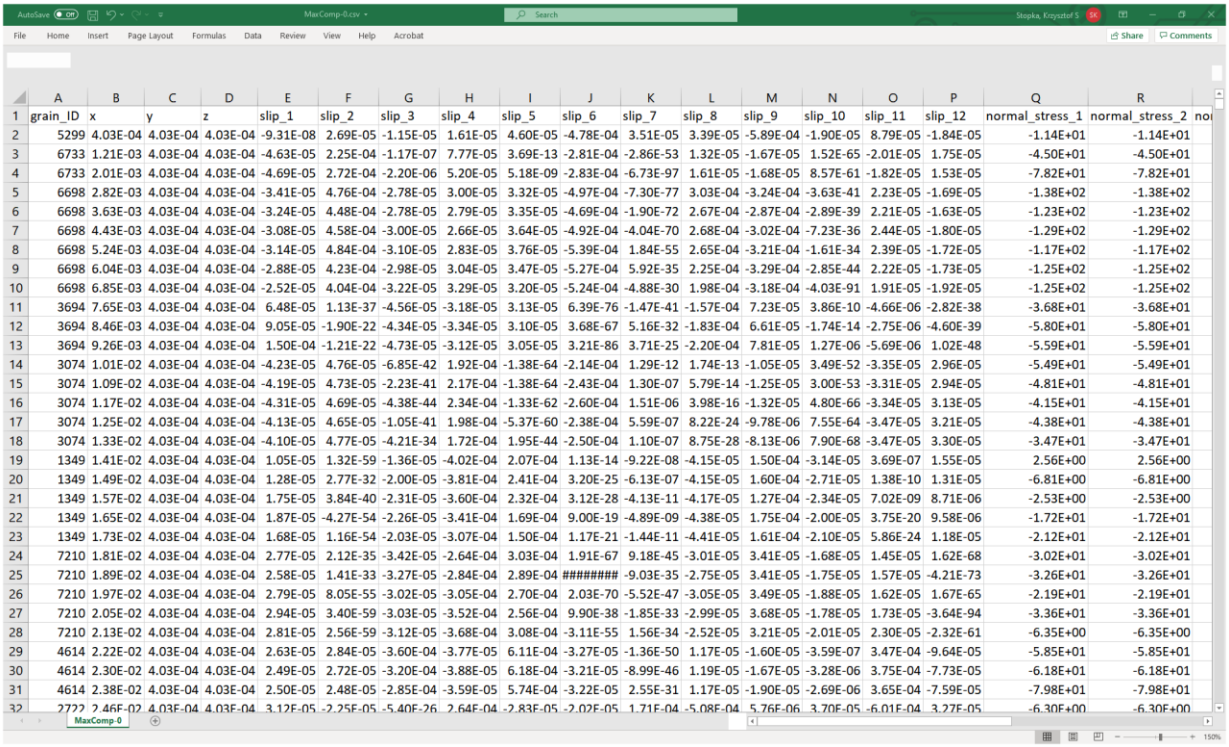
```
def main():

    # Directory where PRISMS-Plasticity results .csv files are stored
    # In the case of gamma plane simulations, this should contain all the folders numbered 0 thru (number_of_folder - 1),
    each of which contains some number of instantiations simulated at some combination of strain state and magnitude
    directory = os.path.dirname(DIR_LOC) + '\\tutorial\\MultiaxialFatigue_Al7075T6'

    # directory =
    r'C:\Users\stopk\Documents\GitHub\PRISMS-Fatigue\tutorial\Texture_Effect_Al7075T6\rolled_equiaxed_periodic'

    # Shape of microstructure instantiations (at this point, only CUBIC voxel functionality supported)
    # THIS MUST MATCH THE SHAPE FROM THE 'generate_microstructure.py' SCRIPT!
    shape = np.asarray([29,29,29])

    # Number of microstructure instantiations for which to calculate FIPs
    # THIS MUST MATCH THE SHAPE FROM THE 'generate_microstructure.py' SCRIPT! Otherwise, FIPs will not be computed for
    all instantiations!
    num_instantiations = 10

    # Specify the FIP to be calculated; default is "FS_FIP"
    # The other FIP that can be calculated uses the plastic shear strain range on each slip system; FIP_type =
    'plastic_shear_strain_range'
    FIP_type = 'FS_FIP'

    # Define the two Fatemi-Socie (FS) FIP parameters; Please see the references below for more information
    k_fip   = 10.0
    sigma_y = 517.0

    FIP_params = [k_fip, sigma_y]

    # For an fcc material, there are 12 slip systems for each element and therefore 12 FIPs per element
    num_slip_systems = 12

    # Specify whether this folder contain multiple instantiations to generate the multiaxial Gamma plane
    # This requires additional post-processing of PRISMS-Plasticity output files to calculate macroscopic plastic strain
    tensors
    gamma_plane_simulations = False

    # Specify whether to append the .vtk file generated by DREAM.3D to visualize the highest FIP per element
    vtk_visualize_FIPs = False

    # Number of multiaxial strain state and magnitude folders, i.e., strain states simulated
    # This is only required if calculating the necessary files to generate a multiaxial gamma plane
    num_gamma_plane_folders = 1

    if gamma_plane_simulations:

        # Iterate through folders containing result files from multiaxial simulations
        for jj in range(num_gamma_plane_folders):
            dirr = os.path.join(directory, str(jj))
            print(dirr)

            for ii in range(num_instantiations):
                # print('Instantiation number %d' % ii)
                import_PRISMS_data(dirr, shape, ii, FIP_type, num_slip_systems, FIP_params, gamma_plane_simulations)


    else:
        # Otherwise, compute FIPs for a single folder
        for ii in range(num_instantiations):
            import_PRISMS_data(directory, shape, ii, FIP_type, num_slip_systems, FIP_params, gamma_plane_simulations)

    if vtk_visualize_FIPs and not gamma_plane_simulations:
        for ii in range(num_instantiations):
            # Write FIPs to .vtk file for visualization
            append_FIPs_to_vtk(directory, shape, ii, FIP_type, num_slip_systems)
```

Fig. 15. Screenshot of the main() function in *calculate_FIPs.py*.

