

IMPROVING THE FUNCTION FOR GRAIN BOUNDARY ENERGY
INTERPOLATION IN URANIUM DIOXIDE

by

Jarin French

A senior thesis submitted to the faculty of

Brigham Young University - Idaho

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics

Brigham Young University - Idaho

December 2016

Copyright © 2016 Jarin French

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY - IDAHO

DEPARTMENT APPROVAL

of a senior thesis submitted by

Jarin French

This thesis has been reviewed by the research committee, senior thesis coordinator, and department chair and has been found to be satisfactory.

Date

Evan Hansen, Advisor

Date

Matt Zachreson, Committee Member

Date

David Oliphant, Committee Member

Date

Stephen McNeil, Chair

ABSTRACT

IMPROVING THE FUNCTION FOR GRAIN BOUNDARY ENERGY INTERPOLATION IN URANIUM DIOXIDE

Jarin French

Department of Physics

Bachelor of Science

Others have made efforts to find an interpolary function for the grain boundary (GB) energies of uranium dioxide, based on work done by Bulatov *et al.*[Acta Mater. 65, 161 (2014)]. This work developed a MATLAB[®] script based on Harbison[B.S. Thesis, Brigham Young University - Idaho (2015)] and Bulatov *et al.* to improve such a function. This work collected molecular dynamics data using the LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulation) program developed at Sandia National Laboratory. This work collected results for the $\langle 100 \rangle$, $\langle 110 \rangle$, and $\langle 111 \rangle$ symmetric tilt and twist GBs. Calculating the new data with an 800 K anneal allowed the atoms to relax to a lower energy state. The $\langle 100 \rangle$ and $\langle 110 \rangle$ symmetric tilt and $\langle 110 \rangle$ twist sets show an improved fit, whereas the $\langle 100 \rangle$ twist and $\langle 111 \rangle$ symmetric tilt and twist sets show unexpected trends. Further research needs to be done for the $\langle 100 \rangle$ and $\langle 111 \rangle$ sets to determine why the fitting procedure does not accurately reflect the expected results. Additional research should also be done to determine if outlying data points necessitate fitting additional cusps.

ACKNOWLEDGMENTS

I would first like to thank the Department of Energy Office of Science for the Visiting Faculty Program (VFP) - Student, and the Nuclear Energy Advanced Modeling and Simulation (NEAMS) programs which allowed me this research opportunity. Thanks also goes to Brigham Young University - Idaho and Idaho National Laboratory for providing me with the necessary facilities to do this work. I would especially like to thank my mentor, Dr. Yongfeng Zhang, for his patience and guidance as I have worked on this research. Additionally I would like to thank Dr. Evan Hansen, John-Michael Bradley, and Dr. Xianming Bai for their valuable contributions to my understanding. Thanks also goes to my committee who helped me to clarify my words and ideas to make them clear for this thesis. Finally, I would like to thank my wife Katrina who has constantly supported and strengthened me as I have spent so much time doing this work.

Contents

Table of Contents	xi
List of Figures	xiii
1 Introduction	1
2 Background	3
3 Methods	7
3.1 Molecular Dynamics	7
3.2 Bulatov <i>et al.</i> 's Methods	9
3.3 Code Analysis	10
3.3.1 The Fitting Code	11
3.3.2 The Energy Calculation Code	12
3.4 Reduced Chi-Square Statistic	13
3.4.1 Developing the P and Q Matrices	13
3.4.2 Calculating Reduced Chi Squared	18
4 Results and Discussion	21
4.1 Validation of P and Q Matrices	21
4.2 Fitting Results	21
4.3 Reduced Chi-square Results	24
5 Conclusion	29
Bibliography	31
A List of Parameters	35
B Grain Boundary Representations	39
B.1 Axis-Angle Representation	39
B.2 Rodrigues Representation	40
B.3 Fundamental Zone Representation	41
C Graphs	43
D Orientation Matrix Generator	47

E	Rotation Matrix Generator	61
F	genOrientationMatrix.sh Bash Script	69

List of Figures

1.1	Example of the fluorite crystal structure.	1
2.1	Examples and types of grain boundaries.	4
3.1	An example of crystal structure after annealing.	8
3.2	The theoretical relationship between high-symmetry subsets and fundamental zone.	10
3.3	The general form of an RSW function.	11
3.4	Geometric method of determining grain boundary normals.	16
4.1	A comparison of the $\langle 100 \rangle$ copper curve with the calculated results.	22
4.2	Results for the $\langle 100 \rangle$ fitting.	24
4.3	Results for the $\langle 110 \rangle$ fitting.	24
4.4	Results for the $\langle 111 \rangle$ fitting.	25
4.5	Comparison of the PQ matrices with the expected result for $\langle 100 \rangle$ tilt. . . .	25
4.6	Possible changes to fitting functions for the 1D twist subsets.	26
C.1	A comparison of the $\langle 100 \rangle$ copper curves with the calculated results.	43
C.2	A comparison of the $\langle 110 \rangle$ copper curves with the calculated results.	44
C.3	A comparison of the $\langle 111 \rangle$ copper curves with the calculated results.	44
C.4	Comparison of the PQ matrices with the expected result for $\langle 100 \rangle$ 1D subset. .	45
C.5	Comparison of the PQ matrices with the expected result for $\langle 110 \rangle$ 1D subset. .	45
C.6	Comparison of the PQ matrices with the expected result for $\langle 111 \rangle$ 1D subset. .	46

Chapter 1

Introduction

Today's nuclear reactors primarily use uranium dioxide (UO_2) as their fuel source.¹ Understanding the properties of UO_2 requires an analysis of the basic crystal structure of the material. The ceramic material UO_2 has a series of crystal lattices joined together in various ways (called twist, tilt, or mixed boundaries) to create it. This material has a fluorite crystal structure, where the uranium atoms form a face-centered cubic (fcc) lattice, and the oxygen atoms form a simple cubic lattice within the fcc frame (see Figure 1.1).

Understanding the various properties of the fuel allows nuclear reactors to run as safely and effectively as possible. A few of these properties include thermal conductivity (how well heat flows through the material), fission gas release (how some of the fission products move throughout the material as gases), and mechanical stability (i.e. how the material bends or cracks under pressure or heat). Taking thermal conductivity as an example, knowledge of this material property allows the most effective

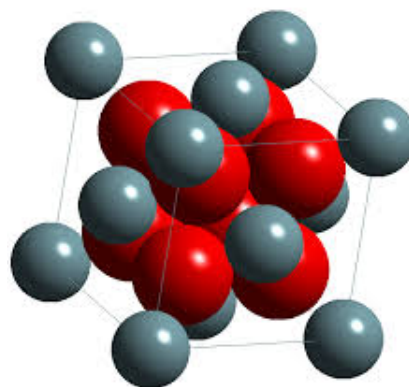


Figure 1.1 An image representing the fluorite crystal structure. For UO_2 , the smaller spheres indicate the uranium atoms, and the larger spheres indicate the oxygen atoms. Image courtesy of the University of Cambridge under the Creative Commons license.

use of coolant to keep the reactor within operating temperatures, maximizing both efficiency and safety. Knowledge of other material properties allows for similar gains in efficiency, safety, or both.

Interest in understanding UO_2 while in-reactor has led to efforts to more deeply understand the properties of the material. Currently, Idaho National Laboratory (INL) faces the challenge of not having a completely accurate model of grain boundary energy anisotropy. The current model assumes an isotropic energy, and this leads to an inability to model the material parameters correctly while in-reactor. This in turn makes efforts in nuclear energy less efficient and/or safe than it otherwise could be. INL aims to accurately model nuclear fuel while in-reactor, allowing accurate predictions regarding how the material properties will change.

This work adds to the safety and efficiency of using nuclear energy by providing the necessary information to accurately calculate the material properties of UO_2 in-reactor. Specifically, this work improves the fitting parameters for grain boundary (GB) energy interpolation for UO_2 by using molecular dynamics (MD) results calculated by Zhang² and Hansen³ with an anneal of 800 K. Furthermore, this work begins preliminary efforts towards using more accurate functions to describe GB energy behavior. Previous data did not use annealed crystal structures,⁴ which prevented the atoms from finding their ideal energetic minimum. The 800 K anneal allows the atoms to relax to a value closer to their global minimum, as shown in Chapter 4. A database will store these simulated energies, and a MATLAB® script will use the database to fit the function parameters. INL will incorporate the updated parameters into its mesoscale phase field modeling platform MARMOT for use in modeling nuclear fuels. As the modeling software incorporates these parameters, various tests of the UO_2 fuel can determine how the material properties change while in-reactor.

Chapter 2

Background

Tiny crystals called *grains* make up every polycrystalline materials (such as ceramics). The orientation of each grain does not generally depend on the orientation of the surrounding grains. Therefore, depending on how the crystal formed,⁵ the crystal structures will possibly not line up at the interfaces where two grains meet. This “atomic mismatch”⁵ leads to broken or stretched atomic bonds where atoms will not line up relative to a perfect crystal structure, creating defects called grain boundaries (GBs, see Figure 2.1a). The most popular way to parameterize a GB uses the five degree-of-freedom (DoF) model.^{4,6–10} This model only uses the macroscopic DoFs (the observable DoFs corresponding to the misorientation and inclination), ignoring the three translational DoFs (the ability of the grain to move or slide anywhere in space) possessed by each grain. Three of the five DoFs specify the misorientation (or misalignment) of the grains with respect to each other. The other two DoFs specify the orientation of the grain boundary plane (called the inclination). The rotation axis and angle define the misorientation DoFs, and the GB normal defines the inclination DoFs.⁷

Three specific types of GBs occur in polycrystalline materials: twist, tilt, and mixed GBs.^{7,10} These GBs describe the misorientation of two grains with respect to each other. Twist boundaries have the axis of rotation between the two grains and the GB normal parallel to each other. Tilt boundaries can be either symmetric or asymmetric. A perpendicular

relationship between the axis of rotation between two grains and the GB normal creates a tilt boundary. Symmetric tilt GBs have a boundary plane as a mirror plane: the atoms on one side of the boundary plane mirror the other side. This makes the angles between the boundary plane and the orientation axes of the two grains equal. Asymmetric tilt boundaries have unequal angles. Figure 2.1b shows a representation of tilt boundaries (top) and twist boundaries (bottom). A mixed GB combines twist and tilt boundary characteristics.

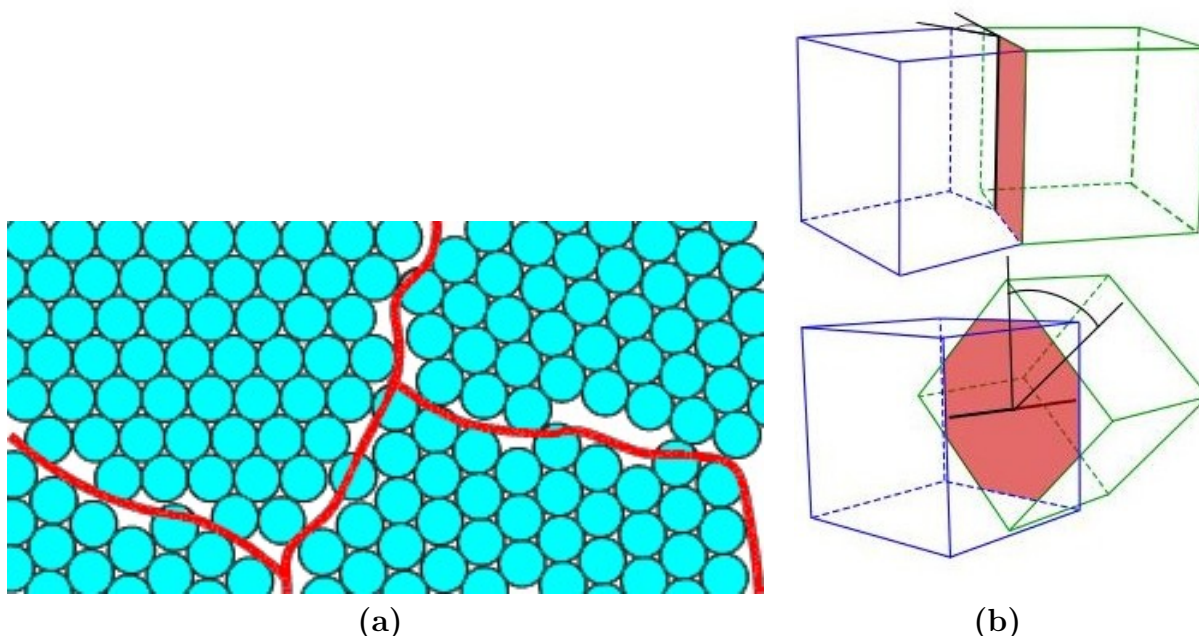


Figure 2.1 A representation of GBs, where (a) shows an example of a grain boundary and (b) shows an example of GB types. In (a) circles represent individual atoms of the grains, and the line represents the grain boundary. Atomic mismatch between the differently oriented grains causes an excess of energy within the material, which has an effect on the material's properties. Image courtesy of the University of Cambridge under the Creative Commons license. In (b) the tilt GB (top) has a perpendicular relationship between the axis of rotation and the GB normal, while the twist GB (bottom) has a parallel relationship between the two. Image courtesy of Wikipedia under the Creative Commons license.

GBs have various effects on material properties, making them important to understand.^{6,8,9} The crystal structure has extra energy because of the atomic mismatch at the boundaries. This extra energy, called GB energy, gives rise to GB motion. Knowing and predicting how the GBs will move allows for more accurate calculations of a material's properties. Thus, GB energy needs to be understood to accurately model the evolution of material

properties.

Two methods, isotropic and anisotropic, model GB energy (among others). The isotropic model most commonly (and most easily) describes GB energy. This model ignores the impact of inclination on the GB energy, and assumes equal inclinations for a given misorientation, reducing the five-dimensional (5D) parameter space to a three-dimensional (3D) parameter space. Historically, isotropic models assumed that the inclination had little or no impact on the GB energy. Later researchers used this model because of the difficulty in creating a full five DoF model.⁸ Alternatively, the anisotropic approach seeks to quantify the effect that misorientation *and* inclination have on the GB energy. Currently, researchers acknowledge the need for a full five DoF model of GB energy, but assert the difficulty inherent in developing such a model.^{7,8,10} Despite these difficulties, GB energy functions for certain materials, namely fcc metals copper, gold, aluminum, and nickel, have proven successful.⁹ Recently,⁴ similar efforts created a GB energy function for ceramics. This work improves the accuracy of the energy interpolation function for UO_2 .

