MethodsX 9 (2022) 101763

Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/mex

Method Article

A method to extract slip system dependent information for crystal plasticity models ☆

Dylan Agius [a,*], Abdullah Al Mamun [a,b], Christopher Truman [a], Mahmoud Mostafavi [a], David Knowles [a]

[a] Solid Mechanics Research Group, Department of Mechanical Engineering, University of Bristol, United Kingdom
[b] Nuclear Futures Institute, Bangor University, Gwynedd LL57 2DG, United Kingdom

A B S T R A C T

A tool to implement a length scale dependency to classical crystal plasticity simulations is presented. Classical crystal plasticity models do not include a size effect; therefore, the size of the grain does not influence the simulated deformation. Classical crystal plasticity advancements have been through the inclusion of stress or strain gradient based constitutive models to improve the simulation of length scale dependent deformation. However, this tool presents an alternative to implementing a length scale, where the influence of slip pile-up in the form of dislocations at grain boundaries as a potential to explaining the Hall-Petch effect in materials. This is achieved by calculating the slip distance in adjacent grains for each slip system, by assuming the total slip length spans the grain in the slip direction. These calculations can occur in two ways. The first is the analysis occurs at the start of the simulation, therefore, only occurs once. If this approach is used, the computational cost of this tool is minute. However, if the simulations consider large deformations, during which it is expected that the grains are going to undergo large rotations, then it would be advantageous to the have the tool recalculate the information during the analysis. Consequently, the computational cost would depend on the resolution of the modelled geometry, the number of grains, and the number of slip systems. The tool also provides a capability to develop constitutive models based on complex grain boundary features which can be implemented in classical crystal plasticity models and gradient based crystal plasticity models. The described calculation process is implemented through a Fortran subroutine, which has been designed to be easily used in crystal plasticity simulations. The presented tool also includes Python code designed to link with microstructures built using DREAM.3D to extract the required input data to the Fortran subroutine.

The proposed tool is not limited to classical crystal plasticity formulations, instead the data extracted and outputted from the Fortran subroutine can be used to serve alternative purposes in both stress and strain gradient crystal plasticity models.

The proposed tool can be modified to extract additional data to that presented.

The slip distance in the adjacent grain, the distance from the grain boundary of the current calculation point, and the interaction between slip systems between grains can be used in any crystal plasticity constitutive models.

☆ **Direct Submission or Co-Submission**: Co-submissions are papers that have been submitted alongside an original research paper accepted for publication by another Elsevier journal

DOI of original article: 10.1016/j.ijplas.2022.103249

* Corresponding author.
E-mail address: dylan.agius@bristol.ac.uk (D. Agius).

https://doi.org/10.1016/j.mex.2022.101763

ARTICLE INFO
Method name: LengMorph: a tool to add a length scale dependence to crystal plasticity simulations.
Keywords: Crystal plasticity, Grain size effect, Slip distance, Grain boundary, Misorientation, Slip system interaction
Article history: Received 24 February 2022; Accepted 15 June 2022; Available online 20 June 2022

Specifications table

Subject Area:	Materials Science
More specific subject area:	*A tool to implement a length scale dependency to crystal plasticity simulations*
Method name:	*LengMorph: a tool to add a length scale dependence to crystal plasticity simulations.*
Name and reference of original method:	*N.A.*
Resource availability:	*The Fortran subroutine and Fortran example program files will be made available in the supplementary documentations. All other materials can be found from the following GitHub repository:* https://github.com/DylanAgius/LengMorph.git.

Method details

In the following sections, details of an algorithm to implement a length scale dependency to classical crystal plasticity models is outlined in detail. The approach is based on calculating slip distances for each slip system at a voxel/element using the geometry and orientation of the closest adjacent grain. The proposed approach is based on the theory of the underlying mechanisms of the Hall-Petch effect being associated with dislocation pile-ups at grain boundaries [2–9]. The simulated incompatibility between grains is modified through the critical resolved shear stress (CRSS) to reflect the influence of pile-ups in adjacent grains. It must be noted that the current tool has been generalised to voxel-based representative volume elements (RVE); therefore, it cannot currently be used with tetrahedral discretised RVEs.

In the present work the code used to construct the necessary input data (in Python [17]) with the code used to calculate the slip distance (in Fortran) are outlined. An additional output of the Fortran code is the Luster-Morris parameter [11] which can also be used in crystal plasticity constitutive models as demonstrated in [1]. RVEs employing the length scale size dependence can be synthetically built using DREAM.3D [10], with details on the DREAM.3D constructed grains fed into the Python code. The integration of these codes is schematically shown in Fig. 1.

The specifics of how the length scale dependent subroutine presented here integrates with classical crystal plasticity theory can be found in [1]. The tools presented facilitates implementation in both finite element methods and spectra methods based on Fast Fourier Transforms. All materials needed to implement the proposed approach can be found at [12], which includes a DREAM.3D pipeline to construct the required information to be fed into the Python program used to create the input data and a standalone Fortran program which demonstrates how the Fortran code functions.

Extracting and processing grain data (DREAM.3D and python)

One effective tool to create RVEs for different microstructures is DREAM.3D. Using available filters in the DREAM.3D pipeline, information on the features of the generated grains can be extracted. This includes the distance of each voxel to the grain boundary, the centroid, and orientation of each grain. An example minimum filter pipeline to create an RVE is provided in Fig. 2, which includes additional filters used to extract information needed as inputs to the Python code in the present work. This example DREAM.3D pipeline can be found at [12].