As described in the PRISMS-Fatigue manuscript, a particular FIP is employed to demonstrate the capabilities of these scripts. However, prospective users can define their own FIPs which may require new and unique CPFE simulation outputs.

One of the output files of PRISMS-Plasticity CPFE simulations is the "Quadrature Output" as described in the fourth video tutorial (https://www.youtube.com/watch?v=YqaMhRLzqu0). It contains local (i.e., for each element integration point) user-defined state variables. Fig. 16 shows a sample file as used by PRISMS-Fatigue with a header row to distinguish each column. The columns include grain number, position in the X, Y, and Z directions, the current value of plastic shear strain for the 12 slip systems, and the stress normal to each slip system slip plane. These state variables are reported because of the desired FIP formulation. If the user's FIP formulation requires different state variables, then the "updateAfterIncrement.cc" file must be updated accordingly. These .csv files are read by this script using the "pandas" Python package.

Any FIP formulation must carefully consider the point at which FIPs are calculated. In the default FIP considered here (i.e., the crystallographic version of the Fatemi-Socie FIP), state variables at the points of maximum applied compression and tension across the final fully reversed loading/straining cycle are considered. Thus, the *calculate_FIPs.py* script searches for files entitled *MaxComp_#.csv* and *MaxTen_#.csv* from which state variables are read. Prospective users will heavily modify the import_PRISMS_data() function in this script to define unique FIP formulations. These must be saved to the *PRISMS_*FIP_type*_FIPs_#.p* file for subsequent FIP volume averaging. The following script requires that each element contain a FIP for every slip system. If a FIP formulation does not consider unique slip systems and instead provides a single scalar value at each element, this value can simply be repeated.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	grain_ID	x	y	z	slip_1	slip_2	slip_3	slip_4	slip_5	slip_6	slip_7	slip_8	slip_9	slip_10	slip_11	slip_12	normal_stress_1	normal_stress_2
2	5299	4.03E-04	4.03E-04	4.03E-04	-9.31E-08	2.69E-05	-1.15E-05	1.61E-05	4.60E-05	-4.78E-04	3.51E-05	3.39E-05	-5.89E-04	-1.90E-05	8.79E-05	-1.84E-05	-1.14E+01	-1.14E+01
3	6733	1.21E-03	4.03E-04	4.03E-04	-4.63E-05	2.25E-04	-1.17E-07	7.77E-05	3.69E-13	-2.81E-04	-2.86E-53	1.32E-05	-1.67E-05	1.52E-65	-2.01E-05	1.75E-05	-4.50E+01	-4.50E+01
4	6733	2.01E-03	4.03E-04	4.03E-04	-4.69E-05	2.72E-04	-2.20E-06	5.20E-05	5.18E-09	-2.83E-04	-6.73E-97	1.61E-05	-1.68E-05	8.57E-61	-1.82E-05	1.53E-05	-7.82E+01	-7.82E+01
5	6698	2.82E-03	4.03E-04	4.03E-04	-3.41E-05	4.76E-04	-2.78E-05	3.00E-05	3.32E-05	-4.97E-04	-7.30E-77	3.03E-04	-3.24E-04	-3.63E-41	2.23E-05	-1.69E-05	-1.38E+02	-1.38E+02
6	6698	3.63E-03	4.03E-04	4.03E-04	-3.24E-05	4.48E-04	-2.78E-05	2.79E-05	3.35E-05	-4.69E-04	-1.90E-72	2.67E-04	-2.87E-04	-2.89E-39	2.21E-05	-1.63E-05	-1.23E+02	-1.23E+02
7	6698	4.43E-03	4.03E-04	4.03E-04	-3.08E-05	4.58E-04	-3.00E-05	2.66E-05	3.64E-05	-4.92E-04	-4.04E-70	2.68E-04	-3.02E-04	-7.23E-36	2.44E-05	-1.80E-05	-1.29E+02	-1.29E+02
8	6698	5.24E-03	4.03E-04	4.03E-04	-3.14E-05	4.84E-04	-3.10E-05	2.83E-05	3.76E-05	-5.39E-04	1.84E-55	2.65E-04	-3.21E-04	-1.61E-34	2.39E-05	-1.72E-05	-1.17E+02	-1.17E+02
9	6698	6.04E-03	4.03E-04	4.03E-04	-2.88E-05	4.23E-04	-2.98E-05	3.04E-05	3.47E-05	-5.27E-04	5.92E-35	2.25E-04	-3.29E-04	-2.85E-44	2.22E-05	-1.73E-05	-1.25E+02	-1.25E+02
10	6698	6.85E-03	4.03E-04	4.03E-04	-2.52E-05	4.04E-04	-3.22E-05	3.29E-05	3.20E-05	-5.24E-04	-4.88E-30	1.98E-04	-3.18E-04	-4.03E-91	1.91E-05	-1.92E-05	-1.25E+02	-1.25E+02
11	3694	7.65E-03	4.03E-04	4.03E-04	6.48E-05	1.13E-37	-4.56E-05	-3.18E-05	3.13E-05	6.39E-76	-1.47E-41	-1.57E-04	7.23E-05	3.86E-10	-4.66E-06	-2.82E-38	-3.68E+01	-3.68E+01