Chapter 3

Methods

Any fitting procedure requires a sufficient amount of data, but unfortunately such data does not exist for uranium dioxide (UO_2) in the literature. As a work-around, this work used molecular dynamics (MD) simulations^{2,3} as fitting data to calculate the GB energies of various lattices based on the coincident site lattice (CSL) model. This model builds off of the idea that the GB energy has lower values when more lattice sites coincide. A number defined as the Σ -number describes the number of coincident sites per total number of lattice sites in a given unit cell of a crystal.^{7,10} This work developed a MATLAB[®] script using Bulatov *et al.*'s methods⁹ and building off of Harbison's script⁴ to fit parameters to the gathered data. A reduced chi-square statistic determined the effectiveness of the fit.

3.1 Molecular Dynamics

Zhang² and Hansen³ collected simulation results from the Large-scale Atomic/Molecular Massively Parallel Simulation (LAMMPS) software (developed at Sandia National Laboratory¹¹) for a number of twist, tilt, and mixed GBs. They performed these calculations by simulating two crystals of UO_2 and placing them together in various orientations. A GB forms at the interface, creating GB energy. Calculating the energy of the system from the interatomic forces inside the crystal, and comparing that energy to the energy of a single grain

(of the same size as the combined two grains) determines the energy at the GB.⁴ Calculating the GB energy follows the form:¹²

$$E_{\text{GB}} = \frac{|E_{\text{single grain}} - E_{\text{two grains}}|}{2A_{\text{GB}}}. \quad (3.1)$$

Here, E_{GB} represents the energy at the grain boundary, and $E_{\text{single grain}}$ and $E_{\text{two grains}}$ represent the energies of the single and double grains respectively. A_{GB} represents the area of the grain boundary. Figure 3.1 shows an example of how the atoms align. Harbison's original calculations⁴ used no anneal ($T_{\text{max}} \approx 0$ K), only allowing the atoms to relax to their local minima. This work used an anneal of 800 K, allowing the atoms to relax to a better estimate of their global minimum value as shown in Chapter 4. This work used the same misorientation angles for the GB energy calculations that Harbison used. The fitting procedure uses these energies to produce parameters describing the five-dimensional GB space.

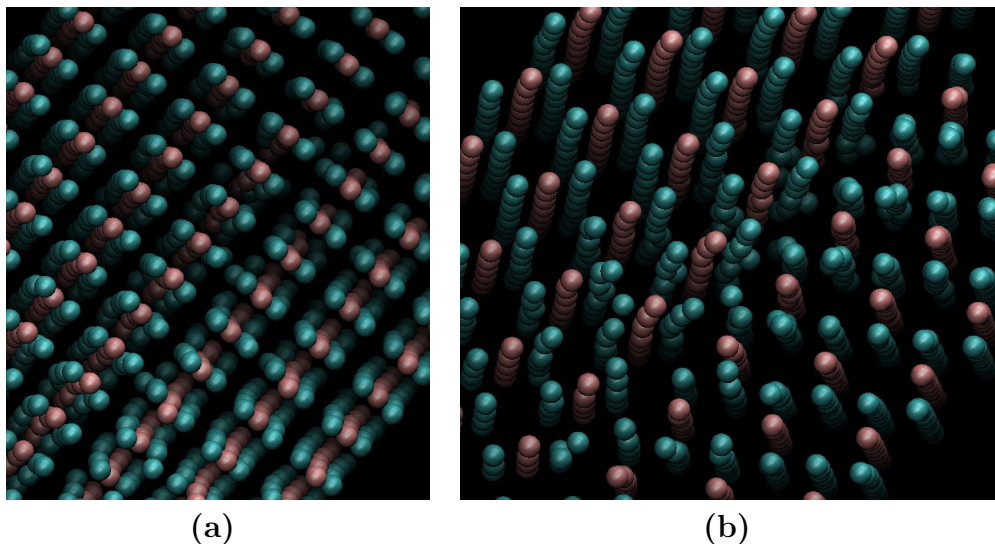


Figure 3.1 These figures demonstrate example crystal structures of UO_2 after an annealing process. The better the atoms line up, the lower the energy. (a) shows an example of a mostly aligned GB, indicative of a lower energy. (b) shows an example of a misaligned GB, indicative of a higher energy. These two images show results from a $\langle 111 \rangle$ twist simulation. Different viewpoints show different amounts of alignment. The LAMMPS simulation package takes care of all the calculations to determine the energy at these GBs. Images courtesy of Dr. Evan Hansen, used with permission.

3.2 Bulatov *et al.*'s Methods

This work implemented Bulatov *et al.*'s hierarchical interpolation method to find the energy of an arbitrary GB in the five-space.⁹ They chose three three-dimensional (3D) axes with at least two-fold symmetry (called high-symmetry axes) to use as scaffolding to build the entire five-dimensional (5D) function. Bulatov *et al.* and this work chose the $\langle 100 \rangle$, $\langle 110 \rangle$, and the $\langle 111 \rangle$ sets for their four-, two-, and three-fold rotational symmetries respectively.* Each 3D subset builds from interpolation of its own one- and two-dimensional subsets. The symmetric tilt and twist GBs for each set were fitted first because of their simplicity. The rotation angle fully defines the energies for these subsets, making them one-dimensional (in Figure 3.2a, the darker bands in the smaller circles). Having the parameters from the symmetric tilt subset allows interpolation to the asymmetric, or general, tilt subset. A second rotation angle defining the rotation of the second grain makes this subset two-dimensional (the lighter, wider band around the symmetric subset). A combination of the general tilt (two dimensions) and the twist subsets (one dimension) interpolates the 3D subset for each high-symmetry axis (the three smaller circles). Bulatov *et al.* and this work used these three 3D subsets to interpolate the GB 5D space. Figure 3.2b shows the simplified GB space using the Rodrigues fundamental zone representation. Appendix B provides a further explanation of Rodrigues space and the fundamental zone.

Bulatov *et al.* and this work used the Read-Shockley-Wolf (RSW) functions,¹⁴ which take the form:

$$E_{min} + (E_{max} - E_{min}) \sin \left(\frac{\pi}{2} \frac{\theta - \theta_{min}}{\theta_{max} - \theta_{min}} \right) \left(1 - a \log \left(\sin \left(\frac{\pi}{2} \frac{\theta - \theta_{min}}{\theta_{max} - \theta_{min}} \right) \right) \right), \quad (3.2)$$

where θ is the misorientation angle, θ_{min} and θ_{max} represent the minimum and maximum angles on the domain respectively, a is a shaping parameter, and E_{min} and E_{max} represent

*Symmetry operations for cubic crystals include rotating by 90° , 180° , or 120° about any $\langle 100 \rangle$, $\langle 110 \rangle$, or $\langle 111 \rangle$ axis respectively.¹³ Thus, the $\langle 100 \rangle$ set has four-fold symmetry ($360^\circ/90^\circ = 4$), the $\langle 110 \rangle$ set has two-fold symmetry ($360^\circ/180^\circ = 2$), and the $\langle 111 \rangle$ set has three-fold symmetry ($360^\circ/120^\circ = 3$).

the energy at θ_{min} and θ_{max} respectively. Each RSW function covers a “low-angle” subset (around 15° , with larger domains being less accurate)^{10, 14} of the domain in the 1D GB space. Figure 3.3 shows an example of a simple RSW function. Stitching together multiple RSW functions forms the 1D subsets.

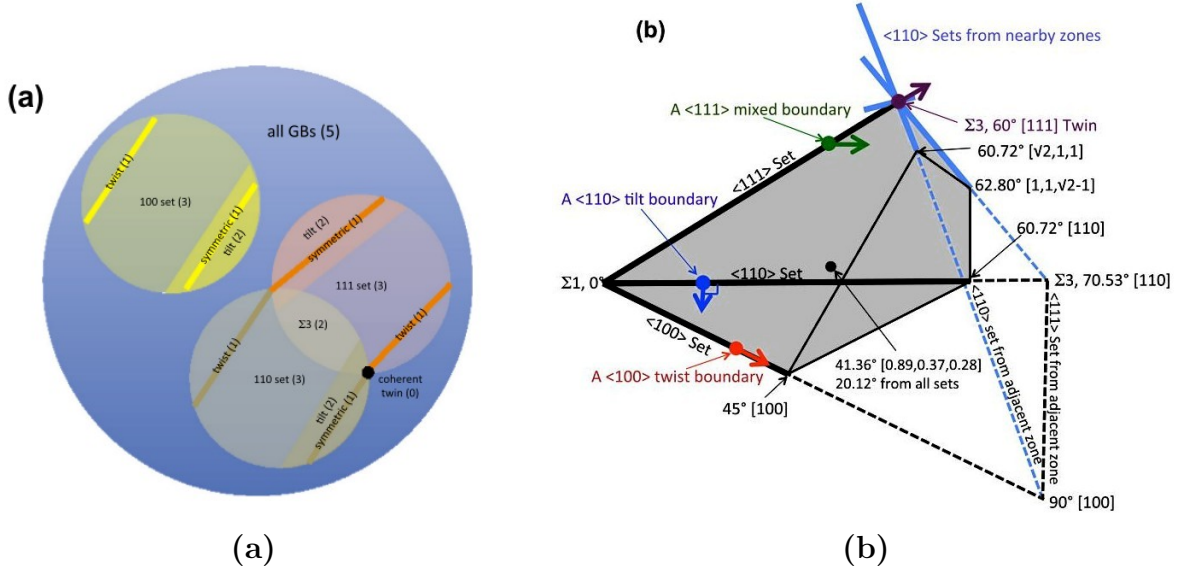


Figure 3.2 Figure 2 from Bulatov *et al.*⁹ (a) demonstrates the theoretical relationship between the high-symmetry subsets of the 5D GB space. Each multi-dimensional subset interpolates from smaller-dimensional subsets. (b) shows the Rodrigues space representation of the fundamental zone of all GBs as built from three high-symmetry axes ($\langle 100 \rangle$, $\langle 110 \rangle$, and $\langle 111 \rangle$). The unit vectors along the axis identify the boundary plane inclination in the frame of grain one. A parallel vector thus represents a twist boundary, a perpendicular vector represents a tilt boundary, and neither parallel nor perpendicular vectors represent a mixed boundary.

3.3 Code Analysis

Harbison⁴ and Bulatov *et al.*⁹ developed MATLAB[®] scripts for their work. This work analyzed these codes and used the ideas from them to develop the code that generated the parameters listed in Appendix A.

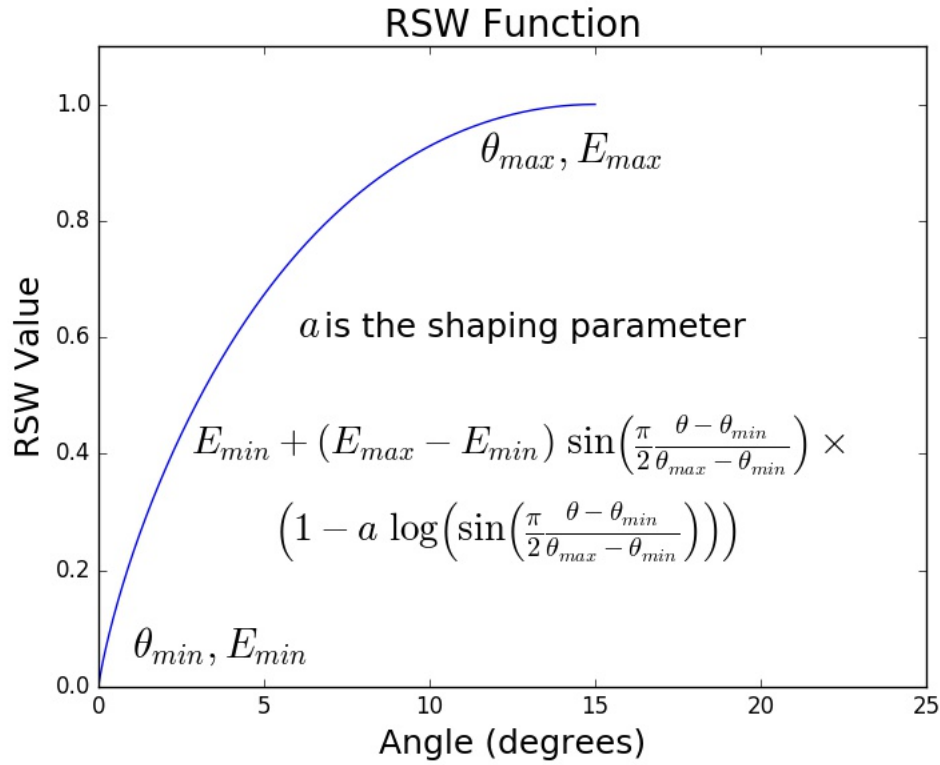


Figure 3.3 An example of an RSW function with $\theta_{min} = 0^\circ$, $\theta_{max} = 15^\circ$, and a (the shaping parameter) = 0.5. Combining these functions into a Piecewise set over a given domain gives the GB energy curves their distinct, cusp-like behavior. The RSW functions scale based on E_{min} and E_{max} . In this example, $E_{min} = 0$ and $E_{max} = 1$. Note that E_{min} and E_{max} do not represent the lowest and highest energies in the domain, but rather represent the energy at θ_{min} and θ_{max} respectively, meaning that the value of E_{min} could be higher than the value of E_{max} .