The DREAM.3D extracted data is used in the Python code to construct matrices containing information of nodes on the boundary of every grain in the RVE as visualised in Fig. 3. This

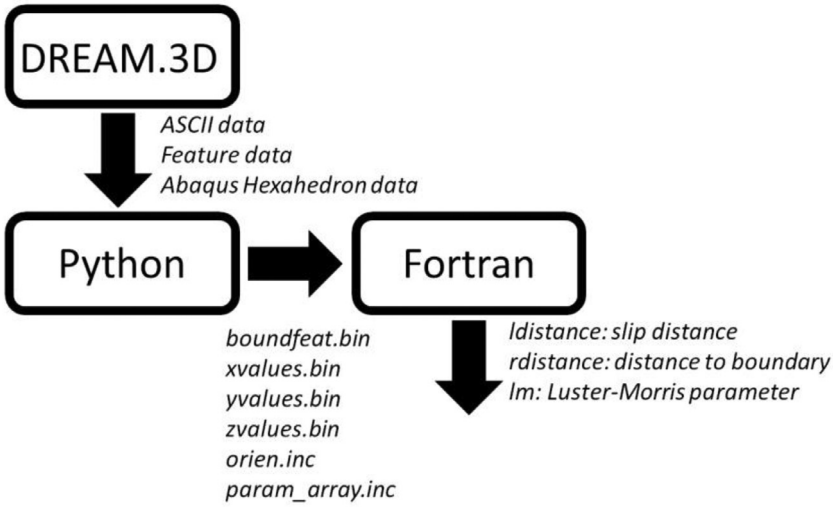

Fig. 1. Integration of software to construct the input data required for the Fortran length scale code which outputs information (ldistance, rdistance, lm) to be applied in crystal plasticity constitutive models found in [1].

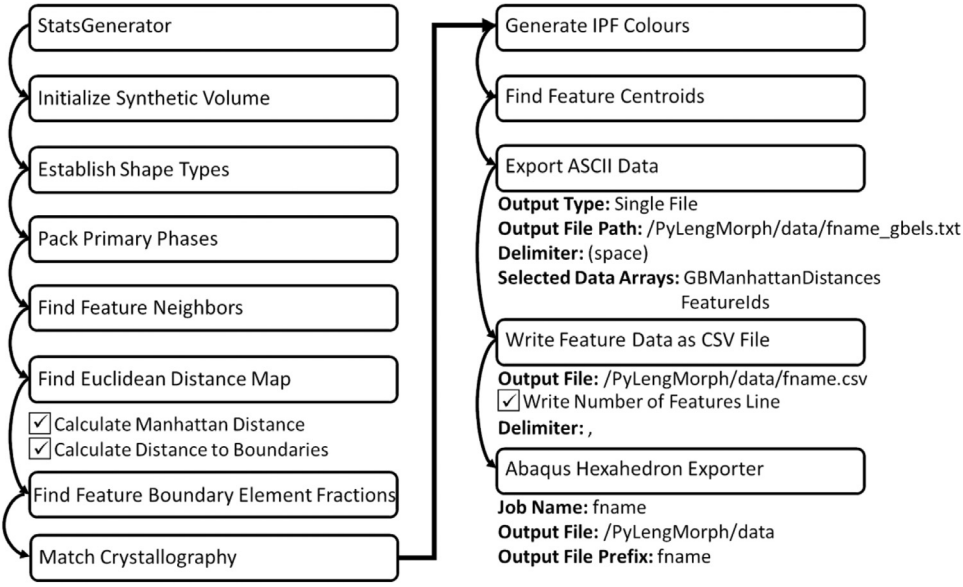

Fig. 2. Example DREAM.3D pipeline included the additional filters required to extract information for the Python code.

information includes the location in x,y, and z coordinates (in the global coordinate system), and the grain IDs of all grains which share these boundary nodes.

The Python code makes use of the NumPy [2] and pandas data analysis [3] libraries. An option is provided to the user to select two possibilities for the density of nodes on the boundary of the grain. A minimum number of nodes defining the grain boundary can be used which refers to the node locations provided in Fig. 4(a) for an example voxel/element. Alternatively, there is the possibility to increase the number of nodes to what can be observed in Fig. 4(b). The advantage of increasing the

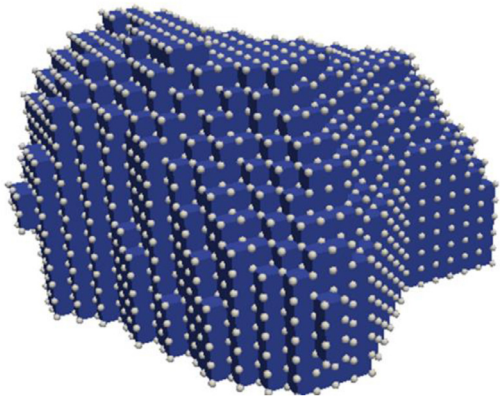

Fig. 3. Nodes on the boundary of a grain. The location in the global coordinate system and the grain IDs of each grain which shares each node used as input data to the Fortran code.

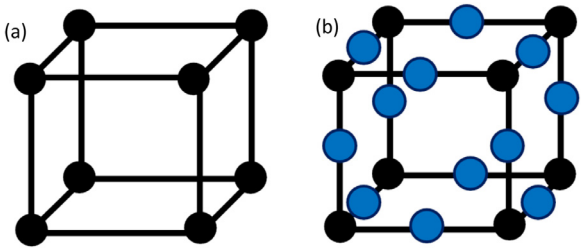

Fig. 4. Difference in the number of nodes defining a voxel/element where (a) is the minimum number of nodes, and (b) is the maximum number of nodes, where the additional nodes are given in shaded blue.

number of nodes on the grain boundary is it ensures a more accurate vector selection from which the slip distance can be calculated. This is because there is a greater number of possible vectors which can be created between the current location to the boundary, increasing the probability of selecting a vector in the closest possible orientation to the slip direction. However, doing so will increase the computational time of the analysis since there are a greater number of nodes to cycle through. Therefore, it is up to the user to choose the preferred option.

The code used to construct the required input matrices can be downloaded from [12] and installed by navigating to the folder *PyLengMorph* before using the install pip command. Once installed, the function *grainboundary* can be used by applying the following convention, (import PyLenghMorph.grainboundary(loc='/path/to/folder', file='fname', nodeinc=True,abq=True)

Four inputs are required to use this function:

1 loc – path to the *data* folder which contains all the data generated by DREAM.3D (as described in Fig. 2).
2 Fname - name of the files generated by DREAM.3D as indicated in Fig. 2 (filters: *Export ASCII, Export Feature Data CSV File, Abaqus Hexahedron Exporter*). These files should be located in */data.*
3 Nodeinc - either *True* or *False* can be used, where *False* results in each boundary voxel defined by the minimum number of nodes (as demonstrated in Fig. 4(a)), and *True* the maximum number of nodes (as demonstrated in Fig. 4(b)).
4 abq – either *True* or *False* can be used, where *True* results in the creation of Fortran INCLUDE files which can be used with a Fortran fixed-form source, while *False* results in an INCLUDE file to be used with a Fortran free-form source.