12	3694	8.46E-03	4.03E-04	4.03E-04	9.05E-05	-1.90E-22	-4.34E-05	-3.34E-05	3.10E-05	3.68E-67	5.16E-32	-1.83E-04	6.61E-05	-1.74E-14	-2.75E-06	-4.60E-39	-5.80E+01	-5.80E+01
13	3694	9.26E-03	4.03E-04	4.03E-04	1.50E-04	-1.21E-22	-4.73E-05	-3.12E-05	3.05E-05	3.21E-86	3.71E-25	-2.20E-04	7.81E-05	1.27E-06	-5.69E-06	1.02E-48	-5.59E+01	-5.59E+01
14	3074	1.01E-02	4.03E-04	4.03E-04	-4.23E-05	4.76E-05	-6.85E-42	1.92E-04	-1.38E-64	-2.14E-04	1.29E-12	1.74E-13	-1.05E-05	3.49E-52	-3.35E-05	2.96E-05	-5.49E+01	-5.49E+01
15	3074	1.09E-02	4.03E-04	4.03E-04	-4.19E-05	4.73E-05	-2.23E-41	2.17E-04	-1.38E-64	-2.43E-04	1.30E-07	5.79E-14	-1.25E-05	3.00E-53	-3.31E-05	2.94E-05	-4.81E+01	-4.81E+01
16	3074	1.17E-02	4.03E-04	4.03E-04	-4.31E-05	4.69E-05	-4.38E-44	2.34E-04	-1.33E-62	-2.60E-04	1.51E-06	3.98E-16	-1.32E-05	4.80E-66	-3.34E-05	3.13E-05	-4.15E+01	-4.15E+01
17	3074	1.25E-02	4.03E-04	4.03E-04	-4.13E-05	4.65E-05	-1.05E-41	1.98E-04	-5.37E-60	-2.38E-04	5.59E-07	8.22E-24	-9.78E-06	7.55E-64	-3.47E-05	3.21E-05	-4.38E+01	-4.38E+01
18	3074	1.33E-02	4.03E-04	4.03E-04	-4.10E-05	4.77E-05	-4.21E-34	1.72E-04	1.95E-44	-2.50E-04	1.10E-07	8.75E-28	-8.13E-06	7.90E-68	-3.47E-05	3.30E-05	-3.47E+01	-3.47E+01
19	1349	1.41E-02	4.03E-04	4.03E-04	1.05E-05	1.32E-59	-1.36E-05	-4.02E-04	2.07E-04	1.13E-14	-9.22E-08	-4.15E-05	1.50E-04	-3.14E-05	3.69E-07	1.55E-05	2.56E+00	2.56E+00
20	1349	1.49E-02	4.03E-04	4.03E-04	1.28E-05	2.77E-32	-2.00E-05	-3.81E-04	2.41E-04	3.20E-25	-6.13E-07	-4.15E-05	1.60E-04	-2.71E-05	1.38E-10	1.31E-05	-6.81E+00	-6.81E+00
21	1349	1.57E-02	4.03E-04	4.03E-04	1.75E-05	3.84E-40	-2.31E-05	-3.60E-04	2.32E-04	3.12E-28	-4.13E-11	-4.17E-05	1.27E-04	-2.34E-05	7.02E-09	8.71E-06	-2.53E+00	-2.53E+00
22	1349	1.65E-02	4.03E-04	4.03E-04	1.87E-05	-4.27E-54	-2.26E-05	-3.41E-04	1.69E-04	9.00E-19	-4.89E-09	-4.38E-05	1.75E-04	-2.00E-05	3.75E-20	9.58E-06	-1.72E+01	-1.72E+01
23	1349	1.73E-02	4.03E-04	4.03E-04	1.68E-05	1.16E-54	-2.03E-05	-3.07E-04	1.50E-04	1.17E-21	-1.44E-11	-4.41E-05	1.61E-04	-2.10E-05	5.86E-24	1.18E-05	-2.12E+01	-2.12E+01
24	7210	1.81E-02	4.03E-04	4.03E-04	2.77E-05	2.12E-35	-3.42E-05	-2.64E-04	3.03E-04	1.91E-67	9.18E-45	-3.01E-05	3.41E-05	-1.68E-05	1.45E-05	1.62E-68	-3.02E+01	-3.02E+01
25	7210	1.89E-02	4.03E-04	4.03E-04	2.58E-05	1.41E-33	-3.27E-05	-2.84E-04	2.89E-04	########	-9.03E-35	-2.75E-05	3.41E-05	-1.75E-05	1.57E-05	-4.21E-73	-3.26E+01	-3.26E+01
26	7210	1.97E-02	4.03E-04	4.03E-04	2.79E-05	8.05E-55	-3.02E-05	-3.05E-04	2.70E-04	2.03E-70	-5.52E-47	-3.05E-05	3.49E-05	-1.88E-05	1.62E-05	1.67E-65	-2.19E+01	-2.19E+01
27	7210	2.05E-02	4.03E-04	4.03E-04	2.94E-05	3.40E-59	-3.03E-05	-3.52E-04	2.56E-04	9.90E-38	-1.85E-33	-2.99E-05	3.68E-05	-1.78E-05	1.73E-05	-3.64E-94	-3.36E+01	-3.36E+01
28	7210	2.13E-02	4.03E-04	4.03E-04	2.81E-05	2.56E-59	-3.12E-05	-3.68E-04	3.08E-04	-3.11E-55	1.56E-34	-2.52E-05	3.21E-05	-2.01E-05	2.30E-05	-2.32E-61	-6.35E+00	-6.35E+00
29	4614	2.22E-02	4.03E-04	4.03E-04	2.63E-05	2.84E-05	-3.60E-04	-3.77E-05	6.11E-04	-3.27E-05	-1.36E-50	1.17E-05	-1.60E-05	-3.59E-07	3.47E-04	-9.64E-05	-5.85E+01	-5.85E+01
30	4614	2.30E-02	4.03E-04	4.03E-04	2.49E-05	2.72E-05	-3.20E-04	-3.88E-05	6.18E-04	-3.21E-05	-8.99E-46	1.19E-05	-1.67E-05	-3.28E-06	3.75E-04	-7.73E-05	-6.18E+01	-6.18E+01
31	4614	2.38E-02	4.03E-04	4.03E-04	2.50E-05	2.48E-05	-2.85E-04	-3.59E-05	5.74E-04	-3.22E-05	2.55E-31	1.17E-05	-1.90E-05	-2.69E-06	3.65E-04	-7.59E-05	-7.98E+01	-7.98E+01
32	7722	2.46E-02	4.03E-04	4.03E-04	3.12E-05	-2.25E-05	-5.40E-26	2.64E-04	-2.83E-05	-2.02E-05	1.71E-04	-5.08E-04	5.76E-06	3.70E-05	-6.01E-04	3.27E-05	-6.30E+00	-6.30E+00

Fig. 16. Sample PRISMS-Plasticity Quadrature output file. The header row was added manually and is not required by *calculate_FIPS.py*. Values are instead read in by column number.