3.3.1 The Fitting Code

This work performed an extensive analysis of Harbison’s code to learn how it works and to implement the ideas therein. The basic outline for the fitting procedure follows. First, a database containing energies associated with either a twist or tilt GB on one of the three high-symmetry axes provides the fitting data. A separate database provides the test parameters which define starting points for the fitted parameters. The parameters found from the 1D fits assist in fitting the higher-dimensional sets. Important angles specify where to expect low energies, such as the $\Sigma 5$ boundary for the $\langle 100 \rangle$ symmetric tilt subset. The *Sigma*-number from CSL theory determines the angles. Because the Σ -number designates the number of

lattice points between each coincident site (and assuming the separation distance between each lattice site, or the lattice constant is known) the angle of the GB misorientation can be determined. A value known as the e_{RGB} parameter scales every energy in the parameter vector to minimize the potential for error in calculations. The e_{RGB} parameter represents the energy of an arbitrary, random GB, and represents an average of the material’s GB energies. Thus, making relevant comparisons requires unscaling the energies based on the units of energy desired (typically J/m²). All of the parameters and the angle-energy pairs from the database get passed into a grid-search fitting function. This work gave each subset a different initial step size to avoid a numerical error where the steps would take the angles currently being looked at outside of their domain. Without this, the grid-search procedure would not return the correct amount of values, preventing the code from running to completion.

After fitting the six one-dimensional subsets and the three two-dimensional subsets, interpolating the twist (1D) and asymmetric tilt (2D) subsets calculates the mixing parameters to fit the three-dimensional subsets. The mixing parameters define the relationship between the twist and general tilt subsets within a high-symmetry axis - i.e. the relationship between the small dark bands representing the twist boundaries and the lighter, wider bands representing the tilt boundaries in Figure 3.2a. The final step calculates the weighting parameters, which defines the relationship between the three high-symmetry subsets. Equations defining the various relationships can be found in Bulatov *et al.*’s work.⁹

3.3.2 The Energy Calculation Code

Bulatov *et al.*’s open-source MATLAB[®] code,⁹ `GB5DOF.m`, calculates the energy of an arbitrary GB in certain fcc metals. This work uses this script for calculating an arbitrary GB in UO₂, and proceeds as follows. First, it compares all symmetrically equivalent representations of a GB (on a per-axis basis) to calculate metrics defining the “distance” between the GB and all three high-symmetry axes (for cubic crystals, there are 24 equivalent representations¹³). Because of the three, six, and four unique axes for the $\langle 100 \rangle$, $\langle 110 \rangle$, and $\langle 111 \rangle$

axes respectively, the script calculates a maximum of $6 \times 24 = 144$ distances, afterwards discarding any that exceed a predefined cutoff distance. After calculating all distances, the script keeps only the unique representations to avoid double-counting.⁹ It then calculates energies for each unique distance in each subset, then weights and sums them to give the interpolated energy for the specified GB.

3.4 Reduced Chi-Square Statistic

A good way to test how well a function fits the data uses a reduced chi-square goodness-of-fit statistic.¹⁵ Bulatov *et al.*'s function required the orientation matrices (which Bulatov *et al.* calls the P and Q matrices for the first and second grains respectively) input parameters to calculate this statistic. These three by three matrices specify the orientation in a lab frame of the two grains individually. A good fit will have a reduced chi-square value close to one, while those values greater than one indicate an under fit, and those values less than one indicate an over fit.¹⁵

3.4.1 Developing the P and Q Matrices

This work created the P and Q matrices. Because of the vast quantity of work done with crystallography over the past few decades, many different methods can specify the orientation matrices of grains. A rotation matrix also needed to be calculated which rotates the axis of rotation to the [100] direction, as Bulatov *et al.*'s energy calculation code assumes. This work used three methods, following the method prescribed in MARMOT, using the Rodrigues rotation formula, and using the Bunge rotation matrix, in the process of developing these matrices.

MARMOT Method

MARMOT, Idaho National Laboratory's (INL's) mesoscale phase-field modeling platform,¹⁶ calculates the P and Q matrices for the grains using Euler angles as input parameters. MARMOT uses the Bunge convention to convert the Euler angles to orientation matrices. The Bunge convention uses the ZXZ or $ZX'Z''$ rotation set, which rotates first about the z axis, then the rotated x axis, and finally about the rotated z axis. Multiplying the z , x and z rotation matrices together in that order generates the formula to convert from Bunge Euler angles to the rotation matrix:

$$\begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_2 c_3 s_1 & s_1 s_2 \\ c_3 s_1 + c_1 c_2 s_3 & c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 \\ s_2 s_3 & c_3 s_2 & c_2 \end{bmatrix} \quad (3.3)$$

where c_n and s_n represent the cosine and sine of the respective angles (1 represents the first z rotation, 2 represents the x rotation, and 3 represents the second z rotation, usually referred to as¹⁷ $\varphi_1, \Phi, \varphi_2$).

MARMOT calculates the rotation matrix using the GB normal by finding the rotation matrix required to rotate that vector to the $[100]$ direction. In MARMOT, input files set up the simulations. In the input files different sections (called blocks) specify material parameters, boundary conditions, initial conditions, and the physical models to use to solve the problem (among others). A horizontal or vertical boundary for tilt or twist GBs respectively defined the initial condition used to calculate the rotation matrices in MARMOT for this set of problems. Because of this set up, tilt boundaries had GB normals along the $[010]$ axis, and twist boundaries had GB normals along the $[\bar{1}00]$ axis.

Rodrigues Rotation Formula

The Rodrigues rotation formula¹⁸ (RRF) calculates the rotation matrices given an axis and an angle using the following formula:

$$\mathbf{R} = \mathbf{I} + \sin \theta \mathbf{K} + (1 - \cos \theta) \mathbf{K}^2, \quad (3.4)$$

where \mathbf{I} is the 3x3 identity matrix, θ is the angle rotated through, and \mathbf{K} is the skew-symmetric matrix formed by the axis of rotation (\mathbf{a}) by:

$$\begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}. \quad (3.5)$$

This work calculated the rotation matrices two different ways with this orientation matrix formulation. The first method used the MARMOT-generated rotation matrices. A second method calculated the rotation matrices using geometric arguments (see Figure 3.4). From the geometric arguments this work identified the normals given in Table 3.1. Appendix E shows the code used to generate the rotation matrices.

Bunge Rotation Matrix

MARMOT uses the Bunge rotation matrix (see Equation (3.3)) to create the orientation matrices. This work used various methods to calculate the Euler angles, of which three are briefly described here. The first two methods use the Euler angles to calculate the entirety of the rotation matrix.

First, this work tried to use scripts developed to calculate the various Euler angles for MARMOT. These scripts did not work because of the same assumptions made earlier about the orientation of the GB, namely, that all pure tilt GBs have a normal of $[010]$, and that all pure twist GBs have a normal of $[\bar{1}00]$. This work assumed boundary conditions to be either perpendicular or parallel to the rotation axis, while MARMOT's boundary conditions assume GB normals along the x - or y -axes.

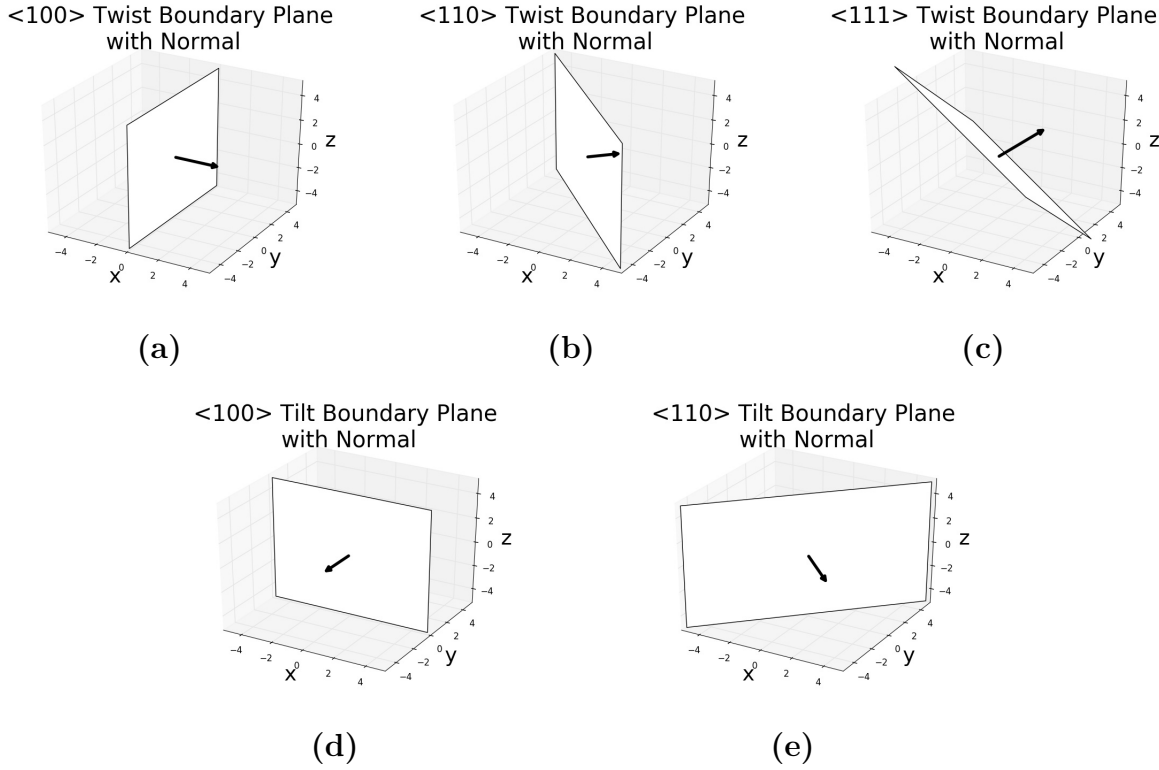


Figure 3.4 A geometric method of determining the normals of a GB. (a) to (c) show the normals for a GB perpendicular to the axis of rotation (a twist GB). The axis about which the grains rotate defines the GB normal. (d) and (e) show the normals for a GB parallel to the axis of rotation (a tilt GB). The same GB normal for $\langle 110 \rangle$ tilt boundaries can be used for $\langle 111 \rangle$ tilt boundaries.

The second method used an open-source MATLAB[®] package called MTEX.¹⁹ This package calculates Euler angles using quaternions. These Euler angles did not generate the correct results either, for the most part creating the same sorts of graphs as the MARMOT method.

The working method used the mathematics of quaternions directly.²⁰ This work calculated the quaternions based on the misorientation axis and angle. A quaternion is a four-dimensional vector containing one real part, and three imaginary parts, calculated as follows:

$$\mathbf{q} = \left[\cos\left(\frac{\theta}{2}\right), a_x \sin\left(\frac{\theta}{2}\right), a_y \sin\left(\frac{\theta}{2}\right), a_z \sin\left(\frac{\theta}{2}\right) \right], \quad (3.6)$$

with axis \mathbf{a} and misorientation angle θ . After converting the axis and misorientation angle to a quaternion, another conversion changes the quaternion to a set of Bunge Euler angles. Calculation of the angles uses Python's `atan2()` method, allowing all four quadrants in

Table 3.1 Table of GB normals for different GB types. The normalized dot product of the axis with the GB normal is zero in all tilt cases and one in all twist cases. Each subset has two options for the grain boundary normals because of inversion symmetries.

Axis	Boundary Type	GB Normal
$\langle 100 \rangle$	Tilt	$[010]$
		$[0\bar{1}0]$
$\langle 110 \rangle$	Tilt	$[1\bar{1}0]$
		$[\bar{1}10]$
$\langle 111 \rangle$	Tilt	$[1\bar{1}0]$
		$[\bar{1}10]$
$\langle 100 \rangle$	Twist	$[100]$
		$[\bar{1}00]$
$\langle 110 \rangle$	Twist	$[110]$
		$[\bar{1}\bar{1}0]$
$\langle 111 \rangle$	Twist	$[111]$
		$[\bar{1}\bar{1}\bar{1}]$

Cartesian space to be accounted for.

$$\begin{aligned}
 \chi &= \sqrt{(q_0^2 + q_3^2)(q_1^2 + q_2^2)} \\
 \varphi_1 &= \text{atan2} \left(\frac{q_0 q_2 + q_1 q_3}{2\chi}, \frac{q_0 q_1 - q_2 q_3}{2\chi} \right) \\
 \Phi &= \text{atan2} (2\chi, q_0^2 + q_3^2 - q_1^2 - q_2^2) \\
 \varphi_2 &= \text{atan2} \left(\frac{q_1 q_3 - q_0 q_2}{2\chi}, \frac{q_0 q_1 + q_2 q_3}{2\chi} \right).
 \end{aligned} \tag{3.7}$$

Inputting the Euler angles into Equation (3.3) created the orientation matrices for the grains.

Appendices D and F provide the codes used to generate the orientation matrices.

Testing The Matrices

This work attempted to reproduce the 1D subset graphs as shown in Bulatov *et al.* as a way to test the different methods. Different methods experienced various levels of success. Figures C.1 to C.3 show the matrices giving the best results.

While the first method works well for MARMOT, the MATLAB[®] script does not necessarily expect the same GB normal assumed by MARMOT. Thus, the results coming from using this combination of matrices ended up working only for the $\langle 100 \rangle$ tilt, $\langle 110 \rangle$ tilt, and $\langle 100 \rangle$ twist subsets. The $\langle 110 \rangle$ twist subset had issues with singularities, and the $\langle 111 \rangle$ subsets did not remotely match the expected outcome.

3.4.2 Calculating Reduced Chi Squared

This work used two methods to calculate the χ^2_{red} statistic. The first method used the P and Q matrices as developed above to test the entirety of the fit. The second method calculated the statistic for each 1D subset, then calculated the full χ^2_{red} value using the statistics from the subsets. Chapter 4 discusses the results from these calculations.

The test for the entire fit used the P and Q matrices to calculate the energy in 1° intervals for each subset, using Bulatov *et al.*'s `GB5DOF.m` script. This work used Equation (3.8) to calculate the χ^2_{red} value for each subset and for the entire fit, producing the results in Table 4.1 under the 800 K anneal column under the “ χ^2_{red} using P and Q matrices” section,

$$\chi^2_{\text{red}} = \frac{1}{N - n - 1} \sum \frac{(\epsilon_{\text{md}} - \epsilon)^2}{e \epsilon_{\text{md}}}. \quad (3.8)$$

In this equation, N is the number of observations, n is the number of parameters, ϵ_{md} are the energies from MD, ϵ are the energies from the model, and e is the uncertainty in the MD results.

