,

# Appendix A

# Orientation Matrix Generator

This is the code used to generate 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 is provided in-code through the use of command-line options.

```
#! /usr/bin/env python

# This script will calculate the orientation matrices
    for any given misorientation
# for any of the high-symmetry axes.
# Arguments:
#
#    _axis: The axis of orientation (type: int)
#    _misorientation: The angle of misorientation (type:
     float)
#              ----------OR----------
#     (with option -e or --euler)
#    _z1: The first rotation angle  (Z ) (type: float)
#    _x:  The second rotation angle (X') (type: float)
#    _z2: The third rotation angle  (Z") (type: float)
#
# If the option -e or --euler-angles is entered, the
    calculation skips to simply
# output the orientation matrices.  Otherwise, the
    Euler angles are calculated from
```

```
# the axis, orientation, and grain boundary normal, and
    then the orientation matrix is
# created through the use of the Rodrigues Rotation
   Formula, which is:
# R = I + sin(theta) * K + (1 - cos(theta))*K^2
# where I is the identity matrix, theta is the
   misorientation angle, and K is
# the skew-symmetric matrix formed by the axis of
   rotation:
# K = 0  -kz   ky
#      kz   0  -kx
#    -ky   kx   0
# where the vector k is the unit vector defining the
   axis of rotation, or using
# 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
# to.  If the user does not specify to save, then the
   angles are not used for
# anything.
#
# Options:
# -e --euler <_z1> <_x> <_z2>              Returns the
   Bunge orientation matrix
#                                          based on the
   euler angles provided.
#
# -f --file <filename>                     Reads the file
   filename and uses the
#                                          Euler angles
   from them to calculate the
#                                          orientation
   matrix.
#
# --rrf                                    Calculates the
   matrices using the Rodrigues
```

```
#                                                        Rotation
   Formula
#
# -a --angles                                            Displays the
   Euler angles.  Can be used
#                                                        in conjunction
   with -q or --quiet to
#                                                        display only
   the Euler angles.
#
# -s --save                                              Saves the
   resultant orientation matrix to
#                                                        a database (
   orientation_matrix_database.m)
#                                                        with the
   accompanying Euler angles.
#
# -q --quiet                                             Suppresses
   output of the orientation matrices
#                                                        to the terminal
#
# --help                                                 Displays this
   help info
#
# Output:
# For an Euler angle set, the ouput is simply its
   orientation matrix.
# For the misorientations, the first matrix is the 'P'
   orientation matrix, and
# the second matrix is the 'Q' orientation matrix (see
   Bulatov et al., Acta Mater
# 65 (2014) 161-175).

from __future__ import division,print_function # To
   avoid numerical problems with division, and for ease
    of printing
from sys import argv # for CLI arguments
```

```
from math import cos, sin, pi, atan2, sqrt # Trig
    functions
from os.path import exists # For checking existence of
    a file
from numpy import array, linalg
from myModules import * # imports my functions from the
     file myModules.py

# Helper functions
def displayHelp():
     print('''
     This script will calculate the orientation matrices
          for any given misorientation
     for any of the high-symmetry axes.
     Arguments:

          _axis: The axis of orientation (type: int)
          _misorientation: The angle of misorientation (
             type: float)
               ----------OR----------
          (with option -e or --euler)
          _z1: The first rotation angle  (Z ) (type:
             float)
          _x:  The second rotation angle (X') (type:
             float)
          _z2: The third rotation angle  (Z") (type:
             float)

     If the option -e or --euler-angles is entered, the
        calculation skips to simply
     output the orientation matrices.  Otherwise, the
        Euler angles are calculated from
     the axis, orientation, and grain boundary normal,
        and then the orientation matrix is
     created through the use of the Rodrigues Rotation
        Formula, which is:
     R = I + sin(theta) * K + (1 - cos(theta))*K^2
```

*where I is the identity matrix, theta is the misorientation angle, and K is the skew−symmetric matrix formed by the axis of rotation:*

$$K = \begin{matrix} 0 & -kz & ky \\ kz & 0 & -kx \\ -ky & kx & 0 \end{matrix}$$

*where the vector k is the unit vector defining the axis of rotation, or using 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 to. If the user does not specify to save, then the angles are not used for anything.*