Table 3. List of variables and settings in the *calculate_FIPs.py* script.

Variable/setting (*type*)	Description
directory *Path to folder*	Path to folder with microstructure data and simulation results. Like the previous script, this can be defined with a relative or absolute path.
shape *integer array*	Number of voxels in the X, Y, and Z directions, respectively.
num_instantiations *integer*	Number of microstructure instantiations for which FIPs should be calculated. This should match the number specified in *generate_microstructure.py*.
FIP_type *string*	The type of FIP that should be computed for each voxel/element using the CPFE simulation output data. The default is "FS_FIP" which calculates a crystallographic version of the Fatemi-Socie FIP. Another option included in the initial release of PRISMS-Fatigue is "plastic_shear_strain_range" that considers solely the plastic shear strain range on each slip system at every element. This script will be updated with other types of FIPs in the future. The user is encouraged to define their own FIPs.
k_fip *float*	Parameter to calculate "FS_FIP"; controls the influence of the stress normal to the slip plane on the FIP magnitude.
sigma_y *float*	Parameter to calculate "FS_FIP"; yield strength of the material system modeled; normalizes the influence of the stress normal to the slip plane.
num_slip_systems *integer*	Number of slip systems in the material of interest. The FS_FIP is calculated for each slip system but other FIP definitions may not require this or the previous two variables (i.e., k_fip and sigma_y).
gamma_plane_simulations *Boolean*	Specifies whether the FIPs to be calculated will be used for a multiaxial gamma plane [4] (explained further in the *gamma_plane.py* section).
vtk_visualize_FIPs *Boolean*	Specifies whether the highest elemental FIP per element should be appended to the .vtk file for visualization of FIPs in ParaView.
num_gamma_plane_folders *integer*	Number of unique simulation folders to populate the multiaxial gamma plane.

Table 4. List and description of files generated for each microstructure by *calculate_FIPs.py*. The "#" indicates the instantiation number and is indexed at 0. Files with the ".p" extension indicate "pickle" files that are easily written and read by Python.

File name	Description
PRISMS_*FIP_type*_FIPs_#.p	List of FIPs for each element. Read by the volume_average_FIPs.py script to average FIPs over grains, bands, or sub-band regions.

volume_average_FIPs.py

The third Python script averages FIPs over one of three types of volumes: grains, bands, or sub-band regions. Fig. 17 depicts each type of volume. A screenshot of the main() function is shown in Fig. 18 and each user setting is described in Table 5.

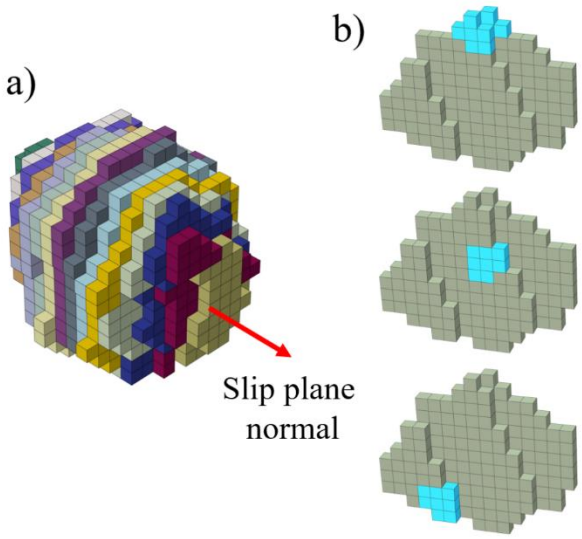


Fig. 17. (a) A single grain with slip bands representative of crystallographic slip planes. (b) A single slip band from the grain in (a) in which different sub-band regions comprised of eight elements each (controlled by the variable *num_vox* in *generate_microstructures.py*) are highlighted.

```python
def main():

    # Directory where microstructure FIP .p files are stored:
    directory = os.path.dirname(DIR_LOC) + '\\tutorial\\MultiaxialFatigue_Al7075T6'

    # directory = r'C:\Users\stopk\Documents\GitHub\PRISMS-Fatigue\tutorial\test_run_1'

    # Number of microstructure instantiations in each folder for which to volume average FIPs
    num_instantiations = 10

    # Specify the FIP to be volume averaged
    FIP_type = 'FS_FIP'

    # Specify type of FIP averaging: 'sub_band', 'band', or 'grain'
    averaging_type = 'sub_band'

    # Specify whether this folder contain multiple instantiations to generate the multiaxial
    Gamma plane
    gamma_plane_simulations = False

    # Number of multiaxial strain state and magnitude folders, i.e., strain states simulated
    # This is only required if calculating necessary files to generate multiaxial gamma plane
    num_gamma_plane_folders = 1

    if gamma_plane_simulations:
        # Iterate through folders containing result files from multiaxial simulations
        for jj in range(num_gamma_plane_folders):
            dirr = os.path.join(directory, str(jj))
            print(dirr)

            for ii in range(num_instantiations):
                call_averaging(ii, dirr, FIP_type, averaging_type, gamma_plane_simulations)

    else:
        # Call function to perform averaging for each instantiation in folder
        for ii in range(num_instantiations):
            call_averaging(ii, directory, FIP_type, averaging_type, gamma_plane_simulations)
```

Fig. 18. Screenshot of the main() function in *volume_average_FIPs.py*.

Table 5. List of variables and settings in the *volume_average_FIPs.py* script.