Using the second method, the same angles used in the fitting procedure were used in the RSW equations that create the 1D subsets. The differences between the values resulting

from there and the MD simulation values lead to the χ_{red}^2 values shown in the 800 K anneal column under the “ χ_{red}^2 comparing the 1D fits” section. This work implemented the same methods to calculate the χ_{red}^2 values for the data without an anneal.

The statistic calculated using these methods differs from the χ^2 statistic used in the grid-search function. The grid-search function used Equation (3.9), and generated values of the same order as Equation (3.8).

$$\chi^2 = \sum (E_{\text{measured}} - E_{\text{calculated}})^2 \quad (3.9)$$

Chapter 4

Results and Discussion

4.1 Validation of P and Q Matrices

Comparison of the energy profiles calculated from the P and Q matrices with the copper energy profiles expected from the parameters defined in Bulatov *et al.*'s code provides a way to validate the generated matrices. Figure 4.1 shows the results from this comparison for the $\langle 100 \rangle$ set, with all six subsets shown in Figures C.1 to C.3. The calculated energies match exactly the predicted values for all but a few points. Each data set does not match the expected energy at 1° , and the tilt data sets also see this mismatch at their second to last data point.

4.2 Fitting Results

Figures 4.2 to 4.4 compare the one-dimensional (1D) results from Harbison⁴ and this work. The results show a general decrease in the grain boundary (GB) energies, allowing trends in the different subsets to emerge. These trends allow for an all around better fit, but also introduce some unexpected results. Appendix A shows the parameters calculated from the fitting procedure.

Initial MD recalculations of the $\langle 100 \rangle$ symmetric tilt GB energies using the 800 K anneal

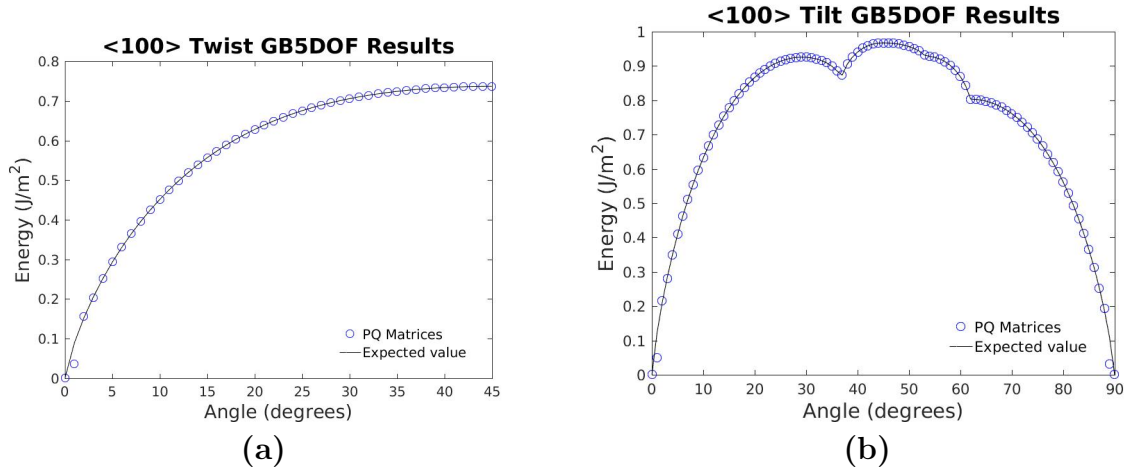


Figure 4.1 The $\langle 100 \rangle$ twist (a) and tilt (b) results for the P and Q matrices as compared to Bulatov *et al.*'s energy profiles. Bulatov *et al.*'s GB5DOF.m MATLAB[®] script calculated the expected values by using the default parameters. The GB5DOF.m script calculated the values using the generated matrices. With the exception of the data points at 1° in both (a) and (b) and 89° in (b), the energies calculated from the matrices matches the expected curves exactly.

(Figure 4.2b) showed an unexpected deep cusp around 28° . An analysis of the molecular dynamics (MD) simulation results for this misorientation revealed that, in this case, abnormally high pressures had caused the two crystals to realign. This realignment caused the misorientation angle to change causing the GB energy to be much lower than expected. Comparison with Harbison's simulation result revealed that the crystal structure from his simulation did not realign. While Harbison's did not use annealed data and thus may not represent a global minimum, the data point follows the surrounding data's trend, justifying the use of his result.

Of the symmetric tilt GB energy sets, the $\langle 110 \rangle$ set has the most improvement. All three sets showed a general decrease in the energy, increasing confidence in the accuracy of the fit for GB energies in uranium dioxide (UO_2). However, each of these sets provides more opportunity for research. The $\langle 100 \rangle$ set needs more work done for data points after around 50° . The scatter associated with those points seems to be higher, and the possibility of a slight cusp presents itself around 68° . The $\langle 110 \rangle$ set as mentioned shows the most

improvement, but some low points in the second and third “humps” do not follow the trend, indicating further possibility for cusps. The first part of the function (the first hump) needs additional data to determine the possibility of a cusp between 40° and 50° . The fitted curve to the $\langle 111 \rangle$ set now has an unexpected upward trend. The relatively high scatter associated with these data points leads to the possibility of a completely different set of functions to define this subset, meaning additional RSW functions would be required for example.

The twist GB energy sets vary in their success. The $\langle 100 \rangle$ set shows little difference between Harbison’s work and this work. An unexpected slight positive concavity at the end of the fitting for this subset indicates the possibility of a cusp. This cusp may occur around 30° . The $\langle 110 \rangle$ set has a definite decrease in the overall energies, creating a plateau profile. An additional cusp around 40° might improve the fit. The $\langle 111 \rangle$ set has the least improvement. Based on Bulatov *et al.*’s work,⁹ this work expected to see a plateau as Harbison’s fitting demonstrated.⁴ Instead, the fitting produced a curved energy profile, indicating the potential for at least one cusp, possibly around 35° . Preliminary work has changed the number of parameters in an effort to maximize the quality of the fit with a minimal number of parameters. Figure 4.6 compares the current fitting to the tentative new fitting for three of the six 1D subsets. These modified fits in general seem to fit better at the cost of additional parameters, with a smaller χ^2_{red} value. Still more parameters may be needed to accommodate additional cusps however. A Levenberg-Marquardt MATLAB® script calculated these tentative fits in part.²¹

Figure 4.5 shows the comparison between the values calculated from the P and Q matrices and the expected values from the MD calculations for the $\langle 100 \rangle$ subset. Figures C.4 to C.6 shows all six subsets. The $\langle 100 \rangle$ tilt subset has an unsolved scaling issue. Overall, the results from the P and Q matrices match the fitted values, with a few anomalies needing to be addressed.

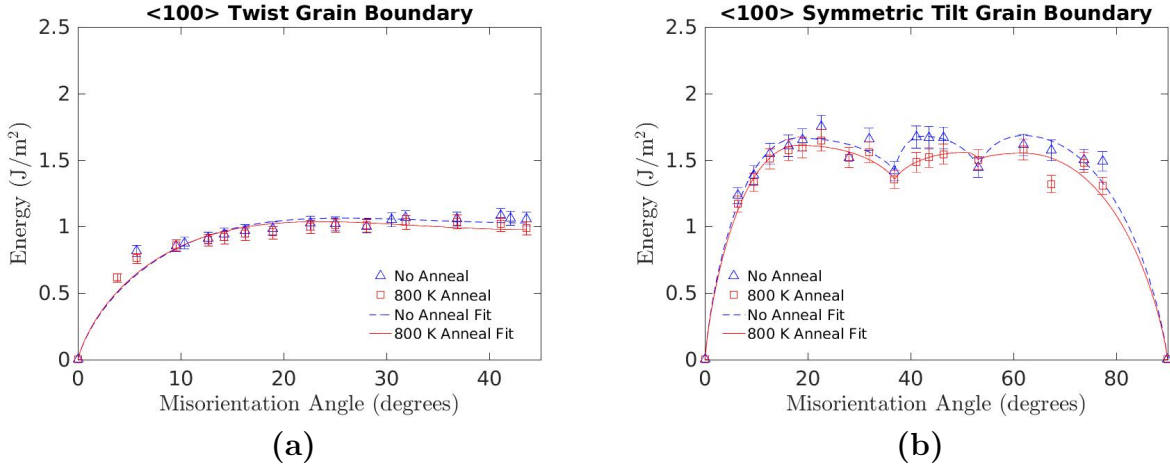


Figure 4.2 The $\langle 100 \rangle$ twist (a) and tilt (b) results. In general the re-calculated energies are lower, with significant differences around 40° to 50° in the tilt subset. The unexpected positive concavity in the twist subset around 40° may indicate the presence of a missing cusp. Possible cusps exist around 30° in the twist subset, and around 68° in the tilt subset.

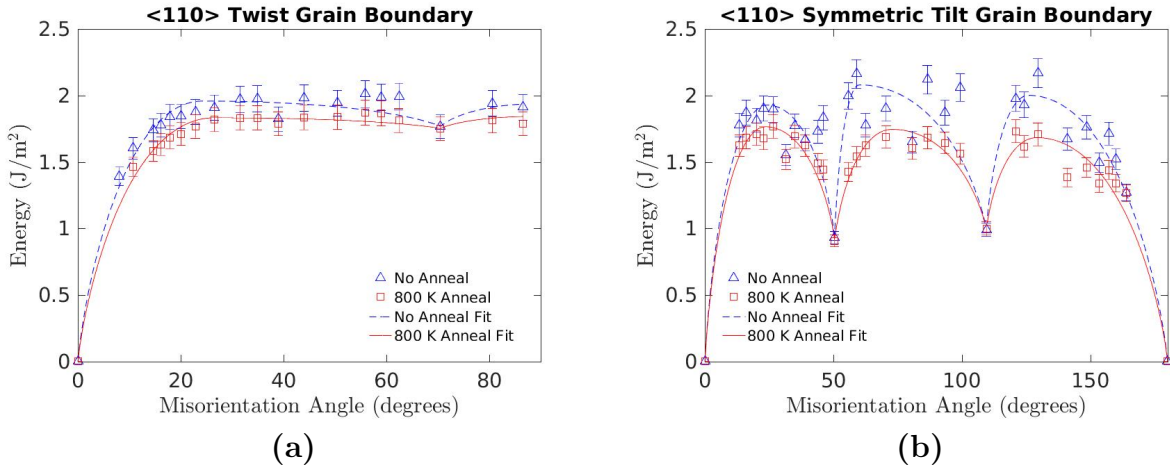


Figure 4.3 The $\langle 110 \rangle$ twist (a) and tilt (b) results. Both subsets have significant decreases in energy. The twist subset has a possible cusp at around 40° , and the tilt subset has possible cusps around 40° , 90° , and 140° .

4.3 Reduced Chi-square Results

The χ_{red}^2 values are much smaller than one for every data set regardless of the method used to calculate the statistic, with the exception of the $\langle 100 \rangle$ symmetric tilt subset using the P and Q matrices. This subset has a high χ_{red}^2 value due to the scaling issue. Because of the

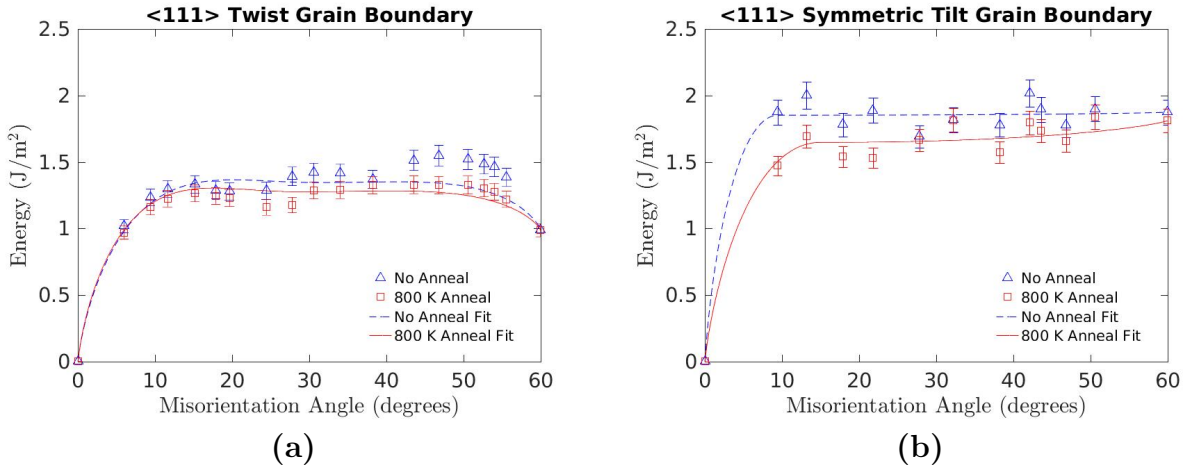


Figure 4.4 The $\langle 111 \rangle$ twist (a) and tilt (b) results. Most energies are found to be lower, but some are found to be higher. The unexpected positive concavity present in these results could indicate the presence of one or more cusps, with one possible location around 33°. This work needs additional data to determine possible cusp locations for the tilt subset.

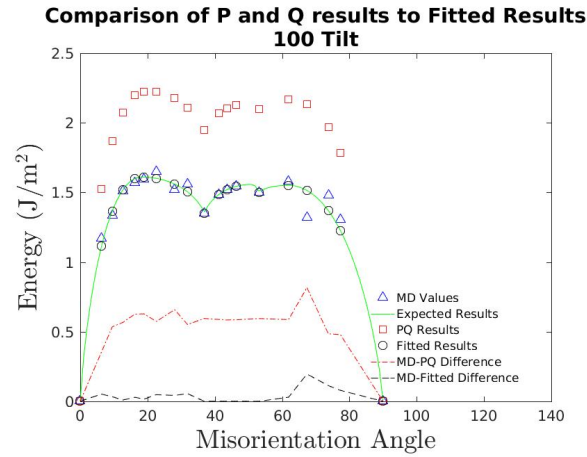


Figure 4.5 A comparison of the expected value of the fitted function with the values calculated using the P and Q matrices for the $\langle 100 \rangle$ 1D tilt subset, with MD values shown for reference. The cause of the scaling issue remains unknown.

low χ_{red}^2 values, the fitted functions overfit the data.¹⁵ Table 4.1 lists the χ_{red}^2 values for the 1D subsets using the two different methods for calculation.

