Theses and Dissertations

2005-11-02

Methodology for Determining the Variance of the Taylor Factor: Application in Fe-3%Si

Craig P. Przybyla
Brigham Young University - Provo

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>

Part of the [Mechanical Engineering Commons](#)

BYU ScholarsArchive Citation

Przybyla, Craig P., "Methodology for Determining the Variance of the Taylor Factor: Application in Fe-3%Si" (2005). *Theses and Dissertations*. 687.
<https://scholarsarchive.byu.edu/etd/687>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

METHODOLOGY FOR DETERMINING THE VARIANCE OF
THE TAYLOR FACTOR: APPLICATION IN FE-3%SI

by

Craig P. Przybyla

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

December 2005

Copyright © 2005 Craig P. Przybyla

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Craig P. Przybyla

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

Brent L. Adams, Chair

Date

Brian D. Jensen

Date

Michael P. Miles

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Craig P. Przybyla in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Brent L. Adams
Chair, Graduate Committee

Accepted for the Department

Matthew R. Jones
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering
and Technology

ABSTRACT

METHODOLOGY FOR DETERMINING THE VARIANCE OF THE TAYLOR FACTOR: APPLICATION IN FE-3%SI

Craig P. Przybyla

Department of Mechanical Engineering

Master of Science

The variance of the Taylor factor due to variations in the microstructure and window size is explored using both a random sampling method and a previously developed statistical relationship. The results from the random sampling method correlated well with the statistical variance relationship when the magnitude of the variance was greater than that of the numerical errors observed in the statistical calculation. An empirical relation was developed to model the results and the constants for this relationship were determined for pseudo-three dimensional Fe-3%Si. Implementation of the statistical variance relationship in true 3D microstructures is not limited by material opacity, since it depends only upon the 2-point pair correlation functions. The connection between the variance of the R-value and variance of the Taylor factor is considered. Although only a weak connection was found, it was observed that relatively small variations in the Taylor factor yield large variances in the R-value.

ACKNOWLEDGMENTS

This work would not have been possible without the contributions of many different people and groups. The patient mentoring of my advisor Dr. Brent L. Adams is gratefully acknowledged along with the help of the members of my graduate committee, Dr. Brian D. Jensen and Dr. Mike Miles. The members of my research group have each in different ways been valuable resources in time and advise during this process including Jay Basinger, Dr. Bin Cai, Minu Gandhi, Xiang Gao, Dong Fang, Dr. David Fullwood, Eric Homer and Scott Lemmon. It goes without saying that without the support and dedication of my wife Laura and daughter Elizabeth, this work could not have been completed. The author would like to thank Dr. Robert Comstock of AK Steel for generously providing the material used in this work. This work was supported by the MRSEC program of the National Science Foundation under Award Number DMR-0079996.

Contents

Acknowledgments	vi
List of Tables	xi
List of Figures	xiv
1 Introduction	1
1.1 Engineering and Variation	1
1.2 Material Properties and Variance	1
1.3 Variance of the Taylor Factor and R Value	3
2 Theoretical	5
2.1 Definitions	5
2.1.1 Frames of Reference	5
2.2 Plasticity	6
2.2.1 Classical Taylor's Theory	6
2.2.2 Strain Rate	8
2.2.3 Calculating the Taylor Factor	9
2.2.4 Calculation of Effective Taylor Factor	12
2.2.5 Rate Sensitive vs. Rate Insensitive	13
2.3 Variance Model	13
2.4 Applications to the Normal Anisotropy R	15
3 Simulations and Experiments	17
3.1 Simulations	17
3.1.1 OIM Data Collection	17

3.1.2	Taylor Factor Calculations	18
3.1.3	Variance Simulations	20
3.2	R-value measurements	22
4	Results and Discussion	25
4.1	Effective Taylor Factor	25
4.2	Variance of the Taylor Factor	26
4.3	Measured R Values and Variance	29
5	Conclusions	33
5.1	Observations	33
5.2	Future Work	34
	Appendix	35
A	Rotation Tensor	37
B	Derivation of Statistical Variance Relationship	39
C	Discretization of the Fundamental Zone	45
D	Results	49
E	C++ Taylor Variance Code	51
E.1	main.cpp	51
E.2	main.hpp	52
E.3	dotAng.hpp	53
E.4	dotAng.cpp	56
E.5	TPvarianc.hpp	86
E.6	TPvariance.cpp	88
E.7	tokenize.hpp	100
E.8	tokenize.cpp	100

F	Matlab Taylor Factor Code	103
F.1	TaylorFactor.m	103
F.2	GetSlipTensors.m	105
F.3	GetContractedStipTensors.m	105
F.4	StrainToStress.m	107
F.5	ReduceCreepTo5D.m	109
F.6	ChinMammelTaylorFactor.m	109
G	Matlab Taylor Factor Variance Random Sampling Code	111
G.1	RandVariance.m	111
	Bibliography	115

List of Tables

3.1	Dimensions of the defined windows Ω_1 and Ω_2 that are shown in Figure 3.4. (in millimeters)	23
4.1	Comparison of Effective Taylor Factors for Different Discretization Schemes at $\lambda = 0.4$	26
4.2	Constants determined for Fe-3%Si for variance relationship in Equation (4.1).	27
4.3	Measured and theoretically estimated R values and contractile strain ratios λ with their corresponding standard deviations for Ω_1 sized 4.7752 x 9.5504 mm.	29
4.4	Measured and theoretically estimated R values and contractile strain ratios λ with their corresponding standard deviations for the Ω_2 sized 9.5504 x 9.5504 mm.	29
C.1	Boundaries of tesserae in $\Phi - \phi_2$ plane in radians where Fundamental Zone has been discretized into 512 tesserae and where $k = 0 \dots K$	47
D.1	Standard deviation of Taylor Factors estimated for Fe-3%Si using random sampling method and the statistical variance relationship with 512 discretizations for $\Omega = 1.0 \times 1.0$ mm.	49
D.2	Standard deviation of Taylor Factors estimated for Fe-3%Si using random sampling method and the statistical variance relationship with 512 discretizations for $\Omega = 2.2 \times 2.2$ mm.	50

List of Figures

1.1	This is picture of a microelectromechanical switch that was taken at BYU. The switch is in the unengaged position on the left and the engaged position on the right.	2
2.1	Coordinate system defined relative to a tensile sample with width w and thickness t	16
3.1	A OIM grain map for one of the four prepared samples. Continuous colors represent individual grains while different color represent different grains.	18
3.2	Pole figures created from the data collected using OIM for Fe-3%Si showing the texture for three different crystallographic directions with the intensity of those directions indicated.	19
3.3	The space of all possible unique orientations when cubic symmetry is applied. This space is bounded by $0 \leq \phi_1 \leq 2\pi$, $\arctan(1/\cos \phi_2) \leq \Phi \leq \pi/2$ and $0 \leq \phi_2 \leq \pi/4$	21
3.4	The tensile samples cut from Fe-3%Si with the circle grid etched onto the surface. The base and height of the rectangles defined by Ω_1 and Ω_2 were determined by taking the average of the measurements taken at each indicated corner.	23
3.5	Tensile samples cut from Fe-3%Si before and after deformation. The grid used to measure the strain before and after deformation is seen on the fractured sample.	24
4.1	Comparison of the effective Taylor factors calculated using the continuous dataset and tessellation schemes with 512, 1728 and 4096 tesserae. 25	

4.2	Standard deviation of the effective Taylor factor over Ω of size 2.3876 x 2.3876 mm.	31
4.3	Standard deviation of the effective Taylor factor over Ω_1 of size 4.7752 x 9.5504 mm.	31
4.4	Standard deviation of the effective Taylor factor over Ω_2 of size 9.5504 x 9.5504 mm.	32
C.1	Definition of invariant measure element B_n in $SO(3)/G$ in terms of the Euler angles.	46
C.2	Boundaries of tesserae in $\Phi - \phi_2$ plane where Fundamental Zone has been discretized into 512 tesserae and where $k = 0 \dots K$	47

Chapter 1

Introduction

1.1 Engineering and Variation

Dealing with variation in material properties and performance has always been part of modern engineering practice. Much of the variation is directly due to variation in the microstructure and nanostructure of materials due to dislocations, inclusions, different phases, etc. On larger scales the effects of microstructure and nanostructure variance is negligible because it is spread out over large areas. Increasingly in the last several years, however, engineers are designing parts on smaller and smaller length scales. Microelectromechanical systems or MEMS, for example, are tiny switches (see Fig. 1.1) that are measured on the scale of microns and can often only be seen with a high powered microscope. In the overall geometry of the MEMS shown, the part is several hundred microns across in width, yet its arms have a cross section of only a few microns by a few microns. Parts with such small dimensions are very susceptible to variation in the microstructure. In order to meet the demands of current technology, such as those posed by MEMS, engineers must gain a better understanding of property variance in materials based on the micro and nanostructure characteristics that affect it.

1.2 Material Properties and Variance

Many efforts have been made to link the properties observed on a macroscopic level in polycrystalline materials to the makeup of that material on a microscopic

Figure 1.1: This is picture of a microelectromechanical switch that was taken at BYU. The switch is in the unengaged position on the left and the engaged position on the right.

level. Early efforts used the Orientation Distribution Function (ODF) to form estimates of macroscopic property values based on weighting the properties of the individual crystals in the polycrystalline material according to the volume averages of different orientations of those crystals (Bunge 1993; Bunge 1982). Previously the ODF was obtained using pole figure inversion. More rapid measurement, however, of lattice orientation on a large scale in polycrystalline materials has been possible since the advent of Orientation Imaging Microscopy (OIM) (Adams, Wright, and Kunze 1993; Adams 1993). OIM automates the process of collecting electron backscatter diffraction patterns in a scanning electron microscope and indexing them in order to determine the orientation of the crystalline lattices. Methods via OIM enable rapid and accurate estimates of material ODFs.

Using volume fraction data or one-point statistics has enabled the reliable prediction of various property values based on the microstructure of the material. With a desire to predict property values even more accurately than possible using only one-point statistics, efforts have been made which incorporate two-point statistical microstructure information which considers both volume fraction and spatial characteristics of microstructure. Mason, in particular, demonstrated that by incorporating two-point statistics in the prediction of the elastic constants in copper the bounds were narrowed by a factor of 3 to 5 (Mason and Adams 1999).

Efforts such as these that predict material properties based on the morphology of the microstructure have greatly increased the understanding of how microstructures

affect material properties and material performance. In addition, these methods are building an important link between the makeup of the microstructure on a microscopic scale to the actual performance of the material on the macroscopic level. In general, however, efforts to link macroscopic material properties to the microstructure in polycrystalline materials have done little to account for the variance in the material properties due to the makeup of the microstructure.

Although variance has been researched on several different platforms, it has only narrowly been researched in relation to the microstructure of polycrystalline materials. Early work was done which showed how the grain size affected plastic anisotropy (Welch and Bunge 1986). Others like Skrotzki, for example, looked at how texture and precipitates affect the anisotropy of certain alloys (Skrotzki, Starke, and Shiflet 1994). More recently however efforts have been made to look at the variance of the ODF depending on a given window with specified dimensions (Lee, Piehler, Rollett, and Adams 2002). Work has also been done to investigate how variance in the microstructure affects variability in the grain boundary character distribution (Adams, Homer, and Basinger 2003). In addition variance dependant on the microstructure in two-phase composites has been examined for heterogeneous and random microstructures (Lu and Torquato 1990; Quintanilla and Torquato 1998).

1.3 Variance of the Taylor Factor and R Value

This work explores the variance of a plastic property commonly called the Taylor factor. In essence the Taylor Factor is simply a measure of the plastic work required to deform a polycrystalline material and is based on simple work relationships. It has been the goal of the authors to estimate only the variance of the Taylor factor that is due to the morphology and texture of the microstructure in polycrystalline materials. Studies have shown that variations in the Taylor factor from one grain orientation to the next can help predict failure in materials. Wright, for example, showed that grains with low Taylor factors next to grains with relatively high Taylor factors are more susceptible to large stress concentrations (Wright and Field 1998).

By quantifying the variance of the Taylor factor on the scale of only a few grains such relationships can be better understood.

Calculation of the variance of the Taylor factor is pursued in a couple of ways, one of which utilizes a variance relation developed previously (see (Przybyla, Adams, and Miles 2004)). This variance relationship in particular incorporates both volume fraction data of orientation distributions and spatial characteristics of one orientation relative to another, thus linking properties variance to microstructure. In addition, this variance relationship is dependent on a specified window of microstructure, thus limiting the variance to a given volume.