Variable/setting (*type*)	Description
directory *path to folder*	Path to folder where all microstructure instantiation data and simulation results are stored. Like the previous script, this can be defined with a relative or absolute path.
num_instantiations *integer*	Number of microstructure instantiations for which FIPs should be volume averaged.
FIP_type *string*	The type of FIP that should be volume averaged. **NOTE:** The desired FIP that should be volume averaged should have been calculated using the previous script!
averaging_type *string*	Desired type of FIP volume averaging. Options include "sub_band", "band", and "grain".
gamma_plane_simulations *Boolean*	Specifies whether the FIPs to be calculated will be used for a multiaxial gamma plane [4] (explained further in the *gamma_plane.py* section).
num_gamma_plane_folders *integer*	Number of unique simulation folders to populate the multiaxial gamma plane.

The "averaging_type" specified by the user determines which of the three pairs of files listed in Table 6 will be generated. The pickle files are read by subsequent scripts whereas the .csv files allow the user to quickly read through the averaged FIPs. Microstructures with a very large number of elements may produce excessively large files. The user may modify the scripts to suppress generation of certain files (e.g., .csv files) to reduce memory usage. The user may also run this script three times and average FIPs over all three types of volumes.

Table 6. List and description of files generated for each microstructure by *volume_average_FIPs.py*. The "#" indicates the instantiation number and is indexed at 0. Files with the ".p" extension indicate "pickle" files that are easily written and read inside Python.

File name	Description
sub_band_averaged_*FIP_type*_pickle_#.p	List of sub-band averaged FIPs. Read by *compile_and_plot_FIPs.py* and *gamma_plane.py*.
sub_band_averaged_*FIP_type*_#.csv	List of sub-band averaged FIPs written to readable .csv file.
band_averaged_*FIP_type*_pickle_#.p	List of band averaged FIPs. Read by *compile_and_plot_FIPs.py*.
band_averaged_*FIP_type*_#.csv	List of band averaged FIPs written to readable .csv file.
grain_averaged_*FIP_type*_pickle_#.p	List of grain averaged FIPs. Read by *compile_and_plot_FIPs.py*.
grain_averaged_*FIP_type*_#.csv	List of grain averaged FIPs written to readable .csv file.

compile_and_plot_FIPs.py

The fourth Python script compiles and plots FIPs from different simulation batch folders. Two types of figures are supported. The first fits the highest FIPs to an EVD. The second plots FIPs as a function of their distance to the microstructure boundary (i.e., to the free surface for "thin sheet" simulations as described previously). Fig. 19 depicts a screenshot of the variables in the main() function which are described in Table 7.

```
def main():
    # Specify type of analysis: either 'fip_evd' to plot FIP extreme value distributions (EVDs)
    or 'surf_dist' to plot the highest FIPs as a function of their distance to the free surface
    analysis_type = 'fip_evd'

    # Specify folder which contains all of the simulation batch folders
    # IMPORTANT: this directory should contain the locations specified at the top of this python
    script, NOT the individual folders with the instantiations!
    # This script goes through EACH ONE of the folders specified and extracts the relevant
    information for plotting purposes

    directory = os.path.dirname(DIR_LOC) + '\\tutorial'

    # directory = r'C:\Users\stopk\Documents\GitHub\PRISMS-Fatigue\tutorial\test_run_1'

    # Specify which 'material' to plot, i.e., which combination of microstructure folders
    # Please see the top of the "plot_FIPS" function
    mat = 'prisms_compare_bcs'

    # Specify which FIP to import
    FIP_type = 'FS_FIP'

    # Specify which volume averaging scheme to import. By default, the sub-band averaged (SBA)
    FIPs are used
    # IMPORTANT: this will fail if the FIPs have not yet been averaged in the desired manner
    (see 'volume_average_FIPs.py' script)
    # Options: 'sub_band', 'band', and 'grain'
    averaging_type = 'sub_band'

    if analysis_type == 'fip_evd':

        # Specify how many of the highest FIPs should be plotted
        n_fip_plot = 50

        # Specify whether FIPs should be fit to the "gumbel" or "frechet" EVD
        plt_type = 'gumbel'

        # Call function to plot FIPs
        plot_FIPS(directory, plt_type, mat, n_fip_plot, FIP_type, averaging_type, save_fig = True)

    elif analysis_type == 'surf_dist':
        # Plot SBA FIP vs distance from free surface:

        # Specify how many of the highest FIPs should be plotted
        n_fip_plot = 10

        # Specify the non-periodic direction (i.e., the set of parallel faces set to
        traction-free conditions)
        # In the references associated with these scripts (see below), this is set to the 'Y'
        direction
        non_periodic_dir = 'Y'

        # Specify the size/length of the SVE in the non-periodic direction in mm, to properly
        locate the FIPs within the microstructure
        SVE_non_periodic_length = 0.0725

        # Call function to plot FIPs
        plot_EVD_FIP_distance(directory, mat, n_fip_plot, FIP_type, averaging_type,
        non_periodic_dir, SVE_non_periodic_length)
```

Fig. 19. Screenshot of the main() function in *compile_and_plot_FIPs.py*.

Table 7. List of variables and settings in the main() function in *compile_and_plot_FIPs.py*.