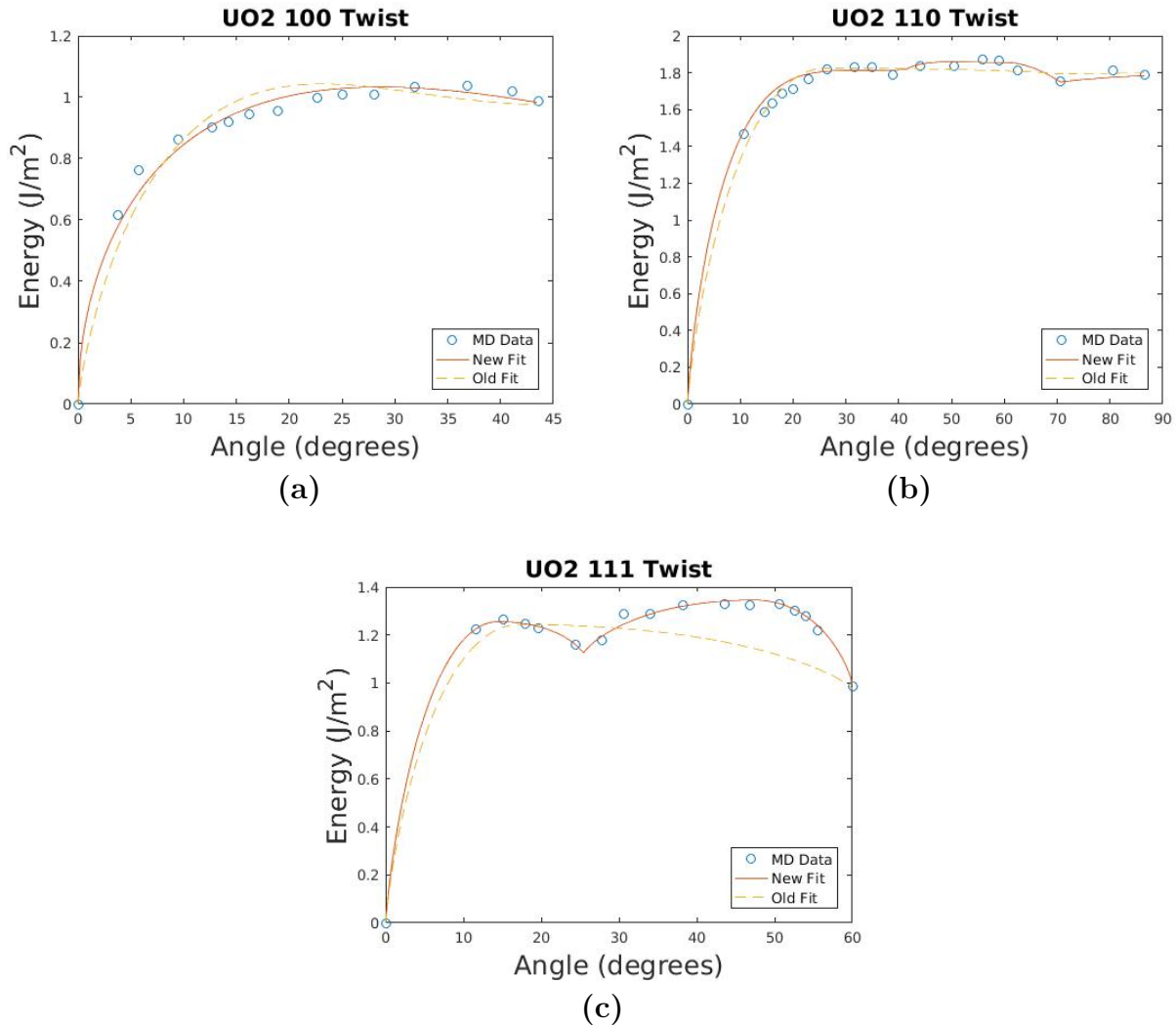


Figure 4.6 A comparison of current fitting functions with a possible change to the functions. Dashed lines show the original functions, and the solid lines show the updated functions, with MD results shown for reference. (a) shows a possible change from the Read-Shockley-Wolf (RSW) functions to a simple square root function multiplied by an exponential decay without any theoretical basis. (b) attempts to fit to a cusp around 40°. Further work can be done to find a better fit for this subset. (c) shows the most potential improvement. The potential fit increases the total number of parameters by three to fit to the cusp around 28°. A quick glance at the MD values compared to the fit shows a great improvement from the current fit. To create the graph shown in (c) this work spliced two additional RSW functions into the original function. This created a total of four RSW functions for this subset.

Table 4.1 A list of the χ^2_{red} results using two different methods: using the P and Q matrices for the various orientations to test the fit, and comparing the results of the 1D fits to the 1D data. The values for χ^2_{red} are all less than one with the exception of the $\langle 100 \rangle$ symmetric tilt using the P and Q matrices. These values indicate an over-fit to the data.

1D Subset	χ^2_{red} using P and Q matrices		χ^2_{red} comparing the 1D fits	
	No Anneal	800 K Anneal	No Anneal	800 K Anneal
$\langle 100 \rangle$ Twist	0.0953	0.1074	0.0752	0.0722
$\langle 110 \rangle$ Twist	0.1010	0.1874	0.0400	0.0137
$\langle 111 \rangle$ Twist	0.3041	0.1139	0.4966	0.1516
$\langle 100 \rangle$ Tilt	0.1038	8.7702	0.0846	0.0932
$\langle 110 \rangle$ Tilt	4.9799	0.3277	0.5951	0.1762
$\langle 111 \rangle$ Tilt	0.1566	0.7814	0.1315	0.1355
Overall χ^2_{red}	1.7652	1.4893	0.2678	0.1153

Chapter 5

Conclusion

This work has successfully created a more accurate interpolation function for grain boundary (GB) energies in uranium dioxide (UO_2), but revealed additional characteristics of the full five-dimensional (5D) GB space that require further research. GB energies were found to be lower when calculated with an 800 K anneal as expected when compared to the data set created without an anneal. Descriptions of those additional characteristics through the use of additional functions will prove beneficial.

Future work should focus on calculating additional data points for fitting. Increasing the number of data points will improve the quality of the fit. Additional data points will also help to identify trends that do not readily appear with the limited data currently available.

Further work could generalize this function to all polycrystalline materials with the fluorite crystal structure. Such a generalization would provide further validation of this model, however, such validation would require a sufficient number of GB energies for many different materials with such a structure. The literature does not provide these energies, indicating a need for additional computational resources.

As the model develops, initial conditions set by actual nuclear fuel data will be put into a MARMOT simulation. After simulating grain growth in a nuclear reactor, the fuel data will be compared with the simulation data to determine the accuracy of the model.

Bibliography

- ¹ Thomas Jefferson National Accelerator Facility – Office of Science Education, *The Element Uranium*, education.jlab.org/itselemental/ele092.html, Accessed: 13 October 2016
- ² Y. Zhang, *Unpublished*, Personal Communication
- ³ E. Hansen, *Unpublished*, Personal Communication
- ⁴ T. Harbison, *Anisotropic grain boundary energy function for uranium dioxide*, B.S. Thesis, Brigham Young University - Idaho (2015)
- ⁵ W. D. Callister Jr., *Materials Science and Engineering: An Introduction* (John Wiley & Sons Inc, USA, 2003)
- ⁶ S. Patala and C. A. Schuh, *Symmetries in the representation of grain boundary-plane distributions*, Philosophical Magazine **93** (2013), pp. 524–573
- ⁷ P. Lejček, *Grain Boundaries: Description, Structure and Thermodynamics* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2010), pp. 5–24
- ⁸ E. R. Homer, S. Patala and J. L. Priedeman, *Grain boundary plane orientation fundamental zones and structure-property relationships*, Sci. Rep. **5** (2015), p. 15476
- ⁹ V. V. Bulatov, B. W. Reed and M. Kumar, *Grain boundary energy function for fcc metals*, Acta Mater. **65** (2014), pp. 161–175

-
- ¹⁰ G. S. Rohrer, *Grain Boundary Energy Anisotropy: A Review*, J. Mater. Sci. **46** (2011), pp. 5881–5895
- ¹¹ S. Plimpton, *Fast Parallel Algorithms for Short-Range Molecular Dynamics*, Journal of Computational Physics **117** (1995), pp. 1–19
- ¹² A. S. Butterfield, *Exploration of the phase-field framework MARMOT to include anisotropic grain boundaries with molecular dynamics*, B.S. Thesis, Brigham Young University - Idaho (2013)
- ¹³ H. T. Stokes, *Solid State Physics: For advanced undergraduate students* (BYU Academic Publishing, Provo, Utah, 2007)
- ¹⁴ D. Wolf, *A Read-Shockley model for high-angle grain boundaries*, Scripta Metallurgica **23** (1989), pp. 1713–1718
- ¹⁵ P. R. Bevington and D. K. Robison, *Data Reduction and Error Analysis for the Physical Sciences* (McGraw-Hill, New York, NY, 2003)
- ¹⁶ M. R. Tonks, D. Gaston, P. C. Millett, D. Andrs and P. Talbot, *An object-oriented finite element framework for multiphysics phase field simulations*, Computational Materials Science **51** (2012), pp. 20–29
- ¹⁷ V. Randle and O. Engler, *Introduction to Texture Analysis: Macrotexture, Microtexture and Orientation Mapping* (CRC Press, USA, 2000)
- ¹⁸ S. Belongie, *Rodrigues Rotation Formula From MathWorld—A Wolfram Web Resource, created by Eric W. Weisstein* (2006), mathworld.wolfram.com/RodriguesRotationFormula.html
- ¹⁹ F. Bachmann, R. Hielscher and H. Schaeben, *Texture Analysis with MTEX – Free and Open Source Software Toolbox*, Solid State Phenomena **160** (2010), pp. 63–68

-
- ²⁰ E. W. Weisstein, *Quaternion From MathWorld—A Wolfram Web Resource* (2004), mathworld.wolfram.com/Quaternion.html
- ²¹ H. P. Gavin, *The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems* (2016)
- ²² F. C. Frank, *Orientation Mapping*, Metallurgical Transactions A **19A** (1988), pp. 403–408
- ²³ A. Morawiec and D. P. Field, *Rodrigues parameterization for orientation and misorientation distributions*, Phil. Mag. A **73** (1996), pp. 1113–1130
- ²⁴ R. Becker and S. Panchanadeeswaran, *Crystal rotations represented as Rodrigues vectors*, Textures and Microstructures **10** (1989), pp. 167–194
- ²⁵ L. Priester, *Geometrical Order of Grain Boundaries* (Springer Netherlands, Dordrecht, 2013), pp. 3–28
- ²⁶ D. M. Kirch, *Fundamentals of grain boundaries and triple junctions* (Cuvillier, Göttingen, 2008), pp. 1–7
- ²⁷ S. Patala, J. K. Mason and C. A. Schuh, *Improved representations of misorientation information for grain boundary science and engineering*, Progress in Materials Science **57** (2012), pp. 1383–1425

Appendix A

List of Parameters

Table A.1 This table gives the parameters for UO_2 generating the energy function.

Array number	Parameter name	Parameter value
1	Energy Scaling Factor (e_{RGB})	1.6012 J/m^2
2	$\langle 100 \rangle$ Max Distance	0.405
3	$\langle 110 \rangle$ Max Distance	0.739
4	$\langle 111 \rangle$ Max Distance	0.352
5	$\langle 100 \rangle$ Weight	50.5
6	$\langle 110 \rangle$ Weight	4.55
7	$\langle 111 \rangle$ Weight	0.08
8	$\langle 100 \rangle$ Tilt/Twist Mix Power Law (1)	0.03325
9	$\langle 100 \rangle$ Tilt/Twist Mix Power Law (2)	0.00053125
10	Maximum $\langle 100 \rangle$ Twist Energy	0.60903
11	$\langle 100 \rangle$ Twist Shape Factor	1.4486
12	$\langle 100 \rangle$ Asymmetric Tilt Interpolation Power	35.8
13	$\langle 100 \rangle$ Symmetric Tilt First Peak Energy	1.0058

Continued on next page.

Table A.1 – *Continued from previous page*

Array number	Parameter name	Parameter value
14	$\langle 100 \rangle$ Symmetric Tilt First $\Sigma 5$ Energy	0.84456
15	$\langle 100 \rangle$ Symmetric Tilt Second Peak Energy	0.97259
16	$\langle 100 \rangle$ Symmetric Tilt Second $\Sigma 5$ Energy	0.9379
17	$\langle 100 \rangle$ Symmetric Tilt $\Sigma 17$ Energy	0.96881
18	$\langle 100 \rangle$ Symmetric Tilt First Peak Angle	0.31569
19	$\langle 100 \rangle$ Symmetric Tilt Second Peak Angle	0.88538
20	$\langle 110 \rangle$ Tilt/Twist Mix Power Law (1)	3.1573
21	$\langle 110 \rangle$ Tilt/Twist Mix Power Law (2)	1.9784
22	$\langle 110 \rangle$ Twist Peak Angle	0.46145
23	$\langle 110 \rangle$ Twist Peak Energy	1.1444
24	$\langle 110 \rangle$ Twist $\Sigma 3$ Energy	1.0931
25	$\langle 110 \rangle$ Twist 90° Energy	1.152
26	$\langle 110 \rangle$ Asymmetric Tilt Shape Factor	3.1843
27	$\langle 110 \rangle$ Symmetric Tilt Third Peak Energy	1.0514
28	$\langle 110 \rangle$ Symmetric Tilt $\Sigma 3$ Energy	0.61703
29	$\langle 110 \rangle$ Symmetric Tilt Second Peak Energy	1.0902
30	$\langle 110 \rangle$ Symmetric Tilt $\Sigma 11$ Energy	0.56686
31	$\langle 110 \rangle$ Symmetric Tilt First Peak Energy	1.1024
32	$\langle 110 \rangle$ Symmetric Tilt Third Peak Angle	0.88736
33	$\langle 110 \rangle$ Symmetric Tilt Second Peak Angle	1.8711
34	$\langle 110 \rangle$ Symmetric Tilt First Peak Angle	2.731
35	$\langle 111 \rangle$ Tilt-Twist Linear Interpolation	38.201
36	$\langle 111 \rangle$ Twist Shape Factor	1.2414
37	$\langle 111 \rangle$ Twist Peak Angle	0.49979

Continued on next page.