Another important plastic property that is directly linked to the Taylor factor is the normal anisotropy parameter or R-value. The R-value is a ratio of width to thickness strains in sheet metal, and is an important measure of sheet metal formability. In particular, the R-value predicts thinning during sheet forming processes. The R-value is an important property to consider in a variance study, because historically scatter in the measurement of the R-value has been a problem. For example, in one study conducted by the International Deep Drawing Research Group (I.D.D.R.G.) in England the R-values measured on identical samples in several different laboratories varied from 1.22 to 1.60 with a 15% scatter (J. L. Duncan and Johnson 1967). Another study sought to quantify sources of variation in R-value measurements but did not consider variation due to microstructure (Liu 1983). Thus in order to better understand the high variation of R-value measurements, variance of the R-value will be considered relative to the variance of the Taylor factor.

Chapter 2

Theoretical

2.1 Definitions

Throughout this work traditional tensor notation will be used. All two-dimensional tensors will be defined by lower case letters using either indices or bold case. In addition all four dimensional tensors will be defined as upper case letters using either indices or bold case. Einsteinian summation convention is assumed in all equations.

2.1.1 Frames of Reference

Two frames of reference will be used, the sample frame defined by the principal directions of the sample and the crystal frame which is defined by the principal directions of the crystalline lattice. As calculations will be performed in both frames of reference, the tensors in the crystal frame will be indicated by the superscript c .

In order to relate the crystal frame to the sample or global frame a tensor is defined by its components according to

$$g_{ij} = \cos(\hat{\mathbf{e}}_i^c, \hat{\mathbf{e}}_j) = \hat{\mathbf{e}}_i^c \cdot \hat{\mathbf{e}}_j \quad (2.1)$$

where $\hat{\mathbf{e}}_i^c$ and $\hat{\mathbf{e}}_j$ are the axes of the coordinate system in the crystal frame and the sample frame respectively. The tensor defined by (2.1) are the direction cosines of the angles between each of the corresponding axis in the two frames of reference. The crystal frame will be related to the sample frame according to the Euler angles ϕ_1 , Φ and ϕ_2 . The rotation matrix g is defined according to its components in Appendix A.

2.2 Plasticity

2.2.1 Classical Taylor's Theory

Modern theory of plasticity originated from the work of Taylor who postulated that the only slip systems active during plastic deformation are those that minimize the internal work (Taylor 1938). The rate of internal work is typically defined as the sum of the products of the critical resolved shear stress $\tau_c^{(k)}$ and the shear-rate $\dot{\gamma}^{(k)}$ over all active slip systems k according to

$$\dot{W}_i^P = \sum_{k=1} \tau_c^{(k)} \dot{\gamma}^{(k)} \quad (2.2)$$

Later work by Bishop and Hill developed the concept of stress corners or points where stress vectors touch the intersection of two or more slip hyperplanes. They then physically reasoned that the set of active stress corners out of the set of all possible stress corners would be the set that maximized the external plastic work (Bishop and Hill 1951a; Bishop and Hill 1951b). The rate of external plastic work is calculated as the product of the stress and strain according to

$$\dot{W}_e^P = \sigma_{ij} \dot{\epsilon}_{ij} \quad (2.3)$$

where $\boldsymbol{\sigma}$ and $\dot{\boldsymbol{\epsilon}}$ are the stress state and strain rate respectively.

Work by Chin and Mammel showed that these two approaches were equivalent by applying the principle of virtual work which requires the internal work to be the same as the external work (Chin and Mammel 1969). Combining the Taylor and Bishop-Hill approaches

$$\dot{W}^P = \sigma_{ij} \dot{\epsilon}_{ij} = \sum_{k=1} \tau_c^{(k)} \dot{\gamma}^{(k)}. \quad (2.4)$$

If the critical resolved shear stress is taken to be constant and uniaxial strain in the $\hat{\mathbf{e}}_1$ direction is applied, equation (2.4) can be rewritten as

$$\dot{W}^P = \tau_c \sum_k \dot{\gamma}^{(k)} = \tau_c \Gamma = \sigma_{11} \dot{\epsilon}_{11} \quad (2.5)$$

where

$$\Gamma = \sum_k \dot{\gamma}^{(k)}. \quad (2.6)$$

The Taylor factor for uniaxial strain can then be defined as

$$M = \frac{\Gamma}{\dot{\epsilon}_{11}} = \frac{\sigma_{11}}{\tau_c}. \quad (2.7)$$

In words, the Taylor factor can be seen as a measure of required plastic work. Grains with a given lattice orientation and subjected to strain rates that result in large values of M require more plastic work than orientations with lower values of M when compared over the same range of slip. It can be seen that the Taylor factor is a function of the lattice orientation according to (2.7) which can be expressed as $M(g)$ where g denotes the lattice orientation according to the definition given previously.

In order to move from a single crystal to the sample or macroscopic Taylor factor, the Taylor factor must be integrated over all possible physically distinctive crystal orientations or $SO(3)/G$ which can be expressed as

$$\overline{M} = \iiint_{SO(3)/G} M(g) f(g) dg \quad (2.8)$$

where $f(g)$ is the orientation distribution function (ODF) (see (Bunge 1993)). The set of physically distinctive crystal orientations $SO(3)/G$ is the fundamental set of all possible uniquely defined orientations with G as its point symmetry subgroup (Morawiec 2003).

2.2.2 Strain Rate

The effective strain rate under uniaxial loading is defined according to

$$\dot{\epsilon} = \begin{bmatrix} \dot{\epsilon}_{11} & 0 & 0 \\ 0 & \dot{\epsilon}_{22} & 0 \\ 0 & 0 & -(\dot{\epsilon}_{11} + \dot{\epsilon}_{22}) \end{bmatrix} \quad (2.9)$$

when it is assumed that the material is plastically incompressible or that

$$\dot{\epsilon}_{ii} = 0. \quad (2.10)$$

Using the formulation of strain rate as in Equation (2.9) it is assumed that no shear strain occurs. It is recognized that this is only usually valid if the microstructure exhibits orthorhombic sample symmetry, and when the load is applied parallel to one of its principal direction of orthorhombicity.

It is convenient to define the strain state according to

$$\dot{\epsilon} = \eta \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\lambda & 0 \\ 0 & 0 & -(1 - \lambda) \end{bmatrix}. \quad (2.11)$$

where η is defined as

$$\eta = \dot{\epsilon}_{11} \quad (2.12)$$

and λ is defined such that

$$\lambda = -\frac{\dot{\epsilon}_{22}}{\eta}. \quad (2.13)$$

In this way, the uniaxial strain state can be written as a function of the magnitude of the principal strain rate η and the parameter λ according to $\dot{\epsilon}(\eta, \lambda)$. The variable η can be loosely considered as the magnitude of the rate of uniaxial strain and the

contractile strain ratio or λ is related to the ratios of the strain rates in the $\hat{\mathbf{e}}_2$ and $\hat{\mathbf{e}}_3$ directions, and must be between 0 and 1.

Taylor also assumed that the macroscopic strain state can be imposed uniformly throughout the entire sample. Thus when the macroscopic strain state is defined the strain state for each orientation can also be calculated by using the rotation matrix defined in (2.1) according to

$$\dot{\epsilon}_{pq}^c = g_{pi}g_{qj}\dot{\epsilon}_{ij} \quad (2.14)$$

In uniaxial tensile tests the form of the strain state can be that of Equation (2.11) if it is assumed that the shear is negligible. In this case, the magnitude of principal strain rate η is known but the contractile strain ratio λ is not known unless it is determined experimentally.

2.2.3 Calculating the Taylor Factor

Three methods to calculate the Taylor factor will be presented for completeness but only the latter or the Hutchinson method will be used for this work. The first two methods are rate insensitive and each have multiple solutions thus being ambiguous while Hutchinson's approach is rate sensitive and only yields one solution.

Bishop-Hill Theory

As discussed previously, Bishop and Hill introduced stress corners or points where slip is predicted to occur when stress vectors touch the intersection of two or more slip hyperplanes (Bishop and Hill 1951a; Bishop and Hill 1951b). These stress corners have been determined for the $\{111\} < 110 >$ slip systems in face-centered cubic (fcc) materials when at least five slip systems are activated at the same time. As postulated by Bishop and Hill, the correct set of stress corners is the set that maximizes the external work in Equation (2.3) for a given strain rate $\dot{\epsilon}(\eta, \lambda)$. When this set is known the stress state is known. It is often the case, however, that more

than one set of stress corners will yield the same maximum external work. Thus it is not exactly known which set is the correct set.

Once a stress state is found the Taylor factor can be directly calculated according to Equation (2.7). It should be noted that the primary slip system in body-centered cubic (bcc) materials, typically the $\{110\} < 111 >$ slip system, yield identical results to that of the $\{111\} < 110 >$ slip systems in fcc materials in terms of the calculation of the Taylor factor. Thus the Bishop-Hill stress corners are valid for the dominant bcc slip systems as well.

Chin-Mammel Linear Optimization

Chin and Mammel followed Taylor's approach of determining the Taylor factor by selecting the set of slip systems that minimized internal work. They used a linear optimization technique to select the set of five slip planes according the Taylor's theory for several different slip systems such that the internal work was minimized according to Equation (2.2) (Chin and Mammel 1967). This work is particularly successful in finding the Taylor factor in the case where multiple slip systems are active but it is often the case that more than one set of slip systems are found that minimizes the internal work.

As indicated previously, both the Bishop-Hill method and Chin-Mammel optimization leaves somewhat of an ambiguity because more than one set of five stress corners will maximize the external work and more than one set of slip systems will minimize the internal work. Thus using either the Bishop-Hill approach or the Chin-Mammel optimization technique leaves doubt as to which set of stress corners or slips systems is actually the correct set.

Hutchinson Stress/Strain Relationship

According to the Taylor theory the strain rate is proscribed uniformly for the whole material leaving the stress state to be determined. Another approach that is less ambiguous uses a formulation by Hutchinson that allows the calculation of the stress state from a proscribed strain rate (Hutchinson 1976). As it can be seen,

using Hutchinson's rate sensitive formulation removes the ambiguity of Bishop-Hill or Chin-Mammel because it includes all possible slip systems in the calculation and results only in a unique solution.

Hutchinson defines the slip tensor which describes the various slip systems in a given crystalline lattice according to

$$\mu_{ij}^{(k)} = \frac{1}{2}(b_i^{(k)}n_j^{(k)} + b_j^{(k)}n_i^{(k)}) \quad (2.15)$$

where $\mathbf{n}^{(k)}$ and $\mathbf{b}^{(k)}$ are the unit normal to the slip plane and the slip direction in the k^{th} slip system respectively. Using this slip tensor, and with $\boldsymbol{\sigma}$ defined as the stress tensor of the system, the resolved shear stress of the k^{th} system is

$$\tau^{(k)} = \boldsymbol{\sigma} \boldsymbol{\mu}^{(k)} \quad (2.16)$$

The shear rate can then be calculated according to

$$\dot{\gamma}^{(k)} = \alpha \left| \frac{\tau^{(k)}}{\tau_o^{(k)}} \right|^{s-1} \left(\frac{\tau^{(k)}}{\tau_o^{(k)}} \right) \quad (2.17)$$

where $\tau_o^{(k)}$ is the reference stress for the k^{th} slip system, s is the strain rate sensitivity which is a material property and α is a scaling factor.

The strain rate can then be calculated by

$$\dot{\epsilon}_{ij} = \sum_k \dot{\gamma}^{(k)} \mu_{ij}^{(k)} = P_{ijmn} \sigma_{mn} \quad (2.18)$$

where

$$P_{ijmn} = \sum_k \left(\frac{\alpha}{\tau_o^{(k)}} \right) \left| \frac{\tau^{(k)}}{\tau_o^{(k)}} \right|^{s-1} \mu_{ij}^{(k)} \mu_{mn}^{(k)}. \quad (2.19)$$

Hutchinson calls $\mathbf{P}$ the tensor of creep compliance. In Equation (2.18) the relationship between stress and strain is explicitly defined. If the stress is known then the strain can be solved directly. However if Taylor theory is applied the state of strain is

proscribed and the stress must be calculated. Hutchinson describes an iterative method that can be used to do this.