Variable/setting (*type*)	Description
analysis_type *string*	Type of plot to generate. Options include "fip_evd" and "surf_dist" and correspond to FIP EVDs and FIPs plotted as a function of distance to the microstructure boundary, respectively.
directory *path to folder*	**NOTE:** This directory should be different than previously defined directories and will store the resultant plots and other generated files. The previous scripts defined the directory as the location with microstructure and simulation data. In contrast, this script will read FIP data from locations specified at the top of the script, as show in Fig. 20.
mat *string*	This specifies additional settings in the plot_FIPs() and plot_EVD_FIP_distance() functions including figure legend labels, marker shapes and colors, number of columns in the figure legend, etc., and where all FIP data is stored ("locs" variable).
FIP_type *string*	The type of FIP that should be plotted. **NOTE:** An error will occur if the user specifies a FIP for which volume averaged files do not exist.
averaging_type *string*	The type of FIP that should be read, compiled, and used in figures. Options include "sub_band", "band", and "grain". **NOTE:** An error will occur if the user has not averaged FIPs in the previous script in the same manner as desired by this script.
n_fip_plot *integer*	Number of FIPs to plot for either type of figure. The default is 50 and 10 for FIP EVDs and FIP vs. distance to microstructure boundary figures, respectively.
plt_type *string*	Specifies whether FIPs should be fit to the Gumbel or Fréchet EVD with options as "gumbel" and "frechet", respectively.
non_periodic_dir *string*	Specifies which pair of microstructure instantiation faces are non-periodic (i.e., free surfaces as set in generate_microstructure.py; option 3 for "face_bc").
SVE_non_periodic_length *float*	The length of the microstructure in the non-periodic direction. Required to correctly calculate distance of the highest FIPs to the free surface or instantiation boundary.

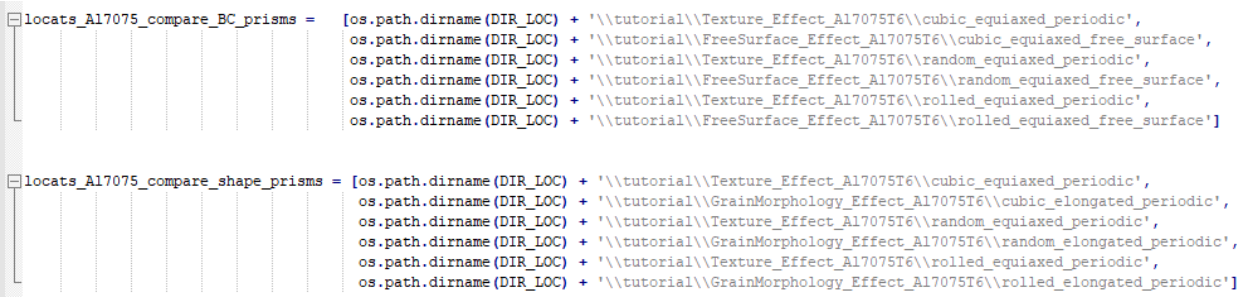


Fig. 20. Screenshot of the top of *compile_and_plot_FIPs.py* where the paths of individual simulation folder are specified.

This script is unique in that users must also modify another function entitled plot_FIPs() and/or plot_EVD_FIP_distance(). The "mat" variable specified by users in the main() function determines which combination of individual simulation batch folders to plot. As an example, the screenshot in Fig. 21 shows two instances of "mat" that plot and compare FIP EVDs that vary in boundary condition (free surface vs. fully periodic CPFE microstructure and boundary conditions, mat = "*prisms_compare_bcs*") and grain morphology (equiaxed vs. elongated, mat = "*prisms_compare_shape*") as described in the PRISMS-Fatigue manuscript. The variable assignments under a specific "mat" definition control figure properties (e.g., name in legend, marker styles, etc.) and the location of each folder as indicated in Fig. 20.

At this point, the "surf_dist" option only supports sub-band averaged FIPs. This is because attempting to locate the grains with the highest grain averaged FIPs with respect to the microstructure boundaries may too excessively smear interpretation of this plot. More specifically, it makes more sense to locate the sub-band region with the highest FIP with respect to the microstructure boundary because it more accurately reflects the fact that fatigue crack formation occurs at the sub-grain scale, and not across an entire grain instantaneously. Even the grain which contains the highest grain averaged FIP may have regions which undergo little local plastic deformation.

```
def plot_FIPS(base_directory, plt_type, mat, num_fips_plot, fip_type, save_fig = True):
    # Function to plot FIPs from bulk (i.e., fully periodic) and surface (i.e., traction-free/free
    surface) simulations, or other types of simulations
    print('Plotting volume averaged FIPs')
    os.chdir(base_directory)

    # Specify which FIPS to plot

    # This will plot FIPs from subsurface/bulk and free-surface simulations
    if mat == 'prisms_compare_bcs':
        # Names to use for figure legend
        # These must match the order of simulation folders specified at the top of this script
        names = ["Cubic Periodic","Cubic Free Surface","Random Periodic","Random Free Surface","Rolled
        Periodic","Rolled Free Surface"]

        # Locations of actual FIP files
        locs = locats_Al7075_compare_BC_prisms

        # Number of columns for figure legend
        plot_col = 1

        # Marker colors
        cfm = ['r','r','b','b','g','g']

        # Marker shapes
        sfm = ['o','o','s','s','^','^']

        # If set to true, FIPs from periodic and free surface simulations will be hollow and full,
        respectively
        hollow_and_full = True

    # This will plot FIPs from fully periodic simulations with different crystallographic textures and
    grain morphologies
    elif mat == 'prisms_compare_shape':
        # Names to use for figure legend
        names = ["Cubic Equiaxed","Cubic Elongated","Random Equiaxed","Random Elongated","Rolled Equiaxed",
        "Rolled Elongated"]

        # Locations of actual FIP files
        locs = locats_Al7075_compare_shape_prisms

        # Number of columns for figure legend
        plot_col = 1

        # Marker colors
        cfm = ['r','r','b','b','g','g']

        # Marker shapes
        sfm = ['o','o','s','s','^','^']

        # If set to true, FIPs from periodic and free surface simulations will be hollow and full,
        respectively
        hollow_and_full = True
```

Fig. 21. Screenshot of the top of the plot_FIPs() function in *compile_and_plot_FIPs.py*.

Several files are generated by this script, as shown in Fig. 22 and described in Table 8. The pickle ".p" files are generated to increase efficiency. Each folder may contain dozens of microstructures and extracting FIPs from each volume averaged pickle file may take considerable time, especially for those with many elements and for the more complex sub-band averaged FIPs. Therefore, the highest FIPs per grain for all microstructures in a single folder are stored into a single array which is then stored in the *compiled_*.p* files shown in this folder. The script first checks whether these files already exist and if this is not true, it generates them by reading the appropriate FIP files. Once the files are present, they are quickly read to produce plots. This allows the user to rapidly change figure properties and spend more time interpreting data.