Table A.1 – *Continued from previous page*

Array number	Parameter name	Parameter value
38	$\langle 111 \rangle$ Twist Peak Energy	0.7971
39	$\langle 111 \rangle$ Symmetric Tilt Peak Angle	0.25966
40	$\langle 111 \rangle$ Symmetric Tilt Max Energy	1.0288
41	$\langle 111 \rangle$ Symmetric Tilt $\Sigma 3$ Energy	1.1311
42	$\langle 111 \rangle$ Asymmetric Tilt Symmetry Point Energy	3.7674
43	$\langle 111 \rangle$ Asymmetric Tilt Scale Factor	0.053417

Appendix B

Grain Boundary Representations

Visual representations of the GB space helped Bulatov *et al.* develop their 5D function. However, the size of the five-space in which GBs reside makes representing them difficult. Researchers have developed different methods to represent them, each with their advantages and disadvantages. Three of these methods are the axis-angle representation, the Rodrigues representation, and the fundamental zone representation. These methods, though described separately, can be used together to form a better picture of what the GB space looks like (see for example Figure 3.2b which combines the Rodrigues representation and the fundamental zone representation).

B.1 Axis-Angle Representation

Of the three described, the axis-angle representation most simplistically describes GB space. The axis of rotation of the GB specifies the point in axis-angle space, and the angle of misorientation between the two grains at the GB specifies the magnitude of the vector. Thus, the axis (\mathbf{a} , where \mathbf{a} has components a_x , a_y , and a_z) and the angle (θ) mathematically represent an axis-angle vector as:

$$\mathbf{A} = \mathbf{a} \theta \tag{B.1}$$

The axis-angle space can only take into account three degrees of freedom: the two angles specifying the axis, and the angle rotated through. Thus, axis-angle space cannot fully visualize all of the necessary information contained in the full 5D space.²² This representation suffers from the difficulties of understanding an infinite space because it maps an axis and an angle onto a Cartesian coordinate system. Without the help of additional methods, this infinite space remains difficult to understand. The best uses of this representation focus on using it as a starting point to move to other, more robust representations, and to represent the misorientation between two grains.¹⁷

B.2 Rodrigues Representation

The Rodrigues representation (sometimes called the “Rodrigues-Frank” representation) uses Rodrigues vectors to represent rotations in Rodrigues space. This representation takes ideas from the axis-angle space, but makes a few changes allowing crystal symmetries to be taken into account. The orientation of the GB normal still specifies the point in space, but the tangent of half the angle represents the magnitude of the vector. Thus, a Rodrigues vector can be represented as:^{17, 22–25}

$$\mathbf{R} = \mathbf{a} \tan\left(\frac{\theta}{2}\right) \quad (\text{B.2})$$

Some researchers favor this representation over others because of the lack of curvature such a mapping entails.^{17, 22} However, it still only specifies three of the five degrees of freedom. Bulatov *et al.* attached a unit vector at the points along the axis to represent the other two DoFs in Figure 3.2b. A parallel vector represents a twist boundary, and a perpendicular vector represents a tilt boundary. Anything else represents a mix of twist and tilt (or a mixed boundary). One limitation of Rodrigues space lies in that it also maps to an infinite space.^{22, 26}

B.3 Fundamental Zone Representation

The fundamental zone best graphically represents the full 5D GB. This representation takes advantage of the symmetries inherent in crystals¹³ to simplify an infinite space into a compact, finite area called the fundamental zone.^{6,8,9,23,27} Every point within the space represents a unique orientation, and every point outside the space can be represented as a point inside the space through symmetry operations.²²⁻²⁴ Bulatov *et al.* used this idea in connection with Rodrigues space to create Figure 3.2b. In Rodrigues space, the crystal symmetries of the material determine the shape of the fundamental zone.^{6,23} For fcc crystals, the fundamental zone takes the form of a truncated tetrahedron.⁹ The edges of the fundamental zone in Rodrigues space represent the high-symmetry rotation axes, and points on one face can represent another point on a different face of the fundamental zone.

Appendix C

Graphs

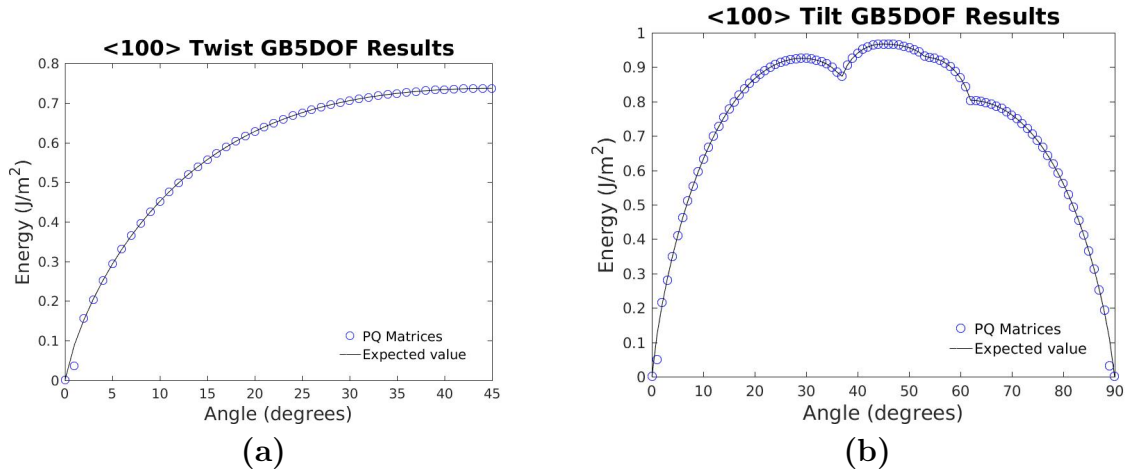


Figure C.1 The $\langle 100 \rangle$ twist (a) and tilt (b) results for the P and Q matrices as compared to Bulatov *et al.*'s energy profiles. Bulatov *et al.*'s GB5DOF.m MATLAB[®] script calculated the expected values by using the default parameters. The GB5DOF.m script calculated the values using the generated matrices. With the exception of the data points at 1° in both (a) and (b) and 89° in (b), the energies calculated from the matrices matches the expected curves exactly.

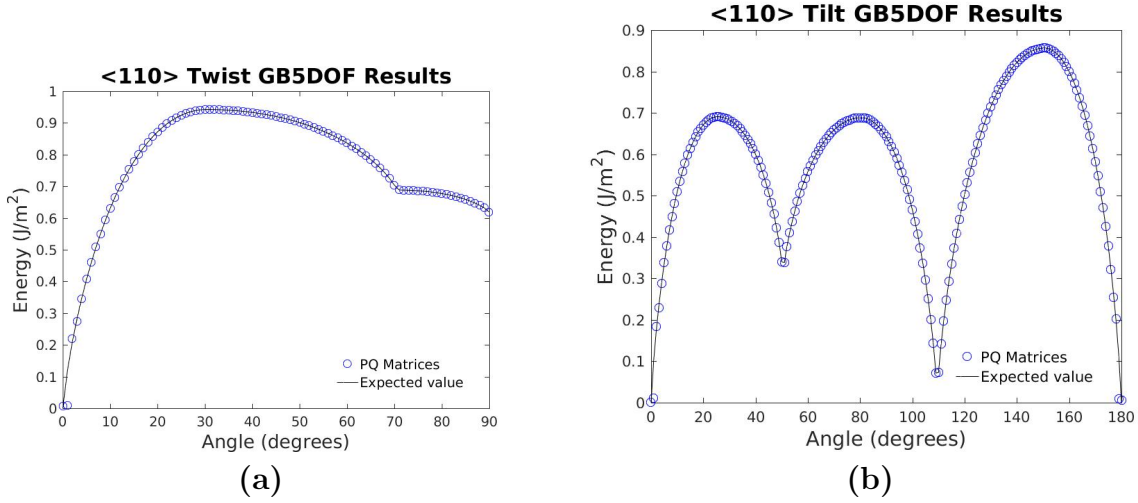


Figure C.2 The $\langle 110 \rangle$ twist (a) and tilt (b) results for the P and Q matrices as compared to Bulatov *et al.*'s energy profiles. Bulatov *et al.*'s GB5DOF.m MATLAB[®] script calculated the expected values by using the default parameters. The GB5DOF.m script calculated the values using the generated matrices. With the exception of the data points at 1° in both (a) and (b) and 179° in (b), the energies calculated from the matrices matches the expected curves exactly.

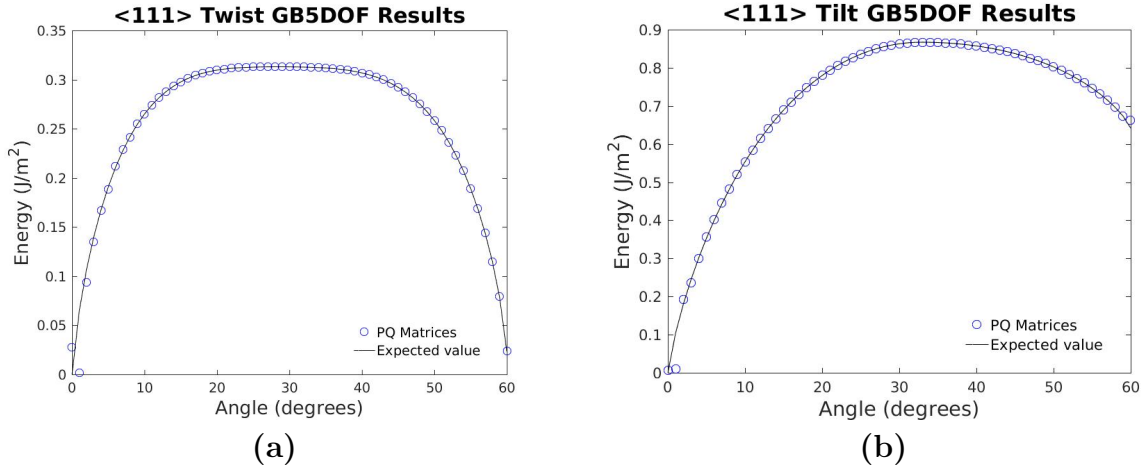


Figure C.3 The $\langle 111 \rangle$ twist (a) and tilt (b) results for the P and Q matrices as compared to Bulatov *et al.*'s energy profiles. Bulatov *et al.*'s GB5DOF.m MATLAB[®] script calculated the expected values by using the default parameters. The GB5DOF.m script calculated the values using the generated matrices. With the exception of the data points at 1° in both (a) and (b) and 60° in (b), the energies calculated from the matrices matches the expected curves exactly.

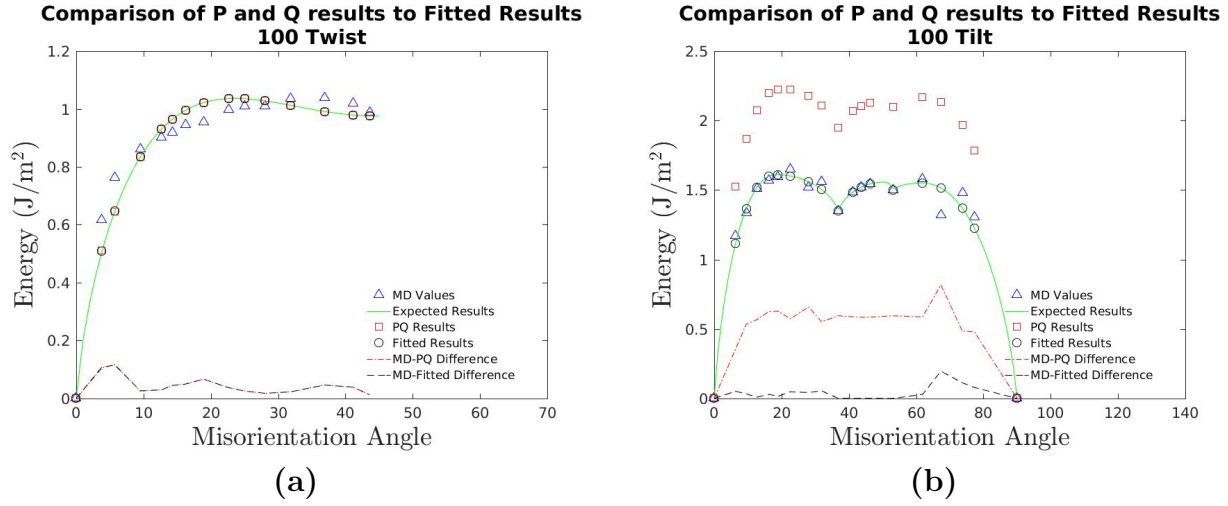


Figure C.4 A comparison of the expected value of the fitted function with the values calculated using the P and Q matrices for the $\langle 100 \rangle$ 1D subsets, with the MD values shown for reference. (a) PQ results follow exactly the fitted curve. (b) has a scaling issue yet to be fixed. The cause of the scaling issue remains unknown.