The strain when $s = 1$ can be directly determined. The strain rate sensitivity s is then incremented by Δs (typically on the order of $1/10^{th}$) and the stress is updated according to

$$\Delta\sigma = \frac{1}{s} (P^{-1}\dot{\epsilon} - \sigma). \quad (2.20)$$

This calculation in Equation (2.20) is repeated for a given s until $\Delta\sigma$ approaches zero. Once Equation (2.20) has converged, s is again incremented by Δs . This process is repeated until the desired strain rate sensitivity s is reached.

Using this method and the definition of strain rate as in Equation (2.11), the Taylor factor and effective Taylor factors can both be expressed as a function of λ according to $M(g, \lambda)$ and $\overline{M}(\lambda)$ respectively.

In order to simplify the calculation the strain rate $\dot{\epsilon}$ is defined in the sample frame and then rotated into the crystal frame according to (2.14). Using Hutchinson's iterative method, the creep compliance is calculated in the crystal frame which in turn yields the correct stress state in the crystal frame. The stress tensor is then rotated in the sample frame according to

$$\sigma_{ij} = g_{pi}g_{qj}\sigma_{pq}^c \quad (2.21)$$

after which the Taylor factor is then calculated. In this manner the slip tensors do not have to be rotated from the crystal frame in which they are defined and the overall number of calculations is less than if the problem was completely worked in the sample frame.

2.2.4 Calculation of Effective Taylor Factor

In the present case of a uniaxial strain rate, the magnitude η from Equation (2.12) is known but the contractile strain ratio λ from Equation (2.13) is not. Thus not only must the stress state be calculated using the Hutchinson method but the

correct strain rate state must also be determined. The correct λ is the one which minimizes the internal work or maximizes the external work depending on how the calculation is being done. This is determined by iterating through all possible values of λ from $\lambda = 0$ to $\lambda = 1$ with some step size $\Delta\lambda$.

Using the Hutchinson method for determining the stress from the proscribed strain rate requires the internal work calculation; thus, in this case the correct λ is the one for which the internal work is minimized. For each λ the stress at each orientation g is determined using the Hutchinson method. Once $\overline{M}(g, \lambda)$ is known for each λ the minimum $\overline{M}(g, \lambda)$ is the correct estimate of the effective Taylor factor for the material and the corresponding λ is the correct estimate of the effective contractile strain ratio.

2.2.5 Rate Sensitive vs. Rate Insensitive

The plastic work calculated using the rate sensitive Hutchinson method converges to the same value as the plastic work calculated using a rate insensitive model as the rate sensitivity s tends to infinity. Thus it is noted that the rate sensitive reference shear stress τ_o becomes the rate insensitive critical resolved shear stress τ_c as $s \rightarrow \infty$.

2.3 Variance Model

A relationship between microstructure and macroscopic property variance has been developed (Przybyla, Adams, and Miles 2004). This relationship applied to the Taylor factor as defined in Equation (2.7) defines the variance of effective Taylor factor $\bar{M}$ in a region Ω to be

$$\sigma_{\bar{M}(\Omega)}^2 = \frac{1}{\nu(\Omega)^2} \iiint_{SO(3)/G} \iiint_{SO(3)/G'} \iiint_{\mathbb{R}^3} f_2(g, g' | \vec{r}) M(g) M(g') \Theta(\vec{r} | \Omega) d\vec{r} dg dg' - \bar{M}(\Omega)^2 \quad (2.22)$$

In Equation (2.22) the variance $\sigma_{\bar{M}(\Omega)}^2$ is shown to depend on the two-point pair correlation functions (PCFs) $f_2(g, g' | \vec{r})$ of the microstructure. This two-point PCF is defined to be the probability density of finding an arbitrary vector $\vec{r}$ whose

head is at orientation g and tail at orientation g in the microstructure. In this way, the two-point PCF contains both volume fraction information and information about how the orientations in the material are spatially located relative one to one another. Work done by Gao, et. al. has in detail explored the two-point PCF in polycrystalline stainless steel (X. Gao 2005).

In addition the variance also depends on $\Theta(\vec{r}|\Omega)$ which is simply defined as the volume of overlap between two equally sized and shaped prisms whose centers are separated by $\vec{r}$. This function in essence incorporates the dependence of this variance relationship on a specified window of microstructure Ω . Approaching the variance problem as a material design optimization for a given application, the window size is on the order of the smallest dimensions in the design for the part under consideration. In the case where Ω is defined in terms of a rectangular prism with length, width and height defined by l , w and h respectively, this volume of overlap is defined explicitly as

$$\Theta(\vec{r}|\Omega) = (l - r_x)(w - r_y)(h - r_z) \quad (2.23)$$

where r_x , r_y and r_z are the x , y and z components of $\vec{r}$ respectively. For overlap to exist

$$\begin{aligned} r_x &< l \\ r_y &< w \\ r_z &< h \end{aligned} \quad (2.24)$$

must also be true. This volume of overlap along with the two-point probability functions has been developed in detail previously (Torquato 2002). The integration over $\Re^3$ is an integration over real space and $\nu(\Omega)$ is the volume of Ω . The complete derivation of this variance relationship is included in Appendix B. This variance relation directly relates microstructure statistics contained in the PCF to the variance of the Taylor factor M in the material.

2.4 Applications to the Normal Anisotropy R

The normal anisotropy parameter, or R-value, as originally introduced by Lankford is defined as

$$R = \frac{\epsilon_{22}}{\epsilon_{33}}. \quad (2.25)$$

or as a ratio of ϵ_w and ϵ_t in a sheet (W. T. Lankford and Bauscher 1950). The width and thickness strains of a tensile specimen become ϵ_{22} and ϵ_{33} as in Equation (2.25) according to the defined coordinate system in Figure 2.1. Because it is typically difficult to measure strain across the thickness of thin sheets the R-value can also be estimated using

$$R = \frac{\epsilon_{22}}{-(\epsilon_{11} + \epsilon_{22})} \quad (2.26)$$

when it is assumed that volume is conserved according to

$$\epsilon_{11} + \epsilon_{22} + \epsilon_{33} = 0. \quad (2.27)$$

Measuring the R-value using the through thickness strain (as in Equation (2.25)) has been termed the “direct method” in the literature while determination of the R-value using the volume conservation relationship in Equation (2.26) is commonly employed in industry and is called the “indirect method.”

The R-value is directly linked to the Taylor factor by the previously introduced contractile strain ratio λ as defined in Equation (2.13) according to

$$R = \frac{\lambda}{(1 - \lambda)}. \quad (2.28)$$

It is desired to determine the variance of the R-value relative to the variance of the Taylor factor. Using relation (2.22) the variance of the Taylor factor can be directly determined from the microstructure for a given window Ω . The estimated variance of the R-value will then be determined by using the variance obtained for

Figure 2.1: Coordinate system defined relative to a tensile sample with width w and thickness t .

the Taylor factor which can be directly related to the R-value by the contractile strain ratio λ according to Equation (2.28).

It is desired to determine the variance of the R-value relative to the variance of the Taylor factor. Although the microstructure dependent variance of the Taylor factor can be directly obtained through relation (2.22), in doing this the contractile strain ratio λ is fixed. Despite the connection between the contractile strain ratio λ and the R-value according to relation (2.28), a viable method has not been developed to formulate the problem in terms of the variance of the contractile strain ratio; however, coarse estimates of what the variance of the R-value might be at sufficiently large window sizes have been made and are described in detail in the results.

Chapter 3

Simulations and Experiments

3.1 Simulations

Silicon 3% Iron provided by AK Steel was used as the test material for all simulations and experimental measurements. For this work the material is rolled to a thickness of 0.3 mm and then annealed. The grain size distribution of the annealed material ranges between 0.5 and 5 mm. In particular this material was selected for its large grain size because it was anticipated that materials with grain sizes of the same order of magnitude as the volumes under consideration Ω (see Equation (2.22)) would have easily detectable variances. The large grain size relative to material thickness also enabled fairly accurate prediction of the properties of the material in 3D using only 2D scan data along the surface because the grains on the surface are continuous through the thickness of the material.

3.1.1 OIM Data Collection

In order to perform the simulations the orientations of the microstructure were measured using orientation imaging microscopy (OIM). The electron backscatter patterns were collected with a Phillips SFEG-XL30 scanning electron microscope utilizing automatic stage control. Four samples of Fe-3%Si were cut from the sheet stock approximately 25.4 x 25.4 mm. These samples were then mounted in thermoplastic and polished using traditional metallurgical sample preparation procedures to a $0.05e^{-3}$ mm finish. One scan was taken on each sample with an area of approximately 20 x 20 mm and with a scan step-size of 0.2 mm. The average confidence index

Figure 3.1: A OIM grain map for one of the four prepared samples. Continuous colors represent individual grains while different color represent different grains.

of all data points taken was 0.4 on a scale of 0 to 1. Typically acceptable confidence indexes are anything above 0.1. A grain map created from one of the OIM scans depicting the grain size and grain morphology is shown in Figure (3.1). In addition the texture of all of the combined data is represented in Figure 3.2.

It is noted that the material exhibits a very uniform Goss texture with little variation. As indicated in the figure the Goss texture was exhibited at over 20 times random.

3.1.2 Taylor Factor Calculations

A uniaxial strain state was assumed and the stress state for each orientation was then calculated using Hutchinson's iterative method that was summarized previously. The Taylor factors at each orientation were then estimated using the internal plastic work formulation over the range of $0 < \lambda < 1$ with an incrementation of 0.05; thus, in all λ was incremented into a set of 19 discrete values. It was concluded that the correct Taylor factor over the range of possible λ 's was at the λ that minimized the internal work.

Figure 3.2: Pole figures created from the data collected using OIM for Fe-3%Si showing the texture for three different crystallographic directions with the intensity of those directions indicated.

Since the Fe-3%Si is a single phase material with a body-centered cubic (bcc) lattice structure there are typically three slip system that can be considered, namely the $\{110\}$, $\{123\}$ and the $\{112\}$ with slip always along the closed packed planes in the $\langle 111 \rangle$ direction. In this work only the primary slip system or $\{110\}$ was considered which is an assumption commonly used for bcc materials because of the large dominance of the $\{110\}$ over the other two. The strain rate sensitivity s from Equation (2.19) for low carbon steels is typically between 65 and 100 (Hosford 2005). In this work s was set to 65 for all Taylor factor calculations because it was observed that for strain rate sensitivities between 65 and 100 the resulting Taylor factors only changed by 0.1%.

Calculation of the Taylor factor using the Hutchinson stress/strain relationship was validated against the methods used by Chin and Mammel in their work (Chin and Mammel 1967). In all cases the rate insensitive methods used by Chin and Mammel to calculate the Taylor factor yielded the same results as the rate sensitive

Hutchinson iterative method as $s \rightarrow \infty$. Hutchinson’s iterative method was used to determine the stress state from a proscribed strain state and then the Taylor factor was subsequently determined using the internal work formulation shown in Equation (2.7). This comparison was primarily used to check the validity of methods used by the author and debug the codes employed for the calculations in this work.

3.1.3 Variance Simulations

Random Sampled Variance

Variance of the Taylor factor was estimated using two different simulations. Firstly the variance of the Taylor factor was calculated by simply sampling a finite set of regions sized according to Ω directly from the datasets obtained via OIM. Again because the grains of the material are continuous through the thickness, Ω could be considered only as a 2D region along the same surface that the data was collected. For this work Ω is defined by a rectangle of base b and height h .

To calculate the variance of the Taylor factor using direct random sampling, the Taylor factor $M(g)$ had to be determined for each orientation in the dataset. This was done utilizing the iterative Hutchinson stress/strain relationship summarized previously. The region Ω was then placed randomly in one of the four scans from the dataset and the average Taylor Factor was then determined across all of the data points that fell within Ω . The standard deviation of the means of the Taylor factor for a specified Ω was then simply evaluated over the finite set of samplings using traditional statistical definitions. In this work, the direct calculation of the variance of the Taylor factor was determined using a sample size of 500 for each volume Ω that was considered.

Statistical Variance Relationship

The variance relationship previously defined in Equation (2.22) was also explored. In order to do these calculations a program was written in C++ using the numerical methods summarized here. The integrations over the fundamental zone

Figure 3.3: The space of all possible unique orientations when cubic symmetry is applied. This space is bounded by $0 \leq \phi_1 \leq 2\pi$, $\arctan(1/\cos \phi_2) \leq \Phi \leq \pi/2$ and $0 \leq \phi_2 \leq \pi/4$.

were estimated using discrete summations by discretizing the fundamental zone into a finite number of equal invariant measures according to

$$\nu(B_n) = \frac{1}{N} \iiint_{SO(3)/G} \sin \Phi d\phi_1 d\Phi d\phi_2 \quad \text{for } 1 \leq n \leq N \quad (3.1)$$

where B_n indicates each of N tesserae in the tessellation of the fundamental zone $SO(3)/G$ represented in Figure 3.3. A more detailed description is included in the Appendix C. As discussed in the results several different tessellation schemes were explored in order to ensure the best numerical accuracy.