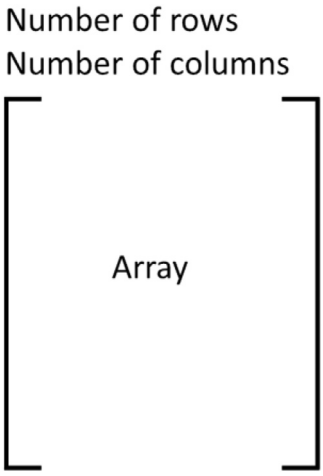

Fig. 5. Structure of the xvalues.bin, yvalues.bin, zvalues.bin, boundfeat.bin, and el_centroid.bin files, where 'Number of rows' is the integer of the rows of the extracted matrix, and 'Number of columns' is the integer of the columns of the extracted matrix.

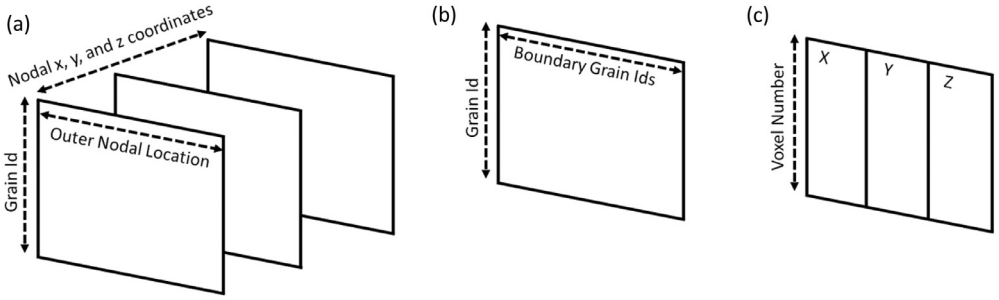

Fig. 6. Schematic of the matrices used in the analysis to implement in the grain size-morphology modification. (a) The row index represents the grain ID of interest, while the columns contain the coordinates of the surface nodes belonging to this grain. (b) A matrix where the row index represents the grain of interest, and the columns corresponding to grain IDs of the grains on the surface of the grain of interest. (c) A matrix of voxel/element centroids where the row index corresponding to the voxel/element number while the columns are the x,y,z coordinates.

After calling *grainboundary*, the input matrices are written to five different binary files with the structure of the file provided in Fig. 5, where *Number of rows/columns* refers to the dimensions of the matrix which are located on the first and second line of the file, respectively. The matrix can be read at the very start of a crystal plasticity analysis.

The five binary files created by *grainboundary* are as follows*:*

- Xvalues.bin, yvalues.bin, zvalues.bin – each file contains the x, y, and z coordinates of each boundary node in the global coordinate system.
- Boundfeat.bin – the grain ID of the grain which each shares the boundary node.
- El_centroid.bin – the centroid coordinates of all voxels/elements within the RVE.

The structure of these matrices is provided in Fig. 6. Fig. 6(a) corresponds to the matrices contained in xvalues.bin, yvalues.bin, zvalues.bin, where the row index corresponds to the grain ID of the grain the surface nodes belonged to. Fig. 6(b) corresponds to the matrix contained in boundfeat.bin. Each row corresponds to the grain ID sharing each boundary node. The row index once again refers to the grain ID which contains these boundary nodes. Fig. 6(c) corresponds to the matrix contained in el_centroid.bin where each row is the voxel/element centroid coordinates.

A Fortran INCLUDE file (orien.inc) is also created, which contains an array of all Euler angles for each grain. This information is used to create the array *oriensh* which is an input to the Fortran subroutine described in the following sections. The crystal plasticity code can be directed to this file using the convention,

INCLUDE 'orien.inc'

Finally, if *abq=True*, an additional INCLUDE file is created (*param_array.inc*). This file contains the sizes of the matrices in the binary files. This is an important INCLUDE file if Abaqus is to be used. The binary files can be read in at the very start of the analysis using the subroutine *UEXTERNALDB,* with the dimensions of these matrices provided in *param_array.inc.* This ensures the matrices can be shared between subroutines using a *COMMON BLOCK*. An example of how this is implemented is provided in the example subroutine *Example_UEXTERNALDB.for* which can be found at [12].

Slip distance and misorientation calculation (Fortran)

In this section, the calculation approach is detailed along with the corresponding Fortran code (the nomenclature of which is listed in Table 1) which implements the presented approach. Outlined in this section are the details contained in the Fortran subroutine which is used to calculate the slip distance and Luster-Morris parameter. In the following mathematical explanation, α is used to represent a slip system since the calculation approach occurs for each slip system.

Calculation approach

Starting from the current voxel/element at which the calculation is being conducted, the grain ID that the voxel/element belongs is determined. This information can be fed in as a property definition of the grain. Once this is known, the matrices containing the grain IDs and nodal coordinates are used. For the current voxel (or integration point if finite elements are being used) ($P_{vox/el}$), the Euclidean distance (d) for each node (Q_{node}) on the boundary of the grain containing the current voxel/element is calculated. The adjacent grain is determined from the index (d_{index}) of the minimum Euclidean distance,

$$d^{\alpha}\left(P_{vox/el}, Q_{node}\right)_i = \sqrt{\sum_{k=1}^{n}\left(Q_{node/k} - P_{vox/elk}\right)^2}, \text{where } D^{\alpha}\left(P_{vox/el}, Q_{node}\right) = \left\{d^{\alpha}\left(P_{vox/el}, Q_{node}\right)_i \, i \in Z^{+}\right\} \quad (1)$$

$$d^{\alpha}_{index} = \operatorname{argmin}\left(D^{\alpha}\left(P_{vox/el}, Q_{node}\right)\right) \quad (2)$$

Once the minimum Euclidean distance is determined, the nearest grain can be determined from the inputted matrices. Using the column index of the node determined to be the shortest distance, the second matrix Fig. 6(b) is used to determine the grain ID that this boundary is associated with.