For the first type of plot, users can specify whether FIPs should be fit to the Gumbel or Fréchet EVD and plots are saved with corresponding file names. A .csv file contains properties of the best fit line for each plot along with the value of the largest FIP. By default, the highest 50 FIPs from each data set are fit to the EVD. The second type of figure by default plots the 10 highest FIPs as a function of their distance to the instantiation boundary. An associated text file contains the average distance of these FIPs to the instantiation boundary.

Table 8. List and description of files generated by *compile_and_plot_FIPs.py*.

File name	Description
compiled_*FIP_type*_*averaging_type* _*folder*.p	List of compiled FIPs for each simulation folder.
compiled_surf_dist_*FIP_type* _*averaging_type**folder*.p	List of compiled FIPs and distance to microstructure boundary for each simulation folder.
r_squared_*plt_type*_*FIP_type* _*averaging_type*_* n_fip_plot *_*mat*.csv	Line of best fit properties of FIPs fit to EVD.
*FIP_type*_EVD_* averaging_type * _*plt_type*_*n_fip_plot*_*mat*.png	Plot of FIPs fit to either Gumbel or Fréchet EVD.
Avg_surf_dist_for_top_*n_fip_plot*_*mat.txt	Average distance of FIPs to microstructure boundary.
SBA_FIP_surface_dist_*n_fip_plot*_*mat*.png	Plot of FIPs vs. distance to microstructure boundary.

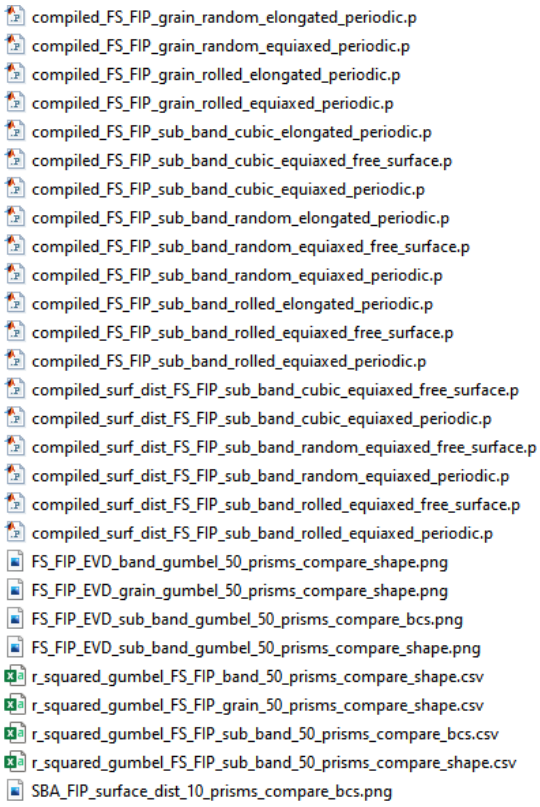


Fig. 22. Folder structure to execute *compile_and_plot_FIPs.py* and output files. See Fig. 20.

gamma_plane.py

The fifth and final Python script creates a multiaxial gamma (Γ) plane. It requires a single, large microstructure or a set of microstructures simulated under multiple combinations of strain states (e.g., uniaxial, biaxial, and pure shear) and magnitudes [4]. A screenshot of the main() function is shown in Fig. 23 and Table 9 describes each input variable.

```
def main():

    # Specify name of directory with simulation folders, each with the same X number of
    microstructures simulated to some combination of strain state and magnitude, and
    numbered 0 thru (number_of_folder - 1)
    directory = os.path.dirname(DIR_LOC) + '\\tutorial\\MultiaxialFatigue_Al7075T6'

    # Specify the number of FIPs to extract from each simulation folder
    num_FIPs_extract = 50

    # Specify how many of the highest FIPs from each batch of simulations should be
    considered to plot the ISO-FIP contours
    num_read_top_FIPs = 10

    # Specify which FIP to import
    FIP_type = 'FS_FIP'

    # Specify type of FIP averaging: 'sub_band', 'band', or 'grain'
    averaging_type = 'sub_band'

    # Plot the locations of each simulation batch folder in terms of only the response
    coordinates (x and y position on the gamma plane)
    plot_gamma_plane_locations(directory)

    # This function will extract the 50 highest fips from each folder and store these in
    a single pickle file.
    # This function is run once to speed up subsequent analysis and plotting
    get_gamma_FIPs(directory, num_FIPs_extract, FIP_type, averaging_type)

    # Plot the gamma plane
    plot_gamma(directory, num_read_top_FIPs, FIP_type, averaging_type)
```

Fig. 23. Screenshot of the main() function in *gamma_plane.py*.

Table 9. List of variables and settings in *gamma_plane.py*.

Variable/setting (*type*)	Description
directory *path to folder*	Path to directory which contains folders numbered 0 thru (number of total simulated strain states and magnitudes). Each of these numbered folders must contain pairs of result .csv files as described previously.
num_read_top_FIPs *integer*	The amount of the highest volume averaged FIPs that should be used to calculate the ISO-FIP contours for the gamma plane (i.e., the number of FIPs that should be averaged and used in the Gaussian Process Regression (GPR) model to interpolate across the Gamma plane).
FIP_type *string*	The type of FIP that should be considered. **NOTE:** An error will occur if the user specifies a FIP for which volume averaged files do not exist.
averaging_type *string*	The type of FIP that should be read, compiled, and used in figures. **NOTE:** An error will occur if the user has not averaged FIPs in the previous script in the same manner as desired by this script.