The two-point PCF was measured directly from the experimentally measured OIM data by employing a discrete set of vectors $\vec{r}$ which sampled all possible pairs of data points spatially. The directions of the discrete set of vectors $\vec{r}$ were defined according to the principal axis of the scans obtained using OIM and the lengths of these vectors ranged from the distance between two scan points in a given direction to the smallest dimension of the smallest scan performed using OIM. In this work, all OIM scans were performed using a hexagonal grid with six principal axes and the smallest scan was on the order to 90 times the scan step size of 0.2 mm. Each scan point was assigned a bin number depending on where each orientation fell in the

tessellation scheme. Variance of the Taylor factor using a tessellation scheme of $SO(3)/G$ were compared using 512, 1728 and 4096 tesserae.

The integrations over the real space $\mathbb{R}^3$ were also performed discretely. Because the grains in the material used for this work extend across the entire thickness of the material it was assumed that accurate 3D information could be obtained using 2D planes. In addition because it was necessary to integrate over the discrete set of vectors $\vec{r}$ which lie in real space, the real space was actually defined as $R(\Omega)$ which represented the region of all possible vectors $\vec{r}$ in Ω . Both positive and negative vectors were considered even though their equivalence is recognized in the case that

$$f_2(g, g' | \vec{r}) = f_2(g', g | -\vec{r}). \quad (3.2)$$

Thus the area of $R(\Omega)$ is actually 4Ω if Ω is assumed to be rectangular. The tessellation of $R(\Omega)$ was performed using square regions with the same dimensions as the step size of the scans taken using OIM or 0.2 mm. Limiting the size of the tesserae to the step size of the scan insured that the information from the two-point PCF was not lost in the interpolation of the two-point PCF across $R(\Omega)$.

3.2 R-value measurements

Many different methods have been proposed to measure the R-value in order to remove the variation that is so common with its measurement. Measurements of the R-value using the 'direct method' are typically obtained using tensile samples (see Fig. 2.1). The R-value is directly calculated using the definition in Equation (2.25).

For this work it was desired to measure the R-value across limited volumes only instead of across the whole sample. In this way, the variance of the R-value determined from a set of measurements could be considered for a given volume Ω , as was done for the variance of the Taylor factor. In order to limit the measurement of the R-value to specific volumes, traditional circle grid analysis was employed. With circle grid analysis, grids are etched onto each tensile sample and the strains in the $\hat{e}_1$ and $\hat{e}_2$ directions (as defined in Figure 2.1) are calculated from the changes in lengths

measured from the predeformed grids to the deformed grids. The third component of strain in the $\hat{e}_3$ direction is deduced from the other two components by assuming that volume is conserved during deformation following the 'indirect method' for determining the R-value. Once the strains are known the R-value can be determined directly according to Equation (2.26).

The circle grids that were used to specify the volumes over which the R-value was measured can be seen in Figure 3.4. All physical measurements were taken using an optical microscope mounted on top of an electronic stage that measures to the closest thousandth of a millimeter. For the experimental measurements two volumes were considered, Ω_1 and Ω_2 as defined according to their predeformed dimensions in the Figure 3.4 and Table 3.1. After deformation, the dimensions were measured for the volumes Ω_1 and Ω_2 on each of 16 samples. Spatially the volumes that were measured were randomly chosen on each sample. The tensile samples before and after fracture can be seen in Figure (3.5). Each sample was pulled to fracture using an Instron tensile testing machine with a 5kN load cell and at a speed of 2.54 mm/min.

Figure 3.4: The tensile samples cut from Fe-3%Si with the circle grid etched onto the surface. The base and height of the rectangles defined by Ω_1 and Ω_2 were determined by taking the average of the measurements taken at each indicated corner.

Table 3.1: Dimensions of the defined windows Ω_1 and Ω_2 that are shown in Figure 3.4. (in millimeters)

	Ω_1	Ω_2
Base	4.7752	9.5504
Height	9.5504	9.5504

Figure 3.5: Tensile samples cut from Fe-3%Si before and after deformation. The grid used to measure the strain before and after deformation is seen on the fractured sample.

Chapter 4

Results and Discussion

4.1 Effective Taylor Factor

The effective Taylor Factor was estimated according to Equation (2.8) for $0 < \lambda < 1$ with an incrementation of 0.05. This calculation was performed using the continuous dataset obtained via OIM and by binning the data points from the scans into various orientation tesserae according to the tessellation schemes previously described. The Effective Taylor Factors are plotted versus the contractile strain ratio λ for the continuous dataset and for the tessellation schemes with 512, 1728, and 4096 tesserae in Figure 4.1.

Figure 4.1: Comparison of the effective Taylor factors calculated using the continuous dataset and tessellation schemes with 512, 1728 and 4096 tesserae.

When the entire dataset was considered (the continuous case), the internal work and the corresponding effective Taylor factor were minimum at approximately $\lambda = 0.4$ as can easily be seen in Figure 4.1. Thus it was concluded that the effective Taylor factor for this material is that which is at $\lambda = 0.4$. In addition even the smallest tessellation scheme of 512 approximated the continuous dataset reasonably with a difference less than 1%. The values of the effective Taylor factor for the continuous dataset and for all of the tessellation schemes at $\lambda = 0.4$ are listed in Table 4.1.

Table 4.1: Comparison of Effective Taylor Factors for Different Discretization Schemes at $\lambda = 0.4$.

Tessellation	Taylor Factor
Continuous	2.4118
512	2.4206
1728	2.4136
4096	2.4135

4.2 Variance of the Taylor Factor

In order to keep the results uniform all variances were considered as standard deviations, which is simply the square root of the variance according to traditional statistical definitions. The standard deviation of the Taylor factor estimated using both the random variance method and the statistical variance relationship using 512 tesserae for two different volumes Ω are listed in Tables D.1 and D.2 which are in Appendix D.

In Table D.1 the results obtained from the random window sampling and the statistical variance relationship agree reasonably well. Although this agreement persists when larger volumes were considered the numerical accuracy of the statistical relationship appeared to be limited. For larger sizes of Ω such as those listed in Table D.2 the statistical relationship is not able to detect smaller standard deviations and in reality resulted in small negative numbers. As larger and larger volumes were

considered these “negative” variances appeared to actually converge to some negative number much less than zero. It was assumed that these were not actually negative variances but that the numbers became negative when the variances became too small to detect due to numerical error. In the tables such “negative” numbers are listed as zeros.

An empirical relationship was developed to model the results of the variance of the Taylor factor. This relationship has the form of

$$\sigma_{\overline{M}(\lambda)}^2 = a_1 \left(\frac{\nu_g}{\nu(\Omega)} \right)^{a_2} [|\lambda_{eff} - \lambda|^{a_3} + a_4] \quad (4.1)$$

where a_1 through a_4 are experimentally determined constants, λ_{eff} is the contractile strain ratio for the corresponding effective Taylor factor and ν_g is the average volume of the grains. For the material used in this work, the grains were assumed to be spherical with an average radius of 0.75mm. The window Ω was considered as a volume extending through the thickness. The constants that were determined for Fe-3%Si are listed in Table 4.2. Fitting this relationship to four different volumes (Ω defined by 0.6 x 0.6 x 0.3 mm, 0.8 x 0.8 x 0.3 mm, 1.0 x 1.0 x 1.0 mm, and 2.4 x 2.4 x 0.3 mm) resulted in an R^2 fit of 0.86.

Table 4.2: Constants determined for Fe-3%Si for variance relationship in Equation (4.1).

a_1	0.03205
a_2	0.30969
a_3	2.22208
a_4	0.04599

The standard deviation of the Taylor factor itself appeared to converge quickly to zero as more grains were included in the calculation. Significantly large standard deviations are seen when Ω is just under the average grain size, such as was the case in Table D.1. But when the size of the window approaches that of the grain size and larger, such as in Table D.2, the magnitude of the standard deviation of the

Taylor factor quickly tends to zero, and is undiscernable using the statistical variance relationship.

It is also interesting to note that the variance of the Taylor factor is larger when the contractile strain ratio is near the extremes such as when $\lambda = 0$ or $\lambda = 1$ than it is near the effective Taylor factor when $\lambda = 0.4$. It is assumed simply that when the contractile strain ratio is at its extremes there are larger differences between the calculated Taylor factors from one orientation to the next than when the contractile ratio is near the correct value. This could simply be because much more plastic work is required to produce slip on lattices of certain orientations under strain states defined by extreme contractile strain ratios than other lattices of different orientations.

Texture inevitably plays an important role in the calculation of the standard deviations of the Taylor factor. As was mentioned previously, this material exhibits a strong Goss texture and thus little variation was actually observed in the Taylor factor from one grain to the next. It is assumed that a more heterogeneous material would exhibit larger variances even when considering window sizes that include more than just 1 or 2 grains.

Although the statistical variance relationship exhibits numerical inaccuracies when the variance is small, it has several strengths over the random variance method. This work has focused on a pseudo 3D realm where the material considered had grains extending through the entire thickness of the material. For such a material, the random sampling method gives quick accurate estimates of the variance; however, if a true 3D microstructure was considered this would not be the case. A random sampling method could not be performed unless 3D microstructure data was obtained using either time consuming serial sectioning or 3D sampling which can be performed only at this point using a synchrotron. The statistical variance relationship is only dependent on the two-point pair correlation function. It has been demonstrated in other work that this statistical information can be obtained relatively easily in 3D using oblique section planes around a fixed axis (X. Gao 2005). Using similar methods the statistical variance relationship could be applied to true 3D microstructures with only slightly more complexity.

4.3 Measured R Values and Variance

The R value was measured for two volumes (defined in Table 3.1) over randomly selected rectangles on each of 16 samples. The average R-values for Ω_1 and Ω_2 for all 16 measurements are listed in Tables 4.3 and 4.4 respectively. The theoretical R-value is also listed which was estimated from the theoretically determined contractile strain ratio, $\lambda = 0.4$, according to the relationship in Equation (2.28). In addition the contractile strain ratio was estimated from the measured R-values in the same manner using Equation (2.28) reversed.

Table 4.3: Measured and theoretically estimated R values and contractile strain ratios λ with their corresponding standard deviations for Ω_1 sized 4.7752 x 9.5504 mm.

Ω_1	Measured	Theoretical
R-Value	0.61	0.67
$\sigma_{\mathbf{R}}$	0.44	0.87
λ	0.38	0.40
σ_{λ}	0.41	0.47

Table 4.4: Measured and theoretically estimated R values and contractile strain ratios λ with their corresponding standard deviations for the Ω_2 sized 9.5504 x 9.5504 mm.

Ω_2	Measured	Theoretical
R-Value	0.42	0.67
$\sigma_{\mathbf{R}}$	0.33	0.74
λ	0.30	0.40
σ_{λ}	0.32	0.43

For this material the Taylor model appears to reasonably predict the R-value and corresponding contractile strain ratio λ . Due to the sensitivity of the R-value to measurement as cited in previous studies it is concluded that even with slight differences between the measured and theoretical estimates these results correlate

well. It is noted that other studies have shown that significant deviations in the R-value predicted using the Taylor model from measured results can occur (Choi and Barlat 1999).

The standard deviation from the measured R-values is listed as σ_R in Tables 4.3 and 4.4. The standard deviation of the measured contractile strain ratios was estimated from the measured standard deviation of the R-values again from Equation (2.28). For the theoretical values σ_R and σ_λ do not represent standard deviations in the traditional sense but are gross bounds between which it is expected that the R-values and corresponding contractile strain ratios must lie. These bounds were determined graphically from the plots of the standard deviations of the Taylor Factors (see Figures 4.2, 4.3 and 4.4).