Once this boundary grain is known, the Euler angles (φ_1, Φ, φ_2) defining its orientation within the global coordinate system can be extracted. This information can be extracted from the matrix defined in *Orien.inc*. Once the Euler angles are determined they are used to form a rotation matrix ($\boldsymbol{R}$),

$$R = \begin{bmatrix} \cos\varphi_1\cos\varphi_2 - \cos\Phi\sin\varphi_1\sin\varphi_2 & \sin\varphi_1\cos\varphi_2 + \cos\Phi\cos\varphi_1\sin\varphi_2 & \sin\Phi\sin\varphi_2 \\ -\cos\varphi_1\sin\varphi_2 - \cos\Phi\sin\varphi_1\cos\varphi_2 & \cos\Phi\cos\varphi_1\cos\varphi_2 - \sin\varphi_1\sin\varphi_2 & \sin\Phi\sin\varphi_2 \\ \sin\Phi\sin\varphi_1 & -\sin\Phi\cos\varphi_1 & \cos\Phi \end{bmatrix} \quad (3)$$

Using the rotation matrix, the slip direction ($\boldsymbol{S}^{\alpha}$) and slip normal ($\boldsymbol{N}^{\alpha}$) in the local coordinate system can be rotated to the global coordinate system ($\boldsymbol{s}^{\alpha}$, $\boldsymbol{n}^{\alpha}$),

$$\boldsymbol{s}^{\alpha} = \boldsymbol{R}^{T}\boldsymbol{S}^{\alpha}; \boldsymbol{n}^{\alpha} = \boldsymbol{R}^{T}\boldsymbol{N}^{\alpha} \quad (4)$$

This calculation occurs for both grain A ($\boldsymbol{s}^{\alpha}_{A}$ and $\boldsymbol{n}^{\alpha}_{A}$) and B ($\boldsymbol{s}^{\alpha}_{B}$ and $\boldsymbol{n}^{\alpha}_{B}$).

The source code for how this implemented for grain B is provide in Fig. 7 implemented in Fortran language.

Once the closest grain boundary is determined, the vectors from the current voxel/element can be formed towards each of the nodes on at the shared boundary. This is done to determine which

Table 1
Nomenclature (listed in order of appearance) used in the Fortran code used to demonstrate how the presented mathematical formulations used in the calculation process are implemented.

Input/output names	Description
feature	The grain ID which contains the current voxel/element.
arraysize	Maximum number of nodes on the surface for each grain.
nodex(:,arraysize), nodey(:,arraysize), nodez(:,arraysize)	x, y, and z-coordinates of all nodes on the surface of the current grain.
coords(3)	Coordinates of the centroid of the current voxel (FFT software) / coordinates of the node of the current element (FE software)
vect(:,:)	x, y, and z components of all vectors drawn from the voxel/element to the boundary nodes.
minindexing	Index of the location of the minimum Euclidean distance.
grainb	Grain ID of the closest adjacent grain.
slpdir(:,:), slpplane(:,:)	Slip directions and slip planes in the local (crystal) coordinate system.
oreinsh(:,:)	Array of Euler angles for each grain.
totalrot(3,3)	Rotation matrix to rotate the local slip closest grain to the global coordinate system.
slpdirrotate(:,:),slpplanrotate(:,:)	Slip direction and slip planes in the global coordinate system.
featureboundsnodes(:,:)	x, y, and z coordinates of all shared boundary nodes.
btotal(:,:)	Vector from the voxel/element to the shared boundary.
normarray(:)	Magnitudes of all vectors (btotal).
normslp(:)	Magnitudes of slip direction (slpdir).
minang0(:,:), minang180(:,:)	Minimum angles.
minangleval(:), minangleval180(:)	Index of the minimum angles.
minangleactual(:), minangleactloc(:)	Overall minimum angle and the corresponding index.
boundnodegrainb(:)	The index of the node on the shared boundary.
rnodes(:,3)	Coordinates of the node on the shared boundary.
nodesnotindex(:)	Index of nodes in grain B not at the interface of grain A and grain B.
vectr(:,:)	Vectors form the current voxel/node to the grain boundary nodes in grain B (adjacent grain).
vectrnorm(:)	Magnitude of vectr(:,:) for each slip system.
lxtotal(:,:,:)	Vectors from the grain boundary node to the grain boundary nodes in grain B (adjacent grain) for each slip system.
lxnorm(:)	Magnitude of vectors lxtotal for each slip system.
btotal(:,:)	Vector from voxel/element to the shared boundary.
normarry(:)	Magnitudes of all vectors (btotal).
normslp(:)	Magnitudes of slip direction (slpdir).
minxval0(:,:), minxval180(:,:)	Index of the minimum angles of each array.
minxval0act(:,:), minxval180act(:,:)	Minimum angles of arrays.
minxangle(:)	Overall minimum angle for each slip system.
minxangleact(:)	Overall minimum angle index for each slip system.
lxnodesindex	Index of the node at the boundary of grain B.
lxnodes(:)	Coordinates of the identified node on the boundary away from the grain A and B interface.
Subroutines	**Description**
eulercosmatrix	Constructs the rotation matrix base on the Euler angles.
enorm	Calculates the magnitude of vectors.

node at the shared boundary ensures the formation of a vector which is deemed to be in the same orientation as the slip directions in the adjacent grain. This is shown schematically in Fig. 8.