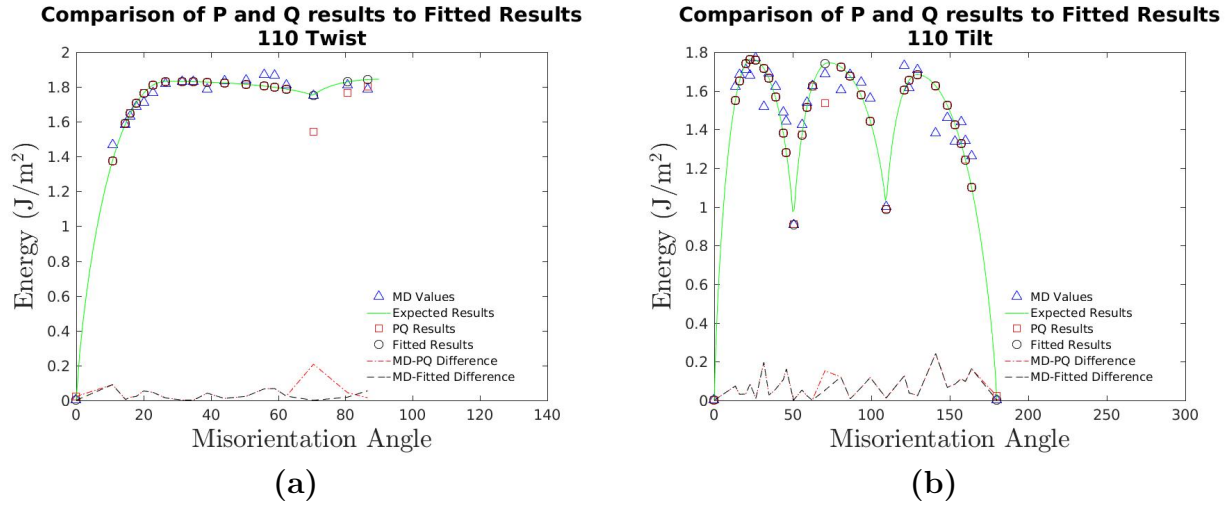


Figure C.5 A comparison of the expected value of the fitted function with the values calculated using the P and Q matrices for the $\langle 110 \rangle$ 1D subsets, with the MD values shown for reference. (a) follows the fitted result until the cusp, at which point some anomalies appear. The results from the PQ matrices dip well below the expected value at the cusp, and never make it back to the original fitted line. (b) has a similar issue on a lesser scale. Only two of the calculated points do not follow the fitted curve. At the endpoint the expected value is zero, but the PQ matrices calculated a value slightly higher. Also, an unexpected cusp from the PQ matrices appears in the middle of the second hump. All other data points follow the fitted curve exactly.

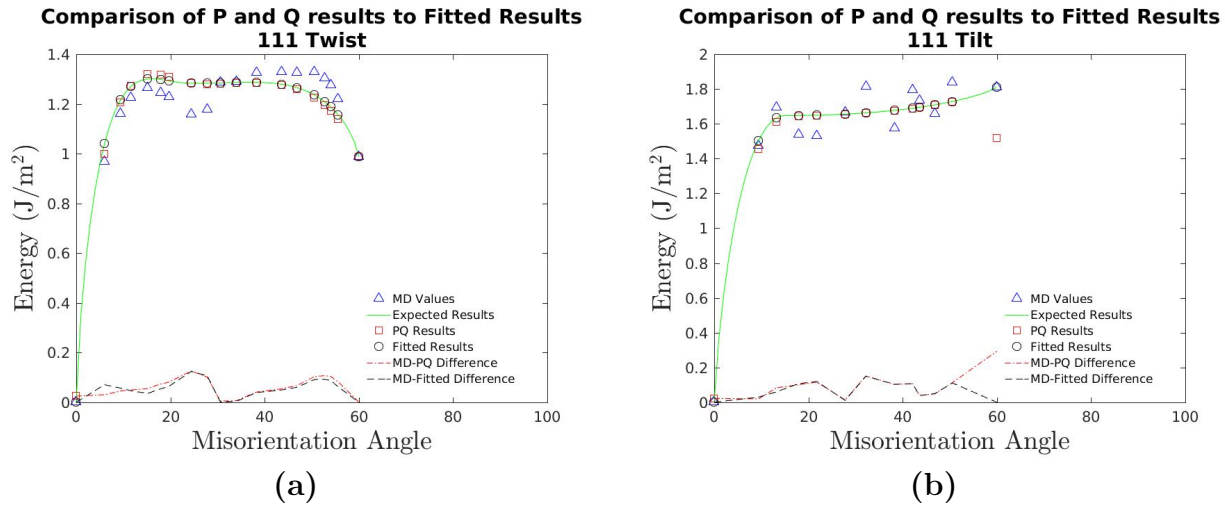


Figure C.6 A comparison of the expected value of the fitted function with the values calculated using the P and Q matrices for the $\langle 111 \rangle$ 1D subsets, with the MD values shown for reference. (a) closely follows the expected fitted values, but has a slight error throughout. (b) follows the expected values exactly in the center of the fitting, but misses slightly for lower angle boundaries, and misses completely at the end.

Appendix D

Orientation Matrix Generator

This code generates the orientation matrices (known as the P and Q matrices in Bulatov *et al.*'s code). Provision for calculating the matrices one of two ways appears in-code through the use of command-line options.

```
from __future__ import division, print_function # To avoid numerical
    problems with division, and for ease of printing
2 from sys import argv # for CLI arguments
    from math import cos, sin, pi, atan2, sqrt # Trig functions
4 from os.path import exists # For checking existence of a file
    from numpy import array, linalg
6 from myModules import * # imports my functions from the file myModules.py

8 # Helper functions
def displayHelp():
10     print(''''
        This script will calculate the orientation matrices for any given
            misorientation
12     for any of the high-symmetry axes.
        Arguments:
14
            _axis: The axis of orientation (type: int)
```

```

16     _misorientation: The angle of misorientation (type: float)
           -----OR-----
18     (with option -e or --euler)
           _z1: The first rotation angle  (Z ) (type: float)
20     _x:   The second rotation angle (X') (type: float)
           _z2: The third rotation angle  (Z") (type: float)
22
    If the option -e or --euler-angles is entered, the calculation skips
        to simply
24    output the orientation matrices.  Otherwise, the Euler angles are
        calculated from
        the axis, orientation, and grain boundary normal, and then the
            orientation matrix is
26    created through the use of the Rodrigues Rotation Formula, which is:

$$R = I + \sin(\theta) * K + (1 - \cos(\theta)) * K^2$$

28    where I is the identity matrix, theta is the misorientation angle, and
        K is
        the skew-symmetric matrix formed by the axis of rotation:
30    
$$K = \begin{pmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{pmatrix}$$

32
        where the vector k is the unit vector defining the axis of rotation,
            or using
34    a set of predefined rotations for each axis (default is the predefined
            rotations).
        The Euler angles are calculated in this case simply for the file to be
            written
36    to.  If the user does not specify to save, then the angles are not
            used for
        anything.
38
    Options:

```

40	<code>-e --euler <_z1> <_x> <_z2></code>	Returns the Bunge orientation
	<code>matrix</code>	
		based on the euler angles provided
		.
42		
	<code>-f --file <filename></code>	Reads the file filename and uses
	<code>the</code>	
44		Euler angles from them to
		calculate the
		orientation matrix.
46		
	<code>--rrf</code>	Calculates the matrices using the
	<code>Rodrigues</code>	
48		Rotation Formula
50	<code>-a --angles</code>	Displays the Euler angles. Can be
	<code>used</code>	
		in conjunction with <code>-q</code> or <code>--quiet</code>
		to
52		display only the Euler angles.
54	<code>-s --save</code>	Saves the resultant orientation
	<code>matrix to</code>	
		a database (
		<code>orientation_matrix_database.m</code>)
56		with the accompanying Euler angles
		.
58	<code>-q --quiet</code>	Suppresses output of the
	<code>orientation matrices</code>	
		to the terminal
60		

```

--help                                Displays this help info

62
Output:
64 For an Euler angle set, the output is simply its orientation matrix.
For the misorientations, the first matrix is the 'P' orientation
    matrix, and
66 the second matrix is the 'Q' orientation matrix (see Bulatov et al.,
    Acta Mater
65 (2014) 161-175).
68 '''
    return

70
def displayAngles(z1, x, z2): # Displays an Euler angle set (Bunge
    convention)
72     print("Euler angles:")
    # This is the "new" way to format strings. The 16 indicates the
        padding to
74     # be done before the next character. The '<' character below says
        which side
        # to pad (the right side).
76     print("{:16}{:16}{:16}".format('Z', 'X', 'Z'))
    print("-----")
78     print("{:<16}{:<16}{:<16}\n".format(rad2deg(z1), rad2deg(x), rad2deg(
        z2)))
    return

80
def check4RRF(args): # Check the args for the rrf command
82     if "--rrf" in args:
        index = args.index("--rrf")
84         del args[index]
        return True, args
86     else:

```

```

        return False, args

88
def check4Euler(args): # Check the args for the -a or --angles command
90     if "-a" in args or "--angles" in args:
        try:
92         index = args.index("-a")
        except:
94         index = args.index("--angles")
        del args[index]
96     return True, args
    else:
98     return False, args

100 # Write the matrix and angles to a file
def writeMat(m, _z1, _x, _z2, grain, axis):
102     # This is to avoid issues with duplicates
    if _z1 == 0:
104         _z1 = abs(_z1)
    if _x == 0:
106         _x = abs(_x)
    if _z2 == 0:
108         _z2 = abs(_z2)

110     lastVal = 1
    # This is the default filename to be used.
112     # TODO: make provisions to provide the database file via command line
    tex_filename = "orientation_matrix_database.m"
114     var_name = "%s%d"%(grain, axis) # Will generally look like P100 or
        Q100
    if not exists(tex_filename):
116         tex_file = open(tex_filename, "a")

```

```

tex_file.write("%Database for orientation matrices for specified
               Euler Angles\n")
118 tex_file.write("
               %-----\n")
tex_file.write("%Orientation Matrix
               Euler Angles\n")
120 tex_file.write("%s(:, :, %d)=[%2.6f  %2.6f  %2.6f          %2.4f
               %2.4f  %2.4f\n"%(var_name, lastVal, m[0][0], m[0][1], m[0][2],
               _z1, _x, _z2))
tex_file.write("%2.6f  %2.6f  %2.6f\n"%(m[1][0], m[1][1], m[1][2])
               )
122 tex_file.write("%2.6f  %2.6f  %2.6f];\n"%(m[2][0], m[2][1], m
               [2][2]))
tex_file.write("
               %-----\n")
124 tex_file.close()
else:
126 f = open(tex_filename, "r")
while True:
128     data = f.readline().split()
    if not data:
130         break
    elif len(data) != 6:
132         continue
    else:
134         assert data[0][0] in {'P', 'Q'}, "Unknown orientation
               matrix type (should be \'P\' or \'Q\')."
        if not "%d"%(axis) in data[0][1:4]:
136             lastVal = 0
        elif "%d"%(axis) in data[0][1:4]:

```

```

138         try:
139             try:
140                 if data[0][0] == 'P': # Handles anything 3
141                     digits long
142                     lastVal = int(data[0][9:12]) - 1
143             else:
144                 lastVal = int(data[0][9:12])
145         except:
146             if data[0][0] == 'P': # Handles anything 2
147                 digits long
148                 lastVal = int(data[0][9:11]) - 1
149             else: # data[0][0] == 'Q'
150                 lastVal = int(data[0][9:11])
151         except:
152             if data[0][0] == 'P': # One digit case
153                 lastVal = int(data[0][9]) - 1
154             else: # data[0][0] == 'Q'
155                 lastVal = int(data[0][9])
156     else:
157         print("Error: Unknown last index.")
158         exit()
159
160     # Checks to see if the Euler angles have already been used
161     before
162     # If so, the calculated matrix is not saved (assumed to
163     already
164     # be in the database)
165     if data[0][0] == grain and data[3] == ('%' + "%2.4f"%_z1)
166         and data[4] == "%2.4f"%_x and data[5] == "%2.4f"%_z2:
167         unique = False
168         break
169     else:

```

```

        unique = True
166     if unique:
        tex_file = open(tex_filename, "a")
168     tex_file.write("%s(:,:,%d)=[%2.6f  %2.6f  %2.6f
        %%2.4f  %2.4f  %2.4f\n"%(var_name, lastVal + 1, m[0][0], m
        [0][1], m[0][2], _z1, _x, _z2))
        tex_file.write("%2.6f  %2.6f  %2.6f\n"%(m[1][0], m[1][1], m
        [1][2]))
170     tex_file.write("%2.6f  %2.6f  %2.6f];\n"%(m[2][0], m[2][1], m
        [2][2]))
        tex_file.write("
        %-----\n")
172     tex_file.close()

    return

174
    if "--help" in argv: # Help info
176         displayHelp()
        exit()

178
    orientation_matrix = []
180 save, argv = check4Save(argv) # Save the file? Delete the save argument
    quiet, argv = check4Quiet(argv) # Checks for suppressing output. Delete
    the quiet argument.
182 useRRF, argv = check4RRF(argv) # Checks for using the RRF method. Delete
    the rrf argument.
    dispEuler, argv = check4Euler(argv) # Checks for displaying the Euler
    angles. Delete the angle argument
184
    # If the arguments come from a file...
186 if "-f" in argv or "--file" in argv: #input arguments come from file
    try:

```

```
188         try:
                index = argv.index("-f")
190         except:
                index = argv.index("--file")
192     except:
        print("ERROR: Unable to find filename.")
194        exit()
    filename = argv[index + 1]
196    try:
        f1 = open(filename, 'r')
198    except:
        print("ERROR: Unable to read file.", filename)
200
    while True: # Read the file line by line.
202        line = f1.readline()
        # break if we don't read anything.  If there are blank lines in
        the
204        # file, this will evaluate to TRUE!
        if not line:
206            break;
        data = line.split()
208        if len(data) != 4: # If there are less than 4 parts to the data,
            move along (format of file MUST be _z1 _x _z2 1.00)
            continue
210        else:
            # Convert the data to stuff we can use
212            _z1 = float(data[0])
            _x  = float(data[1])
214            _z2 = float(data[2])

            _z1 = deg2rad(_z1)
            _x  = deg2rad(_x)
```

```

218         _z2 = deg2rad(_z2)
           orientation_matrix = calcRotMat(_z1, _x, _z2)
220         if not quiet:
           displayMat(orientation_matrix)
222         if save:
           writeMat(orientation_matrix, _z1, _x, _z2, 'P', _axis)
224
           # Input is a set of euler angles
226 elif "-e" in argv or "--euler-angles" in argv:
           try:
228             try:
               index = argv.index("-e")
230             except:
               index = argv.index("--euler-angles")
232         except:
           print("ERROR: Unable to read Euler angles.")
234           exit()
           _z1 = float(argv[index + 1])
236           _x = float(argv[index + 2])
           _z2 = float(argv[index + 3])
238           _z1 = deg2rad(_z1)
           _x = deg2rad(_x)
240           _z2 = deg2rad(_z2)

           orientation_matrix = calcRotMat(_z1, _x, _z2)

244         if not quiet:
           displayMat(orientation_matrix)
246         if save:
           writeMat(orientation_matrix, _z1, _x, _z2, 'P', _axis)
248
           else:

```

```

250     if len(argv) < 3:
        print("ERROR: Not enough command line arguments.")
252     print("Input either an axis, and a misorientation, or a ZXZ Euler
        angle set with the option -e or --euler-angles.")
        displayHelp()
254     exit()

    try:
256         _axis = int(argv[1])
        _misorientation = float(argv[2])
258     except:
        print('''
260     ERROR: Command line argument(s) is (are) not of correct type.
        Please enter an int for argument 1, a float for argument 2, and an
            int for argument 3.
262     ''')
        exit()

264

    if not len(str(_axis)) == 3: # axis length greater than 3
266         print("ERROR: Argument 1 must by a 3 digit number like \'100\'.")
        exit()

268

    _misorientation = deg2rad(_misorientation) # Change input to radians
270    axis = [None]*3
    _z1 = [None]*2
272    _x = [None]*2
    _z2 = [None]*2
274    q = [None]*2
    for i in range(0, len(str(_axis))):
276        axis[i] = int(str(_axis)[i])

278    #

```

```

#-----The Actual Calculations
-----#
280 #
-----#

# First convert to a quaternion
282 # These functions are from a myModules.py.
q[0] = axis2quat(axis, _misorientation / 2)
284 q[1] = axis2quat(axis, -_misorientation / 2)

# Convert the quaternion to Euler Angles
286 for i in range(0, len(_z1)):
288     _z1[i], _x[i], _z2[i] = quat2euler(q[i])

290 #
-----#

# Using the Rodrigues Rotation Formula, defined as  $R = I + \sin(\theta) * K + (1 - \cos(\theta)) * K^2$ 
292 # with  $K = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}$ , and the
    components of
# k coming from the vector being rotated about. Theta is specified by
    the misorientation.

294 if useRRF:
    orientation_matrix1, orientation_matrix2 = calcRotMatRRF(axis,
        _misorientation)

296

# Normalize the matrices using their determinants
298 orientation_matrix1 = orientation_matrix1 / linalg.det(
    orientation_matrix1)

```

```

orientation_matrix2 = orientation_matrix2 / linalg.det(
    orientation_matrix2)

300

    if not quiet:
302        displayMat(orientation_matrix1)
        displayMat(orientation_matrix2)
304

    for i in range(0, len(_z1)):
306        if dispEuler:
            displayAngles(_z1[i], _x[i], _z2[i])
308

        if save:
310            assert i < 2, "ERROR: Too many Euler angles."
            if i == 0:
312                writeMat(orientation_matrix1, _z1[i], _x[i], _z2[i], '
                    P', _axis)
            else:
314                writeMat(orientation_matrix2, _z1[i], _x[i], _z2[i], '
                    Q', _axis)

#
-----

316 else:
    for i in range(0, len(_z1)):
318        orientation_matrix = calcRotMat(_z1[i], _x[i], _z2[i])

320        # Normalize the matrix using the determinant
        orientation_matrix = orientation_matrix / linalg.det(
            orientation_matrix)
322

    if not quiet: # Display the results
324        displayMat(orientation_matrix)

```

```
326         if dispEuler: # Display the Euler Angles
            displayAngles(_z1[i], _x[i], _z2[i])

328

        if save: # We only calculate 2 angles at a time. If there are
            more, that's a problem.

330         assert i < 2, "ERROR: Too many Euler angles."
            if i == 0:

332                 writeMat(orientation_matrix, _z1[i], _x[i], _z2[i], 'P
                    ', _axis)
            else:

334                 writeMat(orientation_matrix, _z1[i], _x[i], _z2[i], 'Q
                    ', _axis)
```

Appendix E

Rotation Matrix Generator

This code generates the rotation matrices used to rotate the axes to the $[100]$ direction as required by Bulatov *et al.*'s script.

```
from __future__ import division, print_function # Automatically divides as
    floats, and considers print() a function
2 from sys import argv # for CLI arguments
    from numpy import array, linalg # for matrix operations
4 from os.path import exists # For checking existence of a file
    from myModules import * # For using my user-defined functions
6
    # Helper functions
8 def displayHelp():
    print('''
10     This script will generate the rotation matrix for the given
        misorientation axis
    Input:
12     _rotation_axis      This specifies the axis around which the
        grains are
                            rotated. (type: int (100) or string ('100'))
14
```

```

    _gbnormal          This specifies the boundary plane normal.  (
        type:
16                    int (100)  or string ('100'))

Options:
18    -s --save          Saves the resultant rotation matrix to a
                        database
                        (rotation_matrix_database.m) with the
                        accompanying
20                    rotation axis and misorientation type.

22    -q --quiet        Suppresses output of the rotation matrix

24    --help            Display this help info.
Output:
26    The output displayed will be the resultant rotation matrix for the
        given
        misorientation.
28    '''

30 # This function is an adaptation from MOOSE RotationMatrix class.
    def rotVecToZ(vec): # Creates the rotation matrix to rotate vec to the z
        direction
32    # REALLY make sure vec is normalized
        vec = vec / linalg.norm(vec)
34
        # Initialize our vectors
36    v1 = array([[0.,0.,0.]])
        v0 = array([[0.,0.,0.]])
38
        # Temp vector that gives a prototype of v1 by looking at the smallest
        component of vec
40    w = array([[abs(vec[0][0]), abs(vec[0][1]), abs(vec[0][2])]])

```

```

    if ( (w[0][2] >= w[0][1] and w[0][1] >= w[0][0]) or (w[0][1] >= w
        [0][2] and w[0][2] >= w[0][0]) ):
42         v1[0][0] = 1.0

    elif ( (w[0][2] >= w[0][0] and w[0][0] >= w[0][1]) or (w[0][0] >= w
        [0][2] and w[0][2] >= w[0][1]) ):
44         v1[0][1] = 1.0

    else:
46         v1[0][2] = 1.0

48     # Gram-Schmidt method to find v1
    v1 = v1 - ((v1.dot(vec.T))*vec)
50     v1 = v1 / linalg.norm(v1)

52     # v0 = v1 x vec
    v0[0][0] = v1[0][1]*vec[0][2] - v1[0][2]*vec[0][1]
54     v0[0][1] = v1[0][2]*vec[0][0] - v1[0][0]*vec[0][2]
    v0[0][2] = v1[0][0]*vec[0][1] - v1[0][1]*vec[0][0]

56

    # Rotation matrix is just:
58     rot = array([[v0[0][0], v0[0][1], v0[0][2]],
                   [v1[0][0], v1[0][1], v1[0][2]],
60                   [vec[0][0], vec[0][1], vec[0][2]]])

    return rot

62

def rotVec1ToVec2(vec1, vec2):
64     rot1_to_z = rotVecToZ(vec1)
    rot2_to_z = rotVecToZ(vec2)
66     return (rot2_to_z.T).dot(rot1_to_z)

68 def writeMat(m, _axis, gbnormal): # Write the matrix and angles to a file
    tex_filename = "rotation_matrix_database.m"
70     normName = _axis + '_' + gbnormal

```

```

var_name = "rot%snorm"%(normName)
72 if not exists(tex_filename):
    tex_file = open(tex_filename, "a")
74 tex_file.write("%Database for rotation matrices for specified
    misorientation axes\n")
    tex_file.write("
        %-----\n")
76 tex_file.write("%Rotation Matrix\n")
    tex_file.write("%s=[%2.4f  %2.4f  %2.4f\n"%(var_name, m[0][0], m
        [0][1], m[0][2]))
78 tex_file.write("%2.4f  %2.4f  %2.4f\n"%(m[1][0], m[1][1], m[1][2])
    )
    tex_file.write("%2.4f  %2.4f  %2.4f];\n"%(m[2][0], m[2][1], m
        [2][2]))
80 tex_file.write("
        %-----\n")
    tex_file.close()
82 else:
    # Check for already written
84 numlines = countFileLines(tex_filename)
    if numlines <= 4:
86 tex_file = open(tex_filename, "a")
        tex_file.write("%s=[%2.4f  %2.4f  %2.4f\n"%(var_name, m[0][0],
            m[0][1], m[0][2]))
88 tex_file.write("%2.4f  %2.4f  %2.4f\n"%(m[1][0], m[1][1], m
            [1][2]))
        tex_file.write("%2.4f  %2.4f  %2.4f];\n"%(m[2][0], m[2][1], m
            [2][2]))
90 tex_file.write("
            %-----\n")

```

```

        n")
    tex_file.close()
92     else:
        f = open(tex_filename, "r")
94     while True:
        data = f.readline().split()
96     if not data:
        break
98     elif len(data[0]) > 14:
        if not data[0][14] == '=':
100         continue
        else:
102         if data[0][0:10] == var_name:
            unique = False
104         else:
            unique = True
106     if unique:
        tex_file = open(tex_filename, "a")
108     tex_file.write("%s=[%2.4f  %2.4f  %2.4f\n"%(var_name, m
        [0][0], m[0][1], m[0][2]))
        tex_file.write("%2.4f  %2.4f  %2.4f\n"%(m[1][0], m[1][1],
        m[1][2]))
110     tex_file.write("%2.4f  %2.4f  %2.4f];\n"%(m[2][0], m
        [2][1], m[2][2]))
        tex_file.write("
        %-----\n")
112     tex_file.close()

114 # Check what we were given...
    if "--help" in argv: # Help info
116     displayHelp()
```

```

        exit()

118
    save, argv = check4Save(argv)
120 quiet, argv = check4Quiet(argv) # Checks for suppressing output

122 if len(argv) != 3: # if not both values given
    print("ERROR: Incorrect number of command line arguments. Line 150")
124    displayHelp()
    exit()
126 else: # len(argv) == 3
    script, _rotation_axis, _gbnormal = argv
128

    if not type(_gbnormal) in {int, str}:
130        print("ERROR: Grain boundary normal type is incorrect. Please enter
            an int or a string. Line 157")
        print("You entered %s with type %s"%(str(_gbnormal), type(_gbnormal)))
132        exit()
    else:
134        if type(_gbnormal) == int:
            _gbnormal = '0' + '0' + '0' + str(_gbnormal)
136            _gbnormal = _gbnormal[-3:] # gets the last three characters
            assert _gbnormal != '000', "ERROR: invalid boundary normal. Line
                173"
138

            assert(len(_rotation_axis) == 3), "ERROR: Something went wrong
                converting _gbnormal into a string. Line 166"
140

        if not type(_rotation_axis) in {int, str}:
142            print("ERROR: Grain boundary rotation axis type is incorrect. Please
                enter an int or a string. Line 169")
            print("You entered %s with type %s"%(str(_rotation_axis), type(
                _rotation_axis)))

```

```

144     exit()

else: # Convert anything besides a string into a string
146     if type(_rotation_axis) == int:
        _rotation_axis = '0' + '0' + '0' + str(_rotation_axis)
148     _rotation_axis = _rotation_axis[-3:] # Get the last three
        characters
        assert _rotation_axis != '000', "ERROR: invalid rotation axis.
            Line 176"

150
        assert(len(_rotation_axis) == 3), "ERROR: Something went wrong
            converting _rotation_axis into a string. Line 178"

152
    # Now that the input is taken care of, do the work
154 axis = array([[1, 0, 0]]) # This is the axis that we rotate the grain
    boundary normal to

156 # This part converts _gbnormal to an array for use in the rotation
    functions
    gbnormal = array([[None]*3])
158 j = 0
    indices = []
160 for i in range(0, len(gbnormal[0])):
    try:
162         gbnormal[0][i] = int(_gbnormal[j])
        j = j+1
164     except:
        #print(_gbnormal[i:i+2])
166         gbnormal[0][i] = int(_gbnormal[j:j + 2])
        j = j + 2
168         indices.append(i)

170

```

```
# So much work...
172 gbnorm = ''
    for i in range(0, len(gbnormal[0])):
174         if gbnormal[0][i] < 0:
            gbnorm += str(abs(gbnormal[0][i])) + 'bar'
176         else:
            gbnorm += str(gbnormal[0][i])
178 gbnormal = gbnormal / linalg.norm(gbnormal) # Normalize the gbnormal
        vector.
        axis = axis / linalg.norm(axis) # Just to be sure...
180 rotation_matrix = rotVec1ToVec2(gbnormal, axis)

182

184 if not quiet:
    displayMat(rotation_matrix)
186 if save:
    writeMat(rotation_matrix, _rotation_axis, gbnorm)
```

Appendix F

genOrientationMatrix.sh Bash Script

This bash script reads a CSV file containing misorientation angles data, and uses those angles to generate the P and Q matrices. This script calls the script `orientation_matrix.py`.

```
#!/bin/bash
2
# This script will generate the orientation matrices through python by
# looping
4 # through the CSV values given in the input files.
# Argument(s):
6 #     $1          Should be a filename that specifies the angles and relative
#                   energies for the 100, 110, and 111 symmetric tilt and twist
8 #                   boundaries

10 # Command-line argument counter that checks for the correct number of
#     arguments.
# Does not check for correct syntax.
12 if [ "$#" -ne 1 ]; then
#     echo "Illegal number of parameters"
14     exit 1
fi
16
```

```

# This takes the first argument from the command line - this is assumed to
# be a
18 # filename of the format 100Tilt.
    FN=$1
20
    echo "Determining the axis..."
22 # Pulls out the axis from the input file name. This uses regex syntax to
    find
    # a series of numbers that match either 100, 110, or 111. This also has
    an issue
24 # where it will find a match for 101, but as long as the files are named
    correctly
    # it shouldn't be an issue.
26 AXIS='echo $FN | grep -o "1[01][01]" '

28 echo "Reading the file..."
    IFS="," # separation character is the comma
30 # Exit with error code 99 if unable to read the file
    [ ! -f $FN ] && { echo "$FN file not found"; exit 99; }
32
    # This makes the assumption that the file orientation_matrix.py has
    executable
34 # rights.
    echo "Running the command: ~/projects/scripts/orientation_matrix.py $AXIS
        <angle> -s -q"
36 while read -r angle en; do # read the file with comma separated values

38     ~/projects/scripts/orientation_matrix.py $AXIS $angle -s -q
    done < "$FN" # the "$FN" is required if it's going to run properly!
40
    IFS=$OLDIFS # go back to the old separation character based on the system
    value.

```