The standard deviations for the Taylor factor over the windows defined by Ω_1 and Ω_2 are shown in Figures 4.3 and 4.4 respectively. It was assumed that because $M(\lambda)$ appears to be quadratic that bounds on the standard deviation of the contractile strain ratio could be made from the standard deviation of the effective Taylor factor. Such bounds are indicated in Figures 4.3 and 4.4. For example if the standard deviation of the Taylor factor is considered over a relatively small window such as that in Figure 4.2, it can be seen that the standard deviation of the Taylor Factor is large relative to the concavity of the parabola of Taylor factors over $0 < \lambda < 1$. Because the parabola is shallow but the standard deviations of the Taylor factors are large it is easy to conceive a minimum Taylor factor almost anywhere over the range $0 < \lambda < 1$. In contrast, however, in Figures 4.3 and 4.4 the standard deviation of the Taylor factor is narrow enough that it is possible to bound the region in which the minimum Taylor factor could fall. These bounds are indicated in Figures 4.3 and 4.4 by λ_l and λ_u which represent lower and upper bounds on λ respectively. Such bounds on the contractile strain ratio can theoretically yield bounds on the R-value. Although it is recognized that the bounds on R obtained from these graphical arguments are quite weak, the connection between the variance of the Taylor factor and the R-value is interesting to consider.

Figure 4.2: Standard deviation of the effective Taylor factor over Ω of size 2.3876 x 2.3876 mm.

Figure 4.3: Standard deviation of the effective Taylor factor over Ω_1 of size 4.7752 x 9.5504 mm.

Figure 4.4: Standard deviation of the effective Taylor factor over Ω_2 of size 9.5504 x 9.5504 mm.

Chapter 5

Conclusions

5.1 Observations

The goal of this work has been to quantify variance of the Taylor factor due to the variance in the microstructure of the material. An attempt has also been made to explore the connection between variance in the Taylor factor and the R-value, a important parameter which is one measure of sheet metal formability. From this study it has been observed that

- A theoretical relationship, relating the variance of the Taylor Factor of polycrystalline material, as a function of window size and the 2-point (pair) correlation functions, has been validated by comparison with calculations on many random samples of the microstructure in columnar Fe-3%Si. In a comparison of the statistical variance relationship to the random sampling method, the results correlate well when the magnitude of the variance is larger than the numerical errors of the statistical method.
- Crystallographic texture can have a significant influence on the variance of the Taylor factor. Highly textured materials are expected to have less variance than more heterogeneous materials of weaker texture.
- The statistical variance method would be a viable tool in quantifying microstructure variance in true 3D microstructures; whereas, the random sampling method would only apply if (rare) 3D microstructure datasets are available.

- Small variances in the Taylor factor suggest large variances in the R-value in the bounds estimates explored here. Gross bounds can be placed on the R-value based on the Taylor factor when the variance of the Taylor factor is relatively small, otherwise when the variance is considered for smaller volumes.
- In support of work using the Taylor model to predict the overall R value, on the basis of minimization of internal work, is shown to be quite accurate, in comparison with measurements using the circle grid analysis method.

5.2 Future Work

There have been many questions raised by this preliminary look into the variance of material properties based on microstructure.

- What would the variance of the Taylor factor be for a more heterogeneous material?
- How can Ω be varied in the statistical variance relationship in order to explore how variance is sensitive to the size and shape of Ω ? For example, one might assume that the variance would be less if Ω has its largest dimension in the same direction as the rolling direction than if its largest dimension was in the transverse or normal direction in a rolled material. How does changing the shape of Ω affect the variance relative to the basic shapes of the grains?
- How can this quantified microstructure based variance help design parts that are more robust?

APPENDIX

Appendix A

Rotation Tensor

Using Euler angles, $\mathbf{g}$ can be defined by its components according to

$$\begin{aligned}g_{11} &= \cos(\phi_1) \cos(\phi_2) - \sin(\phi_1) \sin(\phi_2) \cos(\Phi) \\g_{12} &= \sin(\phi_1) \cos(\phi_2) + \cos(\phi_1) \sin(\phi_2) \cos(\Phi) \\g_{13} &= \sin(\phi_2) \sin(\Phi) \\g_{21} &= -\cos(\phi_1) \sin(\phi_2) - \sin(\phi_1) \cos(\phi_2) \cos(\Phi) \\g_{22} &= -\sin(\phi_1) \sin(\phi_2) + \cos(\phi_1) \cos(\phi_2) \cos(\Phi) \\g_{23} &= \cos(\phi_2) \sin(\Phi) \\g_{31} &= \sin(\phi_1) \sin(\Phi) \\g_{32} &= -\cos(\phi_1) \sin(\Phi) \\g_{33} &= \cos(\Phi)\end{aligned}\tag{A.1}$$

where ϕ_1 , Φ and ϕ_2 are the Euler angles. This is the same rotation tensor that Bunge uses in his work (Bunge 1993).

Appendix B

Derivation of Statistical Variance Relationship

This derivation summarizes the previous work done by Przybyla, *et al* (Przybyla, Adams, and Miles 2004). Let $p^i(x)$ denote a property of interest at position x in the sample i of an ensemble of polycrystals. An ensemble in general can be seen as a set of samples that all have the same general type of microstructure. The microstructure of the sample is described by an arbitrarily large set of indicator functions, $I^i(h_j, x)$ defined such that

$$I^i(h_j, x) = \begin{cases} 1 & \text{if } N(x) \text{ associates with local state } h_j \\ 0 & \text{otherwise} \end{cases} \quad (\text{B.1})$$

Here $H(x)$ denotes the neighborhood of position x in the sample where the neighborhood can be seen as the region immediately around the position x . The local state can be seen as the set of information that describes the microstructure in the neighborhood of $N(x)$. the local state set h_j represents a subset for the complete set of all possible local states, H . It can be defined such that

$$h_j \cap h_{j'} = \emptyset (j \neq j'), \cup_j h_j = H, \quad (\text{B.2})$$

where the union is over all local state sets. It is assumed that local property $p^i(x)$ is known if local state $h(x)$ is known. Thus, to a good approximation,

$$p^i(x) = \sum_j I^i(h_j, x)p(h_j), \quad (\text{B.3})$$

where $p(h_j)$ is defined to be

$$p(h_j) = \frac{\iiint_{h_j \subset H} p(h) dh}{\iiint_{h_j} dh} \quad (\text{B.4})$$

where dh is the invariant measure of the local state space.

Let us consider a region of three dimensional space, $\Omega \subset \mathfrak{R}^3$, occupied by a sample of the ensemble. The volume average of local property, $p^i(\Omega)$, is defined as

$$p^i(\Omega) = \frac{1}{\nu(\Omega)} \iiint_{\Omega} p^i(x) dx \quad (\text{B.5})$$

where $\nu(\Omega)$ denotes the volume of region Ω . Introducing relation (B.3) into relation (B.5) yeilds

$$p^i(\Omega) = \sum_j p(h_j) \phi_j^i, \quad (\text{B.6})$$

where

$$\phi_j^i = \frac{\nu^i(h_j)}{\nu(\Omega)} = \frac{1}{\nu(\Omega)} \iiint_{\Omega} I^i(h_j, x) dx \quad (\text{B.7})$$

is the volume fraction of region Ω in sample i that associates with local state subset h_i .

Next, consider the variation of $p^i(\Omega)$ in the ensemble. Define the ensemble average of $p^i(\Omega)$ to be $\bar{p}(\Omega)$. This can be calculated as

$$\bar{p}(\Omega) = \frac{1}{Q} \sum_{i=1}^Q p^i(\Omega) \quad (\text{B.8})$$

where the total number of samples evaluated out of the entire ensemble is Q .

The variance of $p^i(\Omega)$ in the ensemble is $\sigma_{\bar{p}(\Omega)}^2$ and is defined by

$$\sigma_{\bar{p}(\Omega)}^2 = \frac{1}{Q} \sum_{i=1}^Q [p^i(\Omega) - \bar{p}(\Omega)]^2 = \frac{1}{Q} \sum_{i=1}^Q p^i(\Omega)^2 - \bar{p}(\Omega)^2. \quad (\text{B.9})$$

according to the traditional statistical definition of variance. Introducing relation (B.6) into relation (B.9) yields

$$\sigma_{\bar{p}(\Omega)}^2 = \frac{1}{Q} \sum_{i=1}^Q \sum_j \sum_k [p(h_j)p(h_k)\phi_j^i\phi_k^i - \bar{p}(\Omega)^2] \quad (\text{B.10})$$

The principle goal is to evaluate Equation (B.10) for the case of an ensemble possessing statistical homogeneity and ergodicity. The product $\phi_j^i\phi_k^i$ can be re-written in the form of

$$\phi_j^i\phi_k^i = \frac{1}{\nu(\Omega)} \iiint \iiint [I^i(h_j, x)I^i(h_k, x)\theta(x)\theta(y)dx dy] \quad (\text{B.11})$$

In Equation (B.11) the integration are each over the infinite space $\mathfrak{R}^3$ in the variables x and y , and where

$$\theta(x) = \begin{cases} 1 & \text{if } x \in \Omega \\ 0 & \text{otherwise} \end{cases}. \quad (\text{B.12})$$

In statistically homogeneous ensembles it is convenient to express the variable y according to $y = x + \vec{r}$. Then Equation (B.10) can be rewritten, incorporating relation (B.11), and the change of variables, as

$$\sigma_{\bar{p}(\Omega)}^2 = \sum_j \sum_k p(h_j)p(h_k) \left[\frac{1}{\nu(\Omega)^2} \iiint \iiint \zeta_{jk}^i \theta(x)\theta(x+r)dx dr \right] - \bar{p}(\Omega)^2 \quad (\text{B.13})$$

where

$$\zeta_{jk}^i = \frac{1}{Q} \sum_{i=1}^Q I^i(h_j, x)I^i(h_k, x + \vec{r}). \quad (\text{B.14})$$

However, the assumption of statistical homogeneity dictates that the ensemble average depends only upon $\vec{r}$, and not on position x . This result can be expressed as

$$\zeta_{jk}^i = \iiint_{h_j} \iiint_{h_k} f_2(h, h'|\vec{r})dh dh' \quad (\text{B.15})$$

where $f_2(h, h'|\vec{r})dh dh'$ is the two-point probability density associated with the ensemble that has been introduced previously.

In the limit that the measure of each local state subset h_j becomes infinitesimally small, Equation (B.15) can be rewritten by incorporating the sums over indices j and k within integrals taken over the complete local state space according to

$$\sigma_{\bar{p}(\Omega)}^2 = \frac{1}{\nu(\Omega)} \iiint_H \iiint'_H \iiint_{\mathbb{R}^3} f_2(h, h'|\vec{r}) p(h) p(h') \Theta(\vec{r}|\Omega) dr dh dh' - \bar{p}(\Omega)^2 \quad (\text{B.16})$$

The function $\Theta(\vec{r}|\Omega)$ is equal to the volume of intersection of two 3D regions of identical size and shape, Ω , but whose centers of mass are separated by $\vec{r}$. This can be defined by

$$\Theta(\vec{r}|\Omega) = \iiint \theta(x) \theta(x + \vec{r}) dx. \quad (\text{B.17})$$

The right-most term in Equation (B.16) is the square of the ensemble average of the volume average of the local property in the region Ω . Invoking the ergodic hypothesis in conjunction with the assumption of statistical homogeneity in the ensemble enables the replacement of the ensemble averages with volume averages over the volume as it tends to infinity. Thus, the term is readily expressed in terms of the local state distribution function $f(h)$ according to

$$\bar{p}(\Omega) = \left[\iiint_H f(h) p(h) dh \right]^2 \quad (\text{B.18})$$

The local state distribution function $f(h)$ is closely related to the well-known orientation distribution function (ODF) in cases where lattice orientation is the sole consideration of local state. The relationship between $f(h)$ and $f_2(h, h'|\vec{r})$ is

$$f(h) = \iiint_H f_2(h, h'|\vec{r}) dh' \text{ (for all } \vec{r}) \quad (\text{B.19})$$

In this work the local state h was limited to the lattice orientation g defined in Euler orientation space. The local state space H was thus limited to the space of all possible uniquely defined orientations $SO(3)/G$.

Appendix C

Discretization of the Fundamental Zone

The tessera in the fundamental zone B_n are defined as illustrated in Figure 3.5 where it is assumed that each tessera can be expressed as a cube. The boundaries for each side of the cube are defined according to $\phi_{1,i}$, Φ_{ij} and $\phi_{2,j}$ where $i = 1, 2, \dots, I$, $j = 1, 2, \dots, J$ and $k = 1, 2, \dots, K$ are used to enumerate the range of ϕ_1 , Φ and ϕ_2 associated with the individual tessera. The total invariant measure of the fundamental zone is $\pi^2/3$ for lattices with cubic symmetry.