*Options:*

| | |
|---|---|
| *−e −−euler <_z1> <_x> <_z2>* | *Returns the Bunge orientation matrix based on the euler angles provided.* |
| *−f −−file <filename>* | *Reads the file filename and uses the Euler angles from them to calculate the orientation matrix.* |
| *−−rrf* | *Calculates the matrices using the Rodrigues Rotation Formula* |

```
-a --angles                                   Displays the
    Euler angles.  Can be used
                                              in conjunction
                                                  with -q or
                                                 --quiet to
                                              display only
                                                  the Euler
                                                  angles.

-s --save                                     Saves the
    resultant orientation matrix to
                                              a database (
                                                  orientation_matrix_database
                                                  .m)
                                              with the
                                                  accompanying
                                                   Euler
                                                  angles.

-q --quiet                                    Suppresses
    output of the orientation matrices
                                              to the terminal

--help                                        Displays this
    help info

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

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

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

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

# Write the matrix and angles to a file
def writeMat(m, _z1, _x, _z2, grain, axis):
    # This is to avoid issues with duplicates
```

```
if _z1 == 0:
    _z1 = abs(_z1)
if _x == 0:
    _x = abs(_x)
if _z2 == 0:
    _z2 = abs(_z2)

lastVal = 1
tex_filename = "orientation_matrix_database.m"
var_name = "%s%d"%(grain, axis)
if not exists(tex_filename):
    tex_file = open(tex_filename, "a")
    tex_file.write("%Database for orientation 
        matrices for specified Euler Angles\n")
    tex_file.write("
        %------------------------------------------------------------------------------------\
        n")
    tex_file.write("%Orientation Matrix             
                                   Euler Angles\n")
    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]))
    tex_file.write("%2.6f  %2.6f  %2.6f];\n"%(m
        [2][0], m[2][1], m[2][2]))
    tex_file.write("
        %------------------------------------------------------------------------------------\
        n")
    tex_file.close()
else:
    f = open(tex_filename,"r")
    while True:
        data = f.readline().split()
        if not data:
            break
        elif len(data) != 6:
```

```
        continue
    else:
        assert data[0][0] in {'P', 'Q'}, "
            Unknown orientation matrix type (
            should be \'P\' or \'Q\').  Line 192
            "
        if not "%d"%(axis) in data[0][1:4]:
            lastVal = 0
        elif "%d"%(axis) in data[0][1:4]:
            try:
                try:
                    if data[0][0] == 'P': #
                        Handles anything 3
                        digits long
                        lastVal = int(data
                            [0][9:12]) - 1
                    else:
                        lastVal = int(data
                            [0][9:12])
                except:
                    if data[0][0] == 'P': #
                        Handles anything 2
                        digits long
                        lastVal = int(data
                            [0][9:11]) - 1
                    else: # data[0][0] == 'Q'
                        lastVal = int(data
                            [0][9:11])
            except:
                if data[0][0] == 'P': # One
                    digit case
                    lastVal = int(data[0][9]) -
                        1
                else: # data[0][0] == 'Q'
                    lastVal = int(data[0][9])
        else:
            print("Error: Unknown last index. 
                Line 213")
```

```
                    exit()

                if data[0][0] == grain and data[3] == (
                    '%' + "%2.4f"%_z1) and data[4] == "
                    %2.4f"%_x and data[5] == "%2.4f"%_z2
                    :
                    unique = False
                    break
                else:
                    unique = True
        if unique:
            tex_file = open(tex_filename, "a")
            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]))
            tex_file.write("%2.6f  %2.6f  %2.6f];\n"%(m
                [2][0], m[2][1], m[2][2]))
            tex_file.write("
                %-----------------------------------------------------------------
                n")
            tex_file.close()
    return

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

orientation_matrix = []
save, argv = check4Save(argv) # Save the file?  Delete
    the save argument
quiet, argv = check4Quiet(argv) # Checks for
    suppressing output. Delete the quiet argument.
useRRF, argv = check4RRF(argv) # Checks for using the
    RRF method. Delete the rrf argument.
```

```python
dispEuler, argv = check4Euler(argv) # Checks for displaying the Euler angles.  Delete the angle argument

# If the arguments come from a file...
if "-f" in argv or "--file" in argv: #input arguments come from file
    try:
        try:
            index = argv.index("-f")
        except:
            index = argv.index("--file")
    except:
        print("ERROR: Unable to find filename. Line 248")
        exit()
    filename = argv[index + 1]
    try:
        f1 = open(filename, 'r')
    except:
        print("ERROR: Unable to read file. Line 254", filename)

    while True: # Read the file line by line.
        line = f1.readline()
        if not line: # break if we don't read anything
            break;
        data = line.split()
        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
        else:
            # Convert the data to stuff we can use
            _z1 = float(data[0])
            _x  = float(data[1])
            _z2 = float(data[2])
```

```
            _z1 = deg2rad(_z1)
            _x  = deg2rad(_x)
            _z2 = deg2rad(_z2)
            orientation_matrix = calcRotMat(_z1, _x,
               _z2)
            if not quiet:
                displayMat(orientation_matrix)
            if save:
                writeMat(orientation_matrix, _z1, _x,
                   _z2,'P', _axis)
# Input is a set of euler angles
elif "-e" in argv or "--euler-angles" in argv:
    try:
        try:
            index = argv.index("-e")
        except:
            index = argv.index("--euler-angles")
    except:
        print("ERROR: Unable to read Euler angles. Line
           285")
        exit()
    _z1 = float(argv[index + 1])
    _x =  float(argv[index + 2])
    _z2 = float(argv[index + 3])
    _z1 = deg2rad(_z1)
    _x  = deg2rad(_x)
    _z2 = deg2rad(_z2)

    orientation_matrix = calcRotMat(_z1, _x, _z2)

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

else:
    if len(argv) < 3:
```

```
        print("ERROR: Not enough command line arguments. Line 303")
        print("Input either an axis, and a misorientation, or a ZXZ Euler angle set with the option -e or --euler-angles.")
        displayHelp()
        exit()
    try:
        _axis = int(argv[1])
        _misorientation = float(argv[2])
    except:
        print('''
        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. Line 311
          ''')
        exit()

    if not len(str(_axis)) == 3: # axis length greater than 3
        print("ERROR: Argument 1 must by a 3 digit number like \'100\'.  Line 318")
        exit()

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

#
```

```
#———————————————————————The Actual Calculations
    ———————————————————#
#
    ————————————————————————————————————————————

    # First convert to a quaternion
    q[0] = axis2quat(axis, _misorientation / 2)
    q[1] = axis2quat(axis, -_misorientation / 2)

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

#
    ————————————————————————————————————————————

    # Using the Rodrigues Rotation Formula, defined as
        R = I + sin(theta) * K + (1 - cos(theta))*K^2
    # with K = [0 -k_z, k_y; k_z, 0, -k_x; -k_y, k_x,
        0], and the components of
    # k coming from the vector being rotated about.
        Theta is specified by the misorientation.
    if useRRF:
        orientation_matrix1, orientation_matrix2 =
            calcRotMatRRF(axis, _misorientation)

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

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

            if save:
                assert i < 2, "ERROR: Too many Euler
                    angles. Line 417"
```

```
                if i == 0:
                    writeMat(orientation_matrix1, _z1[i
                       ], _x[i], _z2[i], 'P', _axis)
                else:
                    writeMat(orientation_matrix2, _z1[i
                       ], _x[i], _z2[i], 'Q', _axis)
#
```
---
```
    else:
        for i in range(0,len(_z1)):
            orientation_matrix = calcRotMat(_z1[i], _x[
               i], _z2[i])

            if not quiet:
                displayMat(orientation_matrix)

            if dispEuler:
                displayAngles(_z1[i], _x[i], _z2[i])

            if save:
                assert i < 2, "ERROR: Too many Euler
                   angles. Line 374"
                if i == 0:
                    writeMat(orientation_matrix, _z1[i
                       ], _x[i], _z2[i], 'P', _axis)
                else:
                    writeMat(orientation_matrix, _z1[i
                       ], _x[i], _z2[i], 'Q', _axis)
```

# Appendix B

# genOrientationMatrix.sh Bash Script

This is a bash script used to read a file containing a CSV file containing misorientation angles, and use those angles to generate the P and Q matrices. This script calls the script orientation_matrix.py.

```
#! /bin/bash

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

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

```
# This takes the first argument from the command line -
    this is assumed to be a
# filename of the format 100Tilt.
FN=$1

echo "Determining the axis..."
# 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
# where it will find a match for 101, but as long as
   the files are named correctly
# it shouldn't be an issue.
AXIS=`echo $FN | grep -o "1[01][01]"`

echo "Reading the file..."
IFS="," # separation character is the comma
# Exit with error code 99 if unable to read the file
[ ! -f $FN ] && { echo "$FN file not found"; exit 99; }

# This makes the assumption that the file
   orientation_matrix.py has executable
# rights.
echo "Running the command: ~/projects/scripts/
   orientation_matrix.py $AXIS <angle> -s -q"
while read -r angle en; do # read the file with comma
   separated values

  ~/projects/scripts/orientation_matrix.py $AXIS $angle
      -s -q
done < "$FN" # the "$FN" is required if it's going to
   be run properly!

IFS=$OLDIFS # go back to the old separation character
   based on the system value.
```