To achieve this, the coordinates of the feature boundary nodes are determined, which are then used with the voxel/element coordinates to find the corresponding vector ($\boldsymbol{a}^{\alpha}_{bound\,i}$, where $\boldsymbol{A}^{\alpha}_{bound} = \{\boldsymbol{a}^{\alpha}_{bound\,i} | i \in \mathbb{Z}^{+}\}$). These vectors are compared to the slip directions (for each slip system α) in the adjacent grain ($\boldsymbol{s}^{\alpha}_{B}$) by calculating the angle between them ($\theta^{\alpha}_{diff\,i}$),

$$\theta^{\alpha}_{diff\,i} = \cos^{-1} \frac{\boldsymbol{a}^{\alpha}_{bound\,i} \cdot \boldsymbol{s}^{\alpha}_{B}}{\left|\boldsymbol{a}^{\alpha}_{bound\,i}\right| \left|\boldsymbol{s}^{\alpha}_{B}\right|} \tag{5}$$

The angle is calculated for all vectors resulting a set of values. The boundary vector and slip vector are parallel if the angle between them is 0 and 180°, therefore, the minimum for these two cases are

```
! Create a vector from the current voxel position to the boundary nodes of the current grain.
vect(1,:)=nodex(int(feature),1:arraysize)-coords(1)
vect(2,:)=nodey(int(feature),1:arraysize)-coords(2)
vect(3,:)=nodez(int(feature),1:arraysize)-coords(3)

! Use these vectors to calculate the distance of the current location (voxel) to the grain boundary.
mindist=((vect(1,:)**2.D0) + (vect(2,:)**2.D0) + (vect(3,:)**2.D0))**0.5D0

! Finding the closest boundary grain.
minindexsing=minloc(mindist,1)

! Using the index of the minimum distance, the closest grain can be found and labelled grain B.
grainb=int(boundgrain(feature,minindexsing))

! Rotate the slip systems using the angles from each identified grain.
call eulercosmatrix(oriensh(grainb,:),totalrot)
do j=1,size(slpdir,1)
      slpdirrotate(j,:)=matmul(totalrot,slpdir(j,:))
      slpplanrotate(j,:)=matmul(totalrot,slpplane(j,:))
end do
```

Fig. 7. Fortran code used to calculate the closest grain to the current voxel/element based on the calculated Euclidean distance.

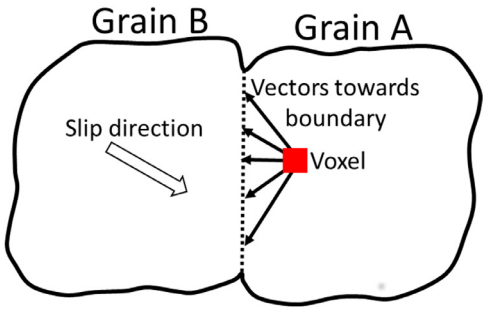

Fig. 8. A schematic demonstrating the approach to find the most suitable nodes on the shared boundary which results in the same orientation as the slip direction in the adjacent grain (grain B).

extracted,

$$\theta_0^\alpha = \left(\Theta_{diff}^\alpha\right);\ \theta_{180}^\alpha = \left(\left|\Theta_{diff}^\alpha - 180\right|\right),\ \text{where}\ \Theta_{diff}^\alpha = \left\{\theta_{diff_i}^\alpha\, i \in \mathbb{Z}^+\right\} \tag{6}$$

The index of the location of the minimum angles are also stored for later use,

$$\theta_{0,index}^\alpha = \text{argmin}\left(\Theta_{diff}^\alpha\right);\ \theta_{180,index}^\alpha = \text{argmin}\left(\left|\Theta_{diff}^\alpha - 180\right|\right) \tag{7}$$

The final minimum angle for each slip system (θ_{min}^α), and therefore the corresponding boundary node can be determined by comparing the two minimums to find the total minimum,

$$\theta_{min}^\alpha = (\theta_0, \theta_{180}) \tag{8}$$

Using the index of the location within the set at which the minimum is located, the corresponding vector can be extracted,

$$\boldsymbol{a}_{bound}^\alpha = \boldsymbol{a}_{bound\,k}^\alpha,\ \text{where}\ k = \theta_{0,index}^\alpha \vee \theta_{180,index}^\alpha \tag{9}$$

The magnitude of this vector ($\boldsymbol{a}_{bound}^\alpha$) is also calculated to extract the distance from the current voxel to the boundary. This is an additional output not utilised in the crystal plasticity constitutive models in [1] but can be used is in alternate crystal plasticity constitutive models.

```
! Find the coordinates of the identified shared boundary nodes.
featureboundsnodes=reshape((/nodex(grainb,grainboundindex),                &
                nodey(grainb,grainboundindex),                             &
                nodez(grainb,grainboundindex)/),(/val-1,3/)

! Calculate he vector from the boundary nodes to the current voxel.
bxvalue=featureboundsnodes(:,1)-coords(1)
byvalue=featureboundsnodes(:,2)-coords(2)
bzvalue=featureboundsnodes(:,3)-coords(3)
btotal=reshape((/bxvalue,byvalue,bzvalue/),(/val-1,3/))

! Calculate the magnitude of vectors.
call enorm(btotal,val-1,normarray)
call enorm(slpdirrotate,size(slpdirrotate,1),normslp)

! Find the closest angle between the vectors from the voxel to the shared boundary and the slip direction in the adjacent grain.
do i=1,size(slpdirrotate,1)
    ! Calculate the angle between the slip direction in the adjacent grain and the vector created between the current voxel and
    ! the boundary nodes.
    bangle(i,:)=dacos(matmul(btotal,slpdirrotate(i,:))/                    &
                (normarray*normslp(i)))
    ! Find minimum angle and the location index of the array for each slip direction.
    minang0(i,:)=minloc(bangle(i,:))
    minangleval(i)=bangle(i,minang0(i,1))
    ! Find the minimums at angles of 180 degrees.
    minang180(i,:)=minloc(abs(bangle(i,:)-pi))
    minangleval180(i)=abs(bangle(i,minang180(i,1))-pi)
    ! Find the smaller of the 0 and 180 degrees to determine the true minimum value.
    if (minangleval(i) .lt. minangleval180(i)) then
        minangleactual(i)=minangleval(i)
        minangleactloc(i)=minang0(i,1)
    else
        minangleactual(i)=minangleval180(i)
        minangleactloc(i)=minang180(i,1)
    end if
end do

! Using the minimum angles, the nodes at the boundary which connect to the voxel can be determined.
boundnodegrainb=grainboundindex(minangleactloc)
```

Fig. 9. Fortran code to find the nodes at the boundary of grain A and B. This is achieved by finding a vector from the current voxel/element to the boundary which is in the direction of slip in the closest grain.