The boundary of each discretization $\phi_{2,k}$ is determined by defining the boundaries of $\phi_{2,k}$ and $k = 0$ and $k = K$ and then solving for the internal points according to

$$\int_{\phi_{2,k}}^{\phi_{2,k+1}} \xi d\phi_2 = \frac{\pi}{6K}, \quad \text{for } k = 1 \dots K - 1 \quad (\text{C.1})$$

where

$$\xi = \frac{1}{\sqrt{\frac{(\cos \phi_2)^2 + 1}{(\cos \phi_2)^2}}} \quad (\text{C.2})$$

is obtained from the integration over Φ and $\pi/6K$ is the invariant measure for each discretization in ϕ_2 .

The interior boundaries for the discretization in Φ are found similarly by defining the boundaries for Φ at $j = 0$ and $j = J$ for each k and then solving the interior points according to

Figure C.1: Definition of invariant measure element B_n in $SO(3)/G$ in terms of the Euler angles.

$$\Phi_{(j-1)k} = \arctan(\cos \Phi_{jk} + \gamma_k) \quad \text{for } j = 1 \dots J \quad \text{and for } k = 0 \dots K - 1 \quad (\text{C.3})$$

where

$$\gamma_k = \left(\frac{\pi}{6JK}\right) \frac{1}{(\phi_{2,k+1} - \phi_{2,k})} \quad \text{for } k = 0 \dots k-1. \quad (\text{C.4})$$

The boundaries of the discretizations along ϕ_1 are easily determined according to

$$\phi_{1,i} = \frac{2\pi}{I}i \quad \text{for } i = 0 \dots I \quad (\text{C.5})$$

Using these methods, the fundamental zone of orientations was discretized into 512 tesserae where ϕ_1 was divided into 32 subdivisions and Φ and ϕ_2 were each divided into 4 subdivisions. The boundaries of the tesserae of the $\Phi - \phi_2$ plane are given in Table C.1 and illustrated in Figure C.2. It is noted that for $k = 4$, Φ is not defined because the boundaries of Φ only need be defined between each pair of $\phi_{2,k}$ for $k = 0..K$.

Table C.1: Boundaries of tesserae in $\Phi - \phi_2$ plane in radians where Fundamental Zone has been discretized into 512 tesserae and where $k = 0 \dots K$.

	k = 0	k = 1	k = 2	k = 3	k = 4
$\phi_{2,k}$	0.0000	0.1857	0.3747	0.5719	0.7854
Φ_{0k}	0.7854	0.7941	0.8213	0.8716	-
Φ_{1k}	1.0136	1.0249	1.0495	1.0931	-
Φ_{2k}	1.2105	1.2173	1.2324	1.2593	-
Φ_{3k}	1.3936	1.3968	1.4040	1.4169	-
Φ_{4k}	1.5708	1.5708	1.5708	1.5708	-

Figure C.2: Boundaries of tesserae in $\Phi - \phi_2$ plane where Fundamental Zone has been discretized into 512 tesserae and where $k = 0 \dots K$.

Appendix D

Results

Table D.1: Standard deviation of Taylor Factors estimated for Fe-3%Si using random sampling method and the statistical variance relationship with 512 discretizations for $\Omega = 1.0 \times 1.0$ mm.

λ	$\sigma_{\overline{\mathbf{M}}(\lambda, \Omega)}$ RAND	$\sigma_{\overline{\mathbf{M}}(\lambda, \Omega)}$ STAT
0.05	0.09148	0.08427
0.10	0.08414	0.07229
0.15	0.06735	0.06380
0.20	0.06659	0.05208
0.25	0.06251	0.04461
0.30	0.05238	0.03300
0.35	0.05362	0.03185
0.40	0.05355	0.02974
0.45	0.05512	0.03411
0.50	0.05995	0.04698
0.55	0.06336	0.05224
0.60	0.06766	0.06002
0.65	0.07494	0.07284
0.70	0.09353	0.08409
0.75	0.09414	0.09431
0.80	0.10798	0.10731
0.85	0.11567	0.11822
0.90	0.12297	0.13070
0.95	0.13552	0.14004

Table D.2: Standard deviation of Taylor Factors estimated for Fe-3%Si using random sampling method and the statistical variance relationship with 512 discretizations for $\Omega = 2.2 \times 2.2$ mm.

λ	$\sigma_{\overline{\mathbf{M}}(\lambda, \Omega)}$ RAND	$\sigma_{\overline{\mathbf{M}}(\lambda, \Omega)}$ STAT
0.05	0.06417	0.03847
0.10	0.05960	0.01970
0.15	0.05493	0.00000
0.20	0.04698	0.00000
0.25	0.03909	0.00000
0.30	0.04045	0.00000
0.35	0.03681	0.00000
0.40	0.03594	0.00000
0.45	0.03529	0.00000
0.50	0.03946	0.00000
0.55	0.04411	0.00000
0.60	0.04650	0.00000
0.65	0.05710	0.03216
0.70	0.05784	0.04490
0.75	0.06363	0.05414
0.80	0.06500	0.06705
0.85	0.08464	0.07573
0.90	0.08497	0.08577
0.95	0.09395	0.09172

Appendix E

C++ Taylor Variance Code

E.1 main.cpp

```
//-----  
// MODULE: main.cpp  
//  
// DESCRIPTION:  
//   This is the MAIN function for Taylor factor Variance Program  
//   This program reads in the appropriate .ang files and then  
//   calculates the variance of the Taylor factor. For further  
//   details see Craig Przybyla's graduate thesis.  
//  
// AUTHOR:  
//   Craig P. Przybyla  
//  
// REVISION HISTORY:  
//   Jan. 21, 2005 - Created  
//-----  
  
#include "main.h"  
  
int main(void)  
{  
    std::cout << "***Taylor Factor Variance Calculator***" <<  
    std::endl;  
    std::cout << "Author: Craig Przybyla" << std::endl;  
    std::cout << std::endl;  
  
    TPvariance* A = new TPvariance();  
    A->ReadFundDOTANGtoMemory("FeSiScan1b.txt");  
    A->ReadFundDOTANGtoMemory("FeSiScan2.txt");
```

```

    A->ReadFundDOTANGtoMemory("FeSiScan3.txt");
    A->ReadFundDOTANGtoMemory("FeSiScan4.txt");
    A->CalculateTwoPointStatistics();
    A->ReadTaylorFactors("TF512ChinMammel.txt");
    A->ReadMacroTaylorFactors("MacroTF512ChinMammel.txt");
    int NumOfLambda = A->GetNumOfLambda();
    for(int i=0;i<NumOfLambda;i++)
    {
        double variance;
        variance = A->CalcTPvariance(i);
        std::cout << variance << std::endl;
    }
    A->WriteVariance("Var512v9x4.txt");
    delete A;
    std::cout << "Done" << std::endl;
    return(0);
}

```

E.2 main.hpp

```

//-----
// MODULE: main.hpp
//
// DESCRIPTION:
//   This is the header for main.cpp
//
// AUTHOR:
//   Craig P. Przybyla
//
// REVISION HISTORY:
//   Jan. 21, 2005 - created
//-----

#ifndef MAIN_H
#define MAIN_H

//Included Libraries

#include <iostream>
#include <string>

#include "dotAng.h"
#include "TPvariance.h"

```

```
#endif //MAIN_H
```

E.3 dotAng.hpp

```
//-----  
// MODULE: dotAng.hpp  
//  
// DESCRIPTION:  
//   This is a class to READ in and store information contained in  
//   .ang files created in TexSex's OIM software  
//  
// AUTHOR:  
//   Craig P. Przybyla  
//  
// REVISION HISTORY:  
//   February 2004 - created  
//   Jan. 21, 2005 - converted into a class structure -cpp  
//-----  
  
#ifndef DOTANG_H  
#define DOTANG_H  
  
//Included Libraries  
#include <iostream>  
#include <fstream>  
#include <string>  
#include <sstream>  
#include <vector>  
#include <math.h>  
  
//Constants  
#define PI 3.14159265359  
#define lim(x) (x)>1.0?1.0:(x)<-1.0?-1.0:(x)  
//User Defined Constants  
#define SYMMETRY "cubicbranch.txt"  
#define PHI1PARTITIONS 32 //Number of disc along phi1  
#define PHIPARTITIONS 4 //Number of disc along PHI  
#define PHI2PARTITIONS 4 //Number of disc along phi2  
#define MAXVECTORSIZE 90 //Maximum Vector Size must be less than the  
                        //scan size  
#define INTERVALS 1000000
```

```

struct DATA_POINT
{
    double mPhi1;
    double mPHI;
    double mPhi2;
    float mX;
    float mY;
    float mCindex;
    int mBoxIndex;
};

struct DISC_LOAF
{
    double mPhi1[PHI1PARTITIONS+1];
    double mPHI[PHI1PARTITIONS+1][PHI2PARTITIONS+1];
    double mPhi2[PHI2PARTITIONS+1];
};

struct DISC_LOAF_CENTERS
{
    double mPhi1;
    double mPHI;
    double mPhi2;
};

class DotAng
{
public:

    DotAng();
    ~DotAng();
    //Read DotAng
    void ReadFundDOTANGtoMemory(std::string angfilename);
    void ReadFundDOTANGtoFile(std::string angfilename,
        std::string outfile);

    void WriteVecTestFile(int index, std::string testfile);

    //Discretize Loaf
    void DiscretizeLOAF(void);
    void GetBoxCoordinates(void);
    void PrintDiscretizedLoaf(void);
    void SaveDiscretizedCenters(std::string filesave);

```

```

//Export Information
int GetNumDataPoints(int FileIndex);
void GetDataPoint(int FileIndex, double &Phi1,
    double &PHI, double &Phi2, int point);
double GetScanStepSize(int FileIndex);
double GetTwoPointStat(int g1,int g2, int vector,
    int vecsize);
int GetMaxVectorLength(void);
int GetNumOfTessera(void);

//Calculate Two-point statistics
void CalculateODF(void);
double GetODFPoint(int g);
void WriteODF(std::string filename);
void CalculateTwoPointStatistics(void);
void Printf2ToFile(std::string filename);

//File Read Indicator
bool mAngFileRead;
bool mCalTwoPointStats;
bool mCalODF;

private:

//Data structure to store data
std::string mGridType;
std::vector<std::vector<DATA_POINT*> > mDataFiles;
void SetGridType(std::string type);

//Fundamentalize
int mSymOps[24][3][3];
bool mSymOpsRead;
void FundamentalizeDataPoint(double &Phi1, double &PHI,
    double &Phi2);
void GetSymmOps(void);
void matrix3x3_mult1(double A[3][3],int B[3][3],
    double C[3][3]);

//Discritize
int mNumOfDisc;
struct DISC_LOAF mLoaf;
struct DISC_LOAF_CENTERS* mCenters;
bool mLoafMade;
int GetDiscritizedLoafIndex(double Phi1, double PHI,
    double Phi2, float ConfidenceIndex=1);

```

```

//ODF
double* mODF;
int mTotalDataPoints;

//TwoPoint Function
int** mDataGrid;
int mNumOfVectors;
int mMaxVecSize;
float**** mf2;
void CreateHexScanGrid(int FileIndex, int &SizeX,
    int &SizeY);
void CreateTestHexScanGrid(int &SizeX, int &SizeY);
int GetTwoPointVector1(int SizeX, int SizeY, int vecSize);
int GetTwoPointVector2(int SizeX, int SizeY, int vecSize);
int GetTwoPointVector3(int SizeX, int SizeY, int vecSize);
int GetTwoPointVector4(int SizeX, int SizeY, int vecSize);
int GetTwoPointVector5(int SizeX, int SizeY, int vecSize);
int GetTwoPointVector6(int SizeX, int SizeY, int vecSize);
};

#endif // DOTANG_H

```

E.4 dotAng.cpp

```

//-----
// MODULE: dotAng.cpp
//
// DESCRIPTION:
//   These are the class functions for dotAng.h
//   These functions READ in and store information contained in
//   .ang files created in TexSex's OIM software
//
// AUTHOR:
//   Craig P. Przybyla
//
// REVISION HISTORY:
//   November 2003 - created
//   Jan. 21, 2005 - converted into a class structure -cpp
//   May 28, 2005 - Added Two Point Functions -cpp
//-----

#include "dotAng.h"