The steps required to generate the gamma plane are as follows:

1) Generate a single large microstructure or an ensemble of microstructures using the *generate_microstructures.py* script.
2) Simulate these at various combinations of strain states and magnitudes.
    a. The download from Materials Commons includes an excel sheet that lists all the combinations of strain states and magnitudes that were used to populate the Gamma plane presented in the PRISMS-Fatigue manuscript. This download also includes the necessary prm_0.prm and microstructure file to apply the appropriate boundary/simulation conditions to the 10 microstructures.
3) Calculate FIPs for each microstructure in each simulation folder numbered 0 thru (number of simulated strain states) using the *calculate_FIPs.py* script.
4) Volume average FIPs over the desired volume using the *volume_average_FIPs.py* script.
5) Execute the *gamma_plane.py* script.

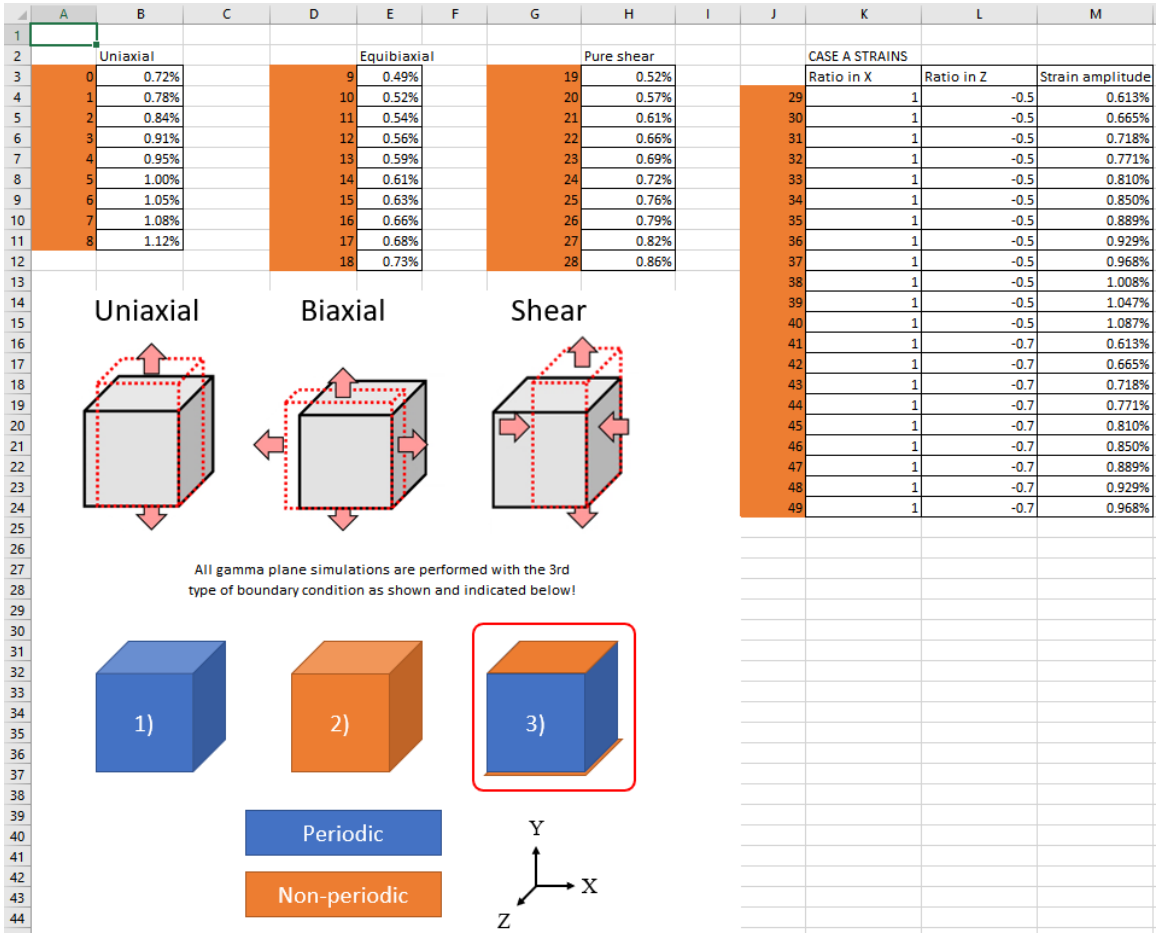


Fig. 24. Screenshot of the excel sheet included in the Materials Commons dataset that specifies each combination of strain state and magnitude used to generate the Gamma plane for Al7075-T6.

Sample tutorial

This playlist: https://www.youtube.com/playlist?list=PL4yBCojM4Swo3CvlA57syFrzk3p1mugP5 guides users through the entire framework to reproduce the results and figures in the PRISMS-Fatigue manuscript.

Useful links:

GitHub Repository: https://github.com/prisms-center/Fatigue

Materials Commons Dataset: https://doi.org/10.13011/m3-rcyy-gx13

PRISMS-Plasticity Forum: https://groups.google.com/g/prisms-cpfe-users

PRISMS Center: http://www.prisms-center.org/#/home

Support/current developers:

Krzysztof S. Stopka: stopka.kris@gmail.com, kstopka3@gatech.edu

Mohammadreza Yaghoobi: Yaghoobi@umich.edu

References

[1] M. A. Groeber and M. A. Jackson, "Dream.3d: A digital representation environment for the analysis of microstructure in 3d," *Integrating Materials and Manufacturing Innovation,* 3, 56 (2014)

[2] K. S. Stopka, T. Gu, and D. L. McDowell, "Effects of algorithmic simulation parameters on the prediction of extreme value fatigue indicator parameters in duplex ti-6al-4v," *International Journal of Fatigue,* 141, 105865 (2020)

[3] K. S. Stopka and D. L. McDowell, "Microstructure-sensitive computational estimates of driving forces for surface versus subsurface fatigue crack formation in duplex ti-6al-4v and al 7075-t6," *JOM,* 72, 28 (2020)

[4] K. S. Stopka and D. L. McDowell, "Microstructure-sensitive computational multiaxial fatigue of al 7075-t6 and duplex ti-6al-4v," *International Journal of Fatigue,* 133, 105460 (2020)