The source code for how this is implemented is provide in Fig. 9.

Once the nodes at the shared boundary (P^{α}_{inter}) are determined, the next step is to determine the slip length in the adjacent grain. This requires finding the nodes on the boundary of grain B away from the shared boundary which forms a vector orientated in the same direction as slip in this grain B. This process is schematically shown in Fig. 10. In Fig. 10 the vector formed from the voxel/element toward the grain boundary in the direction of slip in grain A ($\boldsymbol{a}^{\alpha}_{bound_k}$), is then matched with a vector ($\boldsymbol{b}^{\alpha}_{bound\,i}$) formed from the identified point on the grain A-B interface with nodes on the boundary of grain B (Q^{α}_{bound}, where $Q^{\alpha}_{bound} = \{q^{\alpha}_{bound\,i} | i \in \mathbb{Z}^{+}\}$),

$$\overrightarrow{P^{\alpha}_{inter}Q^{\alpha}_{bound}} = \boldsymbol{b}^{\alpha}_{bound\,i} = Q^{\alpha}_{bound} - P^{\alpha}_{inter}, \text{ where } \boldsymbol{B}^{\alpha}_{bound} = \left\{\boldsymbol{b}^{\alpha}_{bound\,i} | i \in \mathbb{Z}^{+}\right\} \tag{10}$$

This is done until an appropriately orientated vector is found (given as the vector in bold in grain B).

This process used to determine the nodes on the outer surface of grain B to create the appropriately defined vector ($\boldsymbol{B}^{\alpha}_{bound}$) utilises the same approach as before where the angle between vectors are considered. Firstly, since the vector from the current location (voxel/element) to the boundary nodes is already known, a set of angles $\Theta^{\alpha}_{A-B,diff}$ (where $\Theta^{\alpha}_{A-B,diff} = \{\theta^{\alpha}_{A-B,diff_i} | i \in \mathbb{Z}^{+}\}$) is formed between $\boldsymbol{a}^{\alpha}_{bound_k}$ and all vectors within the set $\boldsymbol{B}^{\alpha}_{bound}$,

$$\theta^{\alpha}_{A-B,diff} = \cos^{-1}\frac{\boldsymbol{a}^{\alpha}_{bound_k} \cdot \boldsymbol{b}^{\alpha}_{bound\,i}}{\left|\boldsymbol{a}^{\alpha}_{bound_k}\right|\left|\boldsymbol{b}^{\alpha}_{bound_i}\right|} \tag{11}$$

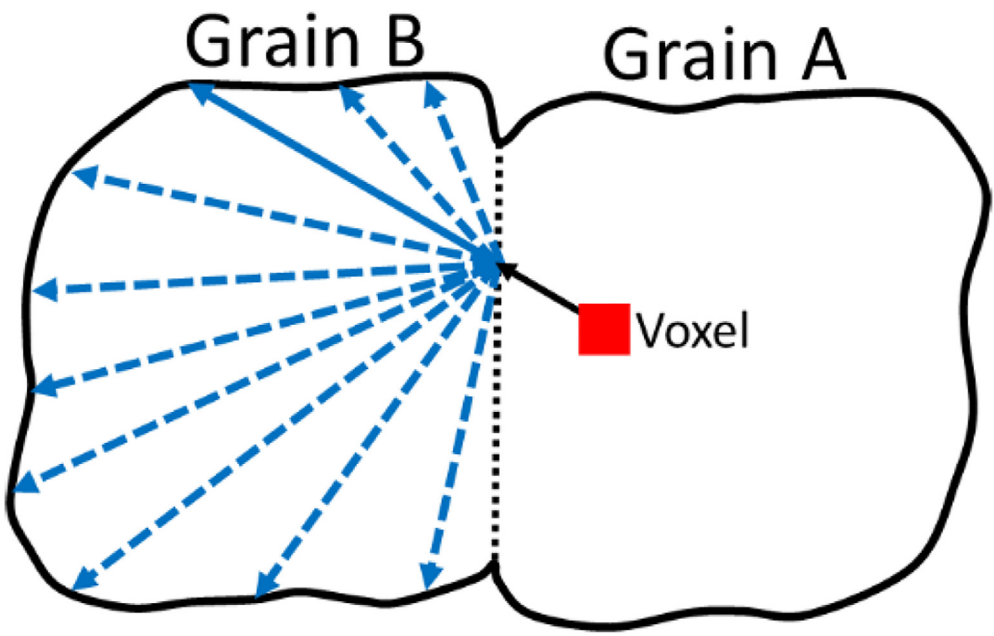

Fig. 10. A schematic showing the process from which an appropriately orientated vector is formed in grain B using the identified grain boundary node.

The vectors $a^{\alpha}_{bound_k}$ and $b^{\alpha}_{bound\,i}$ are parallel if the angle between them is 0 or 180°, therefore, the minimum for these two cases are extracted,

$$\theta^{\alpha}_{A-B(0)} = \min\left(\Theta^{\alpha}_{A-B,diff}\right);\ \ \theta^{\alpha}_{A-B(180)} = \min\left(\left|\Theta^{\alpha}_{A-B,diff} - 180\right|\right) \tag{12}$$

The index of the location of the minimum angles are also stored for later use,

$$\theta^{\alpha}_{A-B(0),index} = \text{argmin}\left(\Theta^{\alpha}_{A-B,diff}\right);\ \theta^{\alpha}_{A-B(180),index} = \text{argmin}\left(\left|\Theta^{\alpha}_{A-B,diff} - 180\right|\right) \tag{13}$$

The final minimum angle for each slip system ($\theta^{\alpha}_{A-B,min}$) and therefore the corresponding boundary node can be determined by comparing the two minimums to find the total minimum,

$$\theta^{\alpha}_{A-B,min} = \min\left(\theta_{A-B(0)}, \theta_{A-B(180)}\right) \tag{14}$$

Using the index of the location within the set at which the minimum is located, the corresponding boundary node (q^{α}_{bound}) can be extracted,

$$q^{\alpha}_{bound} = q^{\alpha}_{bound\,k},\ \text{where}\ k = \theta^{\alpha}_{A-B(0),index} \nu \theta^{\alpha}_{A-B(180),index} \tag{15}$$

The source code for how this implemented is provide in Fig. 11.