```

```

DotAng::DotAng()
{
    mAngFileRead = false;
    mLoafMade = false;
    mSymOpsRead = false;
    mCalTwoPointStats = false;
    mCalODF = false;

    mNumOfDisc = PHI1PARTITIONS*PHIPARTITIONS*PHI2PARTITIONS;
    mNumOfVectors = 6;
    mMaxVecSize = MAXVECTORSIZE;

    mCenters = new DISC_LOAF_CENTERS [mNumOfDisc];

    //Allocate Memory for ODF
    mODF = new double [mNumOfDisc];
    for(int i=0;i<mNumOfDisc;i++)
    {
        mODF[i] = 0.0;
    }
    mTotalDataPoints = 0;
}

DotAng::~DotAng()
{
    //clear mPoints
    int NumFiles = mDataFiles.size();
    for(int i=0;i<NumFiles;i++)
    {
        int length = mDataFiles[i].size();
        for (int j=0;j<length;j++)
        {
            delete mDataFiles[i][j];
        }
    }

    //clear mf2
    if(mCalTwoPointStats == true)
    {
        for(int i=0;i<(mNumOfDisc+1);i++)
        {
            for(int j=0;j<(mNumOfDisc+1);j++)
            {
                for(int k=0;k<mNumOfVectors;k++)

```



```

        {
            delete(mf2[i][j][k]);
        }
        delete(mf2[i][j]);
    }
    delete(mf2[i]);
}
delete(mf2);
}
delete(mCenters);
delete mODF;
}

void DotAng::SetGridType(std::string type)
{
    mGridType = type;
}

//Multiplies two 3x3 matrices (A and B)
//and stores the result in C.
void DotAng::matrix3x3_mult1(double A[3][3],int B[3][3],
    double C[3][3])
{
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            C[i][j]=0;
            for(int k=0;k<3;k++)
            {
                C[i][j]+=A[i][k]*B[k][j];
            }
        }
    }
}

void DotAng::ReadFundDOTANGtoMemory(std::string angfilename)
{
    //open file
    std::ifstream angfile(angfilename.c_str());
    if(angfile==NULL || angfilename.length()==0)
    //End program if file is not found or if file is empty
    {
        std::cout << "Error: .ang file not found or is empty"
            << std::endl;
    }
}

```

```

        exit(1);
    }

    //Temporary strings to read header
    std::string header;
    std::string pound("#");
    int interrupt = 0;
    //Read through header
    while(interrupt == 0 && angfile.eof() == false )
    {
        std::getline(angfile, header, '\n');
        if(header.compare(0,1,pound)){interrupt = 1;}
    }
    angfile.seekg(-(header.size()+1),std::ios::cur);
    //Go back one line in dotang file

    //Read in symmetry operators
    if(mSymOpsRead == false)
    {
        GetSymmOps();
    }
    //discretize fundamental zone
    if(mLoafMade == false)
    {
        DiscretizeLOAF();
    }

    std::vector<DATA_POINT*> filepoints;
    while(angfile.eof()==0)
    {
        std::string data;
        float phi1,PHI,phi2,x,y,cindex;

        //Read data
        std::getline(angfile, data, '\n');
        if(sscanf(data.c_str(),"%f %f %f %f %f %*f %f %*d %*d",
            &phi1,&PHI,&phi2,&x,&y,&cindex)!=6)
        {
            continue;
        }

        //create temporary pointer
        DATA_POINT* a=new DATA_POINT;
        a->mPhi1 = phi1;
        a->mPHI = PHI;
    }

```

```

        a->mPhi2 = phi2;
        a->mX = x;
        a->mY = y;
        a->mCindex = cindex;

        FundamentalizeDataPoint(a->mPhi1,a->mPHI,a->mPhi2);
        //put datapoint into fundamental zone
        a->mBoxIndex =
            GetDiscritizedLoafIndex(a->mPhi1,a->mPHI,a->mPhi2);
        //assign datapoint box in discritized loaf
        filepoints.push_back(a); //Store data in vector mPoints
    }
    mDataFiles.push_back(filepoints);

    angfile.close();

    std::cout << "Number of datapoints in " << angfilename <<
        " is: " << filepoints.size() << std::endl;
    mAngFileRead = true; //File read indicator

    return;
}

void DotAng::ReadFundDOTANGtoFile(std::string angfilename,
    std::string outfile)
{
    //open file
    std::ifstream angfile(angfilename.c_str());
    if(angfile==NULL || angfilename.length()==0)
        //End program if file is not found or if file is empty
    {
        std::cout << "Error: .ang file not found or is empty"
            << std::endl;
        exit(1);
    }

    //save file
    std::ofstream datafile(outfile.c_str());
    if (datafile==NULL)
    {
        std::cout << "Error: Could not open
            test file to save box coordinates!" << std::endl;
    }

    //Temporary strings to read header

```

```

std::string header;
std::string pound("#");
int interrupt = 0;
//Read through header
while(interrupt == 0 && angfile.eof() == false )
{
    std::getline(angfile, header, '\n');
    if(header.compare(0,1,pound)){interrupt = 1;}
}
angfile.seekg(-(header.size()+1),std::ios::cur);
//Go back one line in dotang file

//Read in symmetry operators
if(mSymOpsRead == false)
{
    GetSymmOps();
}
//discretize fundamental zone
if(mLoafMade == false)
{
    DiscretizeLOAF();
}

int count=0;
while(angfile.eof()==0)
{
    std::string data;
    double phi1,PHI,phi2;
    float x,y,cindex;

    //Read data
    std::getline(angfile, data, '\n');
    if(sscanf(data.c_str(),"%lf %lf %lf %f %f %*f %f
        %*d %*d",&phi1,&PHI,&phi2,&x,&y,&cindex)!=6)
    {
        continue;
    }

    FundamentalizeDataPoint(phi1,PHI,phi2);
    int BoxIndex = GetDiscretizedLoafIndex(phi1,PHI,phi2);

    //write to file
    datafile << BoxIndex << " " << phi1 << " " << PHI
    << " " << phi2 << " " << x << " " << y << std::endl;
    count++;
}

```

```

}
datafile << "-5" << std::endl;

std::cout << "Number of datapoints in " << angfilename
    << " is: " << count << std::endl;

angfile.close();
datafile.close();

return;
}

void DotAng::FundamentalizeDataPoint(double &Phi1,
    double &PHI, double &Phi2)
{
    double g[3][3],n[3][3],gn[3][3];
    double a1,a2,a3;
    double bound;

    //define G ("Active" this is transpose
    //of the g used by Bunge pg21)
    g[0][0]=cos(PHI1)*cos(PHI2)-sin(PHI1)*sin(PHI2)*cos(PHI);
    g[0][1]=-cos(PHI1)*sin(PHI2)-sin(PHI1)*cos(PHI2)*cos(PHI);
    g[0][2]=sin(PHI1)*sin(PHI);
    g[1][0]=sin(PHI1)*cos(PHI2)+ cos(PHI1)*sin(PHI2)*cos(PHI);
    g[1][1]=-sin(PHI1)*sin(PHI2)+ cos(PHI1)*cos(PHI2)*cos(PHI);
    g[1][2]=-cos(PHI1)*sin(PHI);
    g[2][0]=sin(PHI2)*sin(PHI);
    g[2][1]=cos(PHI2)*sin(PHI);
    g[2][2]=cos(PHI);

    int exit; //interrupt
    for(int s=0;s<24;s++)
    {
        exit=0;
        for(int j=0;j<3;j++)
        {
            for(int k=0;k<3;k++)
            {
                n[j][k]=0;
                gn[j][k]=0;
            }
        }

        //TO rotate g matrix == G'=G*R

```

```

//Mult G*R
matrix3x3_mult1(g,mSymOps[s],gn);

//solve for euler angles or rotated orientation
a2=acos(lim(gn[2][2]));
a3=atan2(gn[2][0],gn[2][1]);
a1=atan2(gn[0][2],-gn[1][2]);

//force angles to be positive
while(a1<0){a1=a1+(2*PI);}
while(a2<0){a2=a2+(2*PI);}
while(a3<0){a3=a3+(2*PI);}

if(0<=a3 && a3<(PI/4))
{
    bound=atan2(1,cos(a3));
    if(bound<=a2 && a2<(PI/2))
    {
        Phi1=a1;
        PHI=a2;
        Phi2=a3;
        exit=1;
    }
}
if(exit==1){break;}
if(s==24)
{
    std::cout << "Error: Did not find point
        in fundamental zone!" << std::endl;
    Phi1=0.0;
    PHI=0.0;
    Phi2=0.0;
}
}
return;
}

void DotAng::DiscretizeLOAF(void)
{
    mLoafMade = true;

    //Centerpoints
    double TempPHI[PHIPARTITIONS][PHI2PARTITIONS];
    double TempPhi2[PHI2PARTITIONS];

```

```

//Define total volume of Fundamental zone
double LOAFVOL=3.2898681337;

//Discritize phi2
double x,y;

//Variables for Numerical Itegration
int in=INTERVALS;
double tolerance = 1/(static_cast<double>(in));
int NumPhi2=PHI2PARTITIONS; //user defined constant
double dphi2=(PI/4.0)/in; //stepsize

double Phi2DiscArea=(LOAFVOL/(2*PI*NumPhi2));
double Area=0; //area tracker
int halfcount=1; //counter
int count=1; //counter
//Discritize Phi2
mLoaf.mPhi2[0]=0.0;
mLoaf.mPhi2[NumPhi2]=PI/4.0;
for(int i=0; i<in*1.1;i++)
{
    //midpoint numerical integration
    x=(i*dphi2)+(dphi2/2.0);
    //Accounts for Invariant Measure (see thesis
    //Notebook Craig Przybyla pg55)
    y=cos(x)/(sqrt(cos(x)*cos(x)+1.0));
    Area=Area+dphi2*y; //ith area element
    //Find Discritization
    if((0.5*halfcount*Phi2DiscArea-tolerance)< Area
        && Area < (0.5*halfcount*Phi2DiscArea+tolerance))
    {
        //Border Points
        if((count*Phi2DiscArea-tolerance)< Area
            && Area < (count*Phi2DiscArea+tolerance))
        {
            if(count==NumPhi2) //force endpoint to be exact
            {
                break;
            }
            mLoaf.mPhi2[count]=((i+1)*dphi2);
            halfcount++;
            count++;
        }
        //CenterPoints
    }
    else

```

```

        {
            TempPhi2[count-1]=((i+1)*dphi2);
            halfcount++;
        }
    }

//discritize PHI
int NumPHI=PHIPARTITIONS; //user defined constant
double PHIDiscArea=Phi2DiscArea/NumPHI;
for(int i=0;i<NumPhi2;i++)
{
    mLoaf.mPHI[0][i]=atan2(1,cos(mLoaf.mPhi2[i]));
    mLoaf.mPHI[NumPHI][i]=(PI/2.0);
    double gamma = PHIDiscArea*(1/(mLoaf.mPhi2[i+1] -
        mLoaf.mPhi2[i]));
    // Gamma as defined pg 60 Master's Thesis Notes, C Przybyla
    for(int j=NumPHI;j>1;j--)
    {
        mLoaf.mPHI[j-1][i]=acos(cos(mLoaf.mPHI[j][i])+gamma);
    }
}

//PHI centers
double HalfPHIDiscArea = PHIDiscArea/2;
for(int i=0;i<NumPhi2;i++)
{
    double gamma = PHIDiscArea*(1/(mLoaf.mPhi2[i+1] -
        mLoaf.mPhi2[i]));
    double halfgamma = HalfPHIDiscArea*(1/(mLoaf.mPhi2[i+1] -
        mLoaf.mPhi2[i]));
    TempPHI[NumPHI-1][i] = acos(cos(mLoaf.mPHI[NumPHI][i]) +
        halfgamma);
    // Gamma as defined pg 60 Master's Thesis Notes, C. Przybyla
    for(int j=(NumPHI-1);j>0;j--)
    {
        TempPHI[j-1][i]=acos(cos(TempPHI[j][i])+gamma);
    }
}

//discritize phi1
int NumPhi1=PHI1PARTITIONS; //User Defined Constant
double dphi1=(2*PI)/(NumPhi1);
for(int i=0;i<NumPhi1;i++)
{