Once the location on the grain boundary away from the grain A-B interface is determined (q^{α}_{bound}), the Euclidean distance from the grain A-B interface node (P^{α}_{inter}) to q^{α}_{bound} can be calculated to find the slip distance (L^{α}),

$$L^{\alpha} = d^{\alpha}\left(P^{\alpha}_{inter}, q^{\alpha}_{bound}\right) = \sqrt{\sum_{k=1}^{n}\left(q^{\alpha}_{bound\,k} - P^{\alpha}_{inter\,k}\right)^2} \tag{16}$$

Additionally, the Luster-Morris parameter ($m^{\alpha'}$) can also be determined using the slip systems in grain A and B,

$$m^{\alpha'} = \left(\frac{\boldsymbol{n}^{\alpha}_{A} \cdot \boldsymbol{n}^{\alpha}_{B}}{\left|\boldsymbol{n}^{\alpha}_{A}\right|\left|\boldsymbol{n}^{\alpha}_{B}\right|}\right)\left(\frac{\boldsymbol{s}^{\alpha}_{A} \cdot \boldsymbol{s}^{\alpha}_{B}}{\left|\boldsymbol{s}^{\alpha}_{A}\right|\left|\boldsymbol{s}^{\alpha}_{B}\right|}\right) \tag{17}$$

```
! Grain b grain boundary node.
boundnodegrainb=grainboundindex(minangleactloc)
! Node coordinates located at the boundary of grain a and grain b.
rnodes=reshape((/nodex(grainb,int(boundnodegrainb)), nodey(grainb(int(boundnodegrainb)),                    &
                 nodez(grainb,int(boundnodegrainb))/), (/size(slpdirrotate,1),3/))

vectr=reshape((/nodex(grainb,int(boundnodegrainb))-coords(1), nodey(grainb(int(boundnodegrainb))-coords(2),  &
                 nodez(grainb,int(boundnodegrainb))-coords(3)/), (/size(slpdirrotate,1),3/))
! Calculate the magnitude of all vectors
call enorm(vectr,size(slpdirrotate,1),vectrnorm)
! Calculate vectors from the identified grain boundary nodes to the nodes on the boundary of grain b (identified adjacent grain).
do i=1,size(slpdirrotate,1)
    lxtotal(i,:,:)=reshape((/nodex(grainb,int(nodesnotindex))-rnodes(i,1),nodey(grainb,int(nodesnotindex))  &
                -rnodes(i,2),nodez(grainb,int(nodesnotindex))-rnodes(i,3)/),(/valb-1,3/))
    lxnormin=lxtotal(i,:,:)
    call enorm(lxnormin,valb-1,lxnorm)
    ! Find the angle between the vectors from the shared boundary to boundary nodes in grain b and the vector
    ! developed from the voxel to the shared boundary.
    lxangle(i,:)=dacos(matmul(lxtotal(i,:,:),vectr(i,:))/(lxnorm*vectrnorm(i)))
    minxval0(i,:)=minloc(lxangle(i,:))
    minxval0act(i,:)=lxangle(i,int(minxval0(i,:)))
    minxval180(i,:)=minloc(abs(lxangle(i,:)-pi))
    minxval180act(i,:)=abs(pi-lxangle(i,int(minxval180(i,:))))
    ! Initialise the overall minimum angle and corresponding index.
    minxangle(i)=minxval0(i,1)
    minxangleact(i)=minxval0act(i,1)
    ! Since the vectors in this case can be parallel but in opposite directions, an 'if' loop is used to sort through the 0deg and
    ! 180deg possible minimums.
    if(minxval0act(i,1)<minxval180act(i,1))then
        minxangle(i)=minxval0(i,1)
        minxangleact(i)=minxval0act(i,1)
    else
        minxangle(i)=minxval180(i,1)
        minxangleact(i)=minxval180act(i,1)
    end if
end do
! Index in matrix for the location of the node on the boundary of grain a.
lxnodesindex=nodesnotindex(minxangle)
! Coordinates of the node on the boundary.
lxnodes=reshape((/nodex(grainb,lxnodesindex),nodey(grainb,lxnodesindex), nodez(grainb,lxnodesindex)/),    &
        (/size(slpdirrotate,1),3/))
```

Fig. 11. Fortran code used to find the slip distance in grain B from the node at the grain A-B interface.

Crystal plasticity implementation

The calculation procedure described above is implemented in a Fortran subroutine which can be found in the supplementary information and [12]. This subroutine can be added to crystal plasticity code which can then use the calculated outputs from the subroutine in the underlying constitutive equations, as demonstrated in [1]. The inputs and outputs of this subroutine are listed in Table 2.

To demonstrate how the subroutine works, and therefore provide a tool to familiarise potential users, a Fortran program has been developed. This program can be found in the supplementary material and at [12] along with example input data in the form of binary files described in the previous section. The subroutines to read in the binary file matrices are also included in the program (subroutines: *arraycoords, boundfeat, el_centroid*). These subroutines can be copied and used in other crystal plasticity software to initialise the required data at the start of the analysis. If Abaqus is being used, the subroutines *arraycoords, boundfeat, el_centroid* can be added in the subroutine *UEXTERNALDB* which Abaqus calls at the beginning of the analysis, an example of which (*Example_UEXTERNALDB.for*) can be found at [12].

Once compiled, the program will prompt the user to provide a grain ID and element number. Once supplied, the distance of the current voxel/element from the boundary (*rdistance*), slip distance (*ldistance*) in the adjacent grain, and the Luster-Morris parameter (*lm*) will be supplied for each slip system (for this test program, a face centred cubic crystal structure is assumed).

In the developed Fortran program, the subroutine used to generate the slip systems (*slipsysdyn*) was adopted from the classical crystal plasticity subroutine developed by Huang [13].

Table 2
Information on the inputs, where these inputs originate, and outputs for the length-scale subroutine.

Inputs	Description	Origin
Coords	The current voxel/element coordinates which is given as x,y,z values.	This should be available from the software being used since it is the location of the current voxel/integration point at which the calculation is being performed.
Nodeout	Total number of columns for the input matrices (nodex, nodey, nodez, boundfeat), which corresponds to the nodes located on the boundary of every grain.	This is extracted when the matrices are read in at the start of the analysis.
totalfeat	Total number of grains in the RVE.	This is extracted when the matrices are read in at the start of the analysis. Please see the subroutine *arraycoords* in the example Fortran program provided in supplementary material.
nodex, nodey, nodez	x, y, and z coordinates for each node on the boundary of the grains.	Arrays formed from the inputted from the matrices read in at the start of the analysis. Please see the subroutine *arraycoords* in the example Fortran program provided in supplementary material.
slpdir1	Slip directions in the local (crystal) coordinate system	This should be available from information already required for crystal plasticity constitutive models.
slpnor1	Slip plane normals in local (crystal) coordinate system.	This should be available from information already required for crystal plasticity constitutive models.
elcent	Centroid (in x, y, z) of the voxel/element	Array formed from the inputted matrix. Please see the subroutine *el_centroid* in the example Fortran program provided in supplementary material.
feature	The current grain index in which the voxel/element is located.	This should be available from information already required for crystal plasticity constitutive models. If not, this information can be supplied via the inputted material file which is a requirement for all crystal plasticity software.
boundgrain	Total number of boundary grains surrounding each feature (extracted from the inputted multidimensional array boundfeat).	This is extracted when the matrices are read in at the start of the analysis. Please see the subroutine *boundfeat* in the example Fortran program provided in supplementary material.
oriensh	Orientations in Euler angles for each grain within the RVE.	This is information which can be contained in an INCLUDE file (see supplementary information *orien.inc* as an example).
noel	Voxel/element index.	This should be available as a stored value in the software being used since it is the index of the current voxel/integration point.
Outputs	**Description**	
rdistance	Distance calculated from the current voxel/element to the grain boundary in the direction of slip in grain A.	
ldistance	Total slip distance in grain B.	
lm	Luster-Morris parameter.	

The advantage of the program is it promotes the development of understanding the intricacies of the presented Fortran subroutine.

Conclusion

The proposed approach provides the extraction of the slip length in adjacent grains for each slip system. Additionally, the interaction of slip systems in grain A and B are extracted. The Fortran subroutine which implements this approach (found in supplementary documentation) currently uses

the interaction of slip systems to calculate the Luster-Morris parameter. However, the extracted difference in orientations can be used to calculate other geometric slip transfer criteria such as those proposed in [14–16]. Additionally, the distance from the current voxel/element from the boundary in grain A (in the direction of slip in grain B) is also calculated and outputted. This is extra data which can be used in crystal plasticity constitutive models if required.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors would like to thank EDF Energy and EPSRC [grant EP/R020108/1] for funding this work. Additionally, the authors would like to thank the computational facilities of the Advanced Computing Research centre, University of Bristol (http://www.bris.ac.uk/arc/), which was used for the simulation component of this study.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.mex.2022.101763.

References

[1] D. Agius, A. Kareer, A.A. Mamun, C. Truman, D.M. Collins, M. Mostafavi, D. Knowles, A crystal plasticity model that accounts for grain size effects and slip system interactions on the deformation of austenitic stainless steels, Int. J. Plast. 152 (2022) 103249.
[2] E.O. Hall, The deformation and ageing of mild steel: III discussion of results, Proc. Phys. Soc. London Sect. B 64 (9) (1951) 747–753.
[3] N.J. Petch, The cleavage strength of polycrystals, J. Iron Steel Inst. 174 (1953) 25–28.
[4] R.W. Armstrong, I. Codd, R.M. Douthwaite, N.J. Petch, The plastic deformation of polycrystalline aggregates, Philos. Mag. 7 (73) (1962) 45–58.
[5] A.H. Cottrell, Theory of brittle fracture in steel and similar metals, Trans. Metall. Soc. AIME 212 (1958) 192–203.
[6] E. Smith, P.J. Worthington, The effect of orientation on the grain size dependence of the yield strength of metals, Philos. Mag. 9 (98) (1964) 211–216.
[7] A. Navarro, E.R. de los Rios, An alternative model of the blocking of dislocations at grain boundaries, Philos. Mag. A 57 (1) (1988) 37–42.
[8] A.A. Nazarov, On the pile-up model of the grain size-yield stress relation for nanocrystals, Scr. Mater. 34 (5) (1996) 697–701.
[9] L.H. Friedman, D.C. Chrzan, Continuum analysis of dislocation pile-ups: influence of sources, Philos. Mag. A 77 (5) (1998) 1185–1204.
[10] M.A. Groeber, M.A. Jackson, DREAM.3D: a digital representation environment for the analysis of microstructure in 3D, Integr. Mater. Manuf. Innov. 3 (1) (2014) 56–72.
[11] J. Luster, M.A. Morris, Compatibility of deformation in two-phase Ti-Al alloys: dependence on microstructure and orientation relationships, Metall. Mater. Trans. A 26 (7) (1995) 1745–1756.
[12] D. Agius, LengMorph: a tool to add a length scale dependence to crystal plasticity simulations [Computer software] (2022). https://doi.org/10.5281/zenodo.6778289.
[13] Y. Huang, A User-Material Subroutine Incorporating Single Crystal Plasticity in the ABAQUS Finite Element Program, Division of Applied Sciences, Harvard University, Cambridge, Massachusetts, 1991.
[14] J.D. Livingston, B. Chalmers, Multiple slip in bicrystal deformation, Acta Metall. 5 (6) (1957) 322–327.
[15] Z. Shen, R.H. Wagoner, W.A.T. Clark, Dislocation pile-up and grain boundary interactions in 304 stainless steel, Scr. Metall. 20 (6) (1986) 921–926.
[16] W.A.T. Clark, R.H. Wagoner, Z.Y. Shen, T.C. Lee, I.M. Robertson, H.K. Birnbaum, On the criteria for slip transmission across interfaces in polycrystals, Scr. Metall. Mater. 26 (2) (1992) 203–206.
[17] G. Van Rossum, F.L. Drake, Python 3 Reference Manual, CreateSpace, Scotts Valley, CA, 2009.