```



```

        mLoaf.mPhi1[i] = i*dphi1;
    }
    mLoaf.mPhi1[NumPhi1] = 2.0*PI;

    //Find Centers
    //phi2
    //Estimate Midpoints of PHI
    for(int j=0;j<NumPhi2;j++)
    {
        for(int k=0;k<NumPHI;k++)
        {
            for(int l=0;l<NumPhi1;l++)
            {
                //from 0 to NOB-1
                int boxNum = ((l*NumPHI*NumPhi2)+(k*NumPhi2)+j);
                mCenters[boxNum].mPhi2=TempPhi2[j];
                mCenters[boxNum].mPHI=TempPHI[k][j];
                mCenters[boxNum].mPhi1=((mLoaf.mPhi1[l+1] +
                    mLoaf.mPhi1[l])/2);
            }
        }
    }

    std::cout << "Loaf was discretized to " << NumPhi1 << " (Phi1) X "
        << NumPHI << " (Phi) X " << NumPhi2 << " (Phi2)."
        << std::endl;
    return;
}

int DotAng::GetNumOfTessera(void)
{
    return(mNumOfDisc);
}

void DotAng::SaveDiscretizedCenters(std::string filesave)
{
    //open file
    std::ofstream savefile(filesave.c_str());
    //End program if file is not found or if file is empty
    if(savefile==NULL)
    {
        std::cout << "Error: Centers file not opened" << std::endl;
        exit(1);
    }
}

```

```

        savefile << "# Box#" << "\t" << "Phi1" << "\t" << "PHI"
            << "\t" << "Phi2" << std::endl;
    for(int i=0;i<mNumOfDisc;i++)
    {
        savefile << (i+1) << "\t" << mCenters[i].mPhi1 << "\t"
            << mCenters[i].mPHI << "\t" << mCenters[i].mPhi2
            << std::endl;
    }
}

int DotAng::GetDiscritizedLoafIndex(double Phi1, double PHI,
double Phi2, float ConfidenceIndex)
{
    int boxNum;
    if(ConfidenceIndex < 0.1)
    {
        boxNum = mNumOfDisc;
        return(boxNum);
    }

    //read in data and assign index numbers
    int exit;
    int NumPhi1=PHI1PARTITIONS; //user Defined Constant
    int NumPHI=PHIPARTITIONS; //user defined constant
    int NumPhi2=PHI2PARTITIONS; //user defined constant

    exit=0;
    for(int j=0;j<NumPhi2;j++)
    {
        if(mLoaf.mPhi2[j] <= Phi2 && Phi2 < mLoaf.mPhi2[j+1])
        {
            for(int k=0;k<NumPHI;k++)
            {
                if(mLoaf.mPHI[k][j] < PHI
                    && PHI <= mLoaf.mPHI[k+1][j])
                {
                    for(int l=0;l<NumPhi1;l++)
                    {
                        if(mLoaf.mPhi1[l] <= Phi1
                            && Phi1 < mLoaf.mPhi1[l+1])
                        {
                            //from 0 to NOB-1
                            boxNum = ((l*NumPHI*NumPhi2) +
                                (k*NumPhi2)+j);
                            exit=1;
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
        if(exit==1)break;
    }
    }
    if(exit==1)break;
}
    if(exit==1)break;
}
if(exit!=1)
{
    std::cout << "Error: Did not find data point " << Phi1
        << " " << PHI << " " << Phi2 << "
        in the fundamental zone!" << std::endl;
    boxNum = 0;
}
return (boxNum);
}

```

```

void DotAng::GetBoxCoordinates(void)
{
    //box coordinates
    std::string filename = "boxcoordinates.txt";
    std::ofstream boxfile(filename.c_str());
    if (boxfile==NULL)
    {
        std::cout << "Error: Could not open test file
            to save box coordinates!" << std::endl;
    }

    int NumPhi1=PHI1PARTITIONS; //User Defined Constant
    int NumPHI=PHI1PARTITIONS; //user defined constant
    int NumPhi2=PHI2PARTITIONS; //user defined constant

    for(int i=0;i<NumPhi2;i++)
    {
        for(int j=0;j<NumPHI;j++)
        {
            for(int k=0;k<NumPhi1;k++)
            {
                //from 0 to NOB-1
                int box=((k*NumPHI*NumPhi2)+(j*NumPhi2)+i);
                boxfile << box << " " << k << " " << j << " "
                    << i << std::endl;
            }
        }
    }
}

```

```

    }
}
return;
}

void DotAng::PrintDiscretizedLoaf(void)
{
    //box coordinates
    std::string filename = "discloafest.txt";
    std::ofstream discfile(filename.c_str());
    if (discfile==NULL)
    {
        std::cout << "Error: Could not open test file to
            save box coordinates!" << std::endl;
    }

    int NumPhi1=PHI1PARTITIONS; //User Defined Constant
    int NumPHI=PHIPARTITIONS; //user defined constant
    int NumPhi2=PHI2PARTITIONS; //user defined constant

    //Print phi1 to screen
    discfile << "Phi1" << std::endl;
    for(int i=0;i<=NumPhi1;i++)
    {
        discfile << mLoaf.mPhi1[i] << std::endl;
    }

    //Print PHI to screen
    discfile << "PHI" << std::endl;
    for(int i=0;i<NumPhi2;i++)
    {
        for(int j=0;j<=NumPHI;j++)
        {
            discfile << mLoaf.mPHI[j][i] << std::endl;
        }
    }

    //Print phi2 to screen
    discfile << "Phi2" << std::endl;
    for(int i=0;i<=NumPhi2;i++)
    {
        discfile << mLoaf.mPhi2[i] << std::endl;
    }
    return;
}

```

```

}

void DotAng::GetSymmOps(void)
{
    mSymOpsRead = true;
    //open cubic symmetry file
    std::ifstream CubicSymm(SYMMETRY);
    if(CubicSymm==NULL)
    //End program if file is not found or if file is empty
    {
        std::cout << "Error: Symmetry file not found"
            << std::endl;
        exit(1);
    }
    for(int s=0;s<24;s++)
    {
        for(int j=0;j<3;j++)
        {
            for(int k=0;k<3;k++)
            {
                mSymOps[s][j][k] = 0;
            }
        }
        //read in rotation R
        for(int j=0;j<3;j++)
        {
            for(int k=0;k<3;k++)
            {
                CubicSymm >> mSymOps[s][j][k];
            }
        }
    }
    return;
}

int DotAng::GetNumDataPoints(int FileIndex)
{
    return(static_cast<int>(mDataFiles[FileIndex].size()));
}

void DotAng::GetDataPoint(int FileIndex, double &Phi1,
    double &PHI, double &Phi2, int point)
{
    Phi1 = mDataFiles[FileIndex][point]->mPhi1;
    PHI = mDataFiles[FileIndex][point]->mPHI;
}

```

```

        Phi2 = mDataFiles[FileIndex][point]->mPhi2;
    }

double DotAng::GetScanStepSize(int FileIndex)
{
    //This only works for Hexagonal Grids
    return(mDataFiles[FileIndex][1]->mX -
           mDataFiles[FileIndex][0]->mX);
    //Scan StepSize in microns
}

void DotAng::CreateHexScanGrid(int FileIndex,int &SizeX, int &SizeY)
{
    //Determine sizeX
    SizeX = 0;
    for(unsigned int p=0;p<mDataFiles[FileIndex].size();p++)
    {
        if(mDataFiles[FileIndex][0]->mY !=
           mDataFiles[FileIndex][p]->mY)
        {
            break;
        }
        else
        {
            SizeX++;
        }
    }
    //Determine sizeY
    SizeY=1;
    float current = mDataFiles[FileIndex][0]->mY;
    for(unsigned int p=0;p<mDataFiles[FileIndex].size();p++)
    {
        if(current != mDataFiles[FileIndex][p]->mY)
        {
            current = mDataFiles[FileIndex][p]->mY;
            SizeY++;
        }
    }

    mDataGrid = new int* [SizeX];
    for(int i=0;i<SizeX;i++)
    {
        mDataGrid[i] = new int [SizeY];
    }
}

```

```

int countX = 0;
int countY = 0;
for(unsigned int p=0;p<mDataFiles[FileIndex].size();p++)
{
    if(countY%2 == 0) //odd row
    {
        if(countX == SizeX)
        {
            countX = 0;
            countY++;
        }
        mDataGrid[countX][countY] =
            mDataFiles[FileIndex][p]->mBoxIndex;
        countX++;
    }
    else //even row
    {
        if(countX == (SizeX-1))
        {
            countX = 0;
            countY++;
        }
        mDataGrid[countX][countY] =
            mDataFiles[FileIndex][p]->mBoxIndex;
        countX++;
    }
}
}

void DotAng::CreateTestHexScanGrid(int &SizeX, int &SizeY)
{
    SizeX = 5;
    SizeY = 5;
    mDataGrid = new int* [SizeX];
    for(int i=0;i<SizeX;i++)
    {
        mDataGrid[i] = new int [SizeY];
    }

    int countX = 0;
    int countY = 0;
    int point = 0;
    for(int j=0;j<SizeY;j++)
    {
        for(int i=0;i<SizeX;i++)

```

```

{
    if(countY%2 == 0) //odd row
    {
        if(countX == SizeX)
        {
            point = 1;
            countX = 0;
            countY++;
        }
        mDataGrid[countX][countY] = point;
        countX++;
    }
    else //even row
    {
        if(countX == (SizeX-1))
        {
            point = 0;
            countX = 0;
            countY++;
        }
        mDataGrid[countX][countY] = point;
        countX++;
    }
}

//Print Grid to Screen
for(int j=0;j<SizeY;j++)
{
    for(int i=0;i<SizeX;i++)
    {
        std::cout << mDataGrid[i][j] << " ";
    }
    std::cout << std::endl;
}

}

void DotAng::CalculateODF()
{
    mCalODF = true;
    for(unsigned int i=0;i<mDataFiles.size();i++)
    {
        for(unsigned int j=0;j<mDataFiles[i].size();j++)
        {
            mODF[mDataFiles[i][j]->mBoxIndex]++;
            mTotalDataPoints++;
        }
    }
}

```



```

    }
}
for(int i=0;i<mNumOfDisc;i++)
{
    mODF[i] = mODF[i]/mTotalDataPoints;
}
}

double DotAng::GetODFPoint(int g)
{
    return(mODF[g]);
}

void DotAng::WriteODF(std::string filename)
{
    //save file
    std::ofstream savefile(filename.c_str());
    if (savefile==NULL)
    {
        std::cout << "Error: Could not open file to save ODF"
            << std::endl;
        exit(1);
    }
    for(int i=0;i<mNumOfDisc;i++)
    {
        savefile << (i+1) << " " << mODF[i] << std::endl;
    }
}

void DotAng::CalculateTwoPointStatistics()
{
    std::cout << "Calculating two-point statistics..." << std::endl;

    mCalTwoPointStats = true;

    //Initialize mf2 and allocate memory
    mf2 = new float*** [mNumOfDisc+1];
    for(int i=0;i<(mNumOfDisc+1);i++)
    {
        mf2[i] = new float** [mNumOfDisc+1];
        for(int j=0;j<(mNumOfDisc+1);j++)
        {
            mf2[i][j] = new float* [mNumOfVectors];
            for(int k=0;k<mNumOfVectors;k++)
            {

```

```

        mf2[i][j][k] = new float [mMaxVecSize];
        for(int l=0;l<mMaxVecSize;l++)
        {
            mf2[i][j][k][l] = 0.0;
        }
    }
}

int** CountVec;
CountVec = new int* [6];
for(int i=0;i<6;i++)
{
    CountVec[i] = new int [mMaxVecSize];
    for(int j=0;j<mMaxVecSize;j++)
    {
        CountVec[i][j] = 0;
    }
}

for(unsigned int i=0;i<mDataFiles.size();i++)
{
    //Grid Generator
    int SizeX = 0;
    int SizeY = 0;

    //Create Scan Grid
    CreateHexScanGrid(i,SizeX,SizeY);

    for(int vecSize=1;vecSize<=mMaxVecSize;vecSize++)
    {
        CountVec[0][vecSize-1] = CountVec[0][vecSize-1] +
            GetTwoPointVector1(SizeX,SizeY,vecSize);
        CountVec[1][vecSize-1] = CountVec[1][vecSize-1] +
            GetTwoPointVector2(SizeX,SizeY,vecSize);
        CountVec[2][vecSize-1] = CountVec[2][vecSize-1] +
            GetTwoPointVector3(SizeX,SizeY,vecSize);
        CountVec[3][vecSize-1] = CountVec[3][vecSize-1] +
            GetTwoPointVector4(SizeX,SizeY,vecSize);
        CountVec[4][vecSize-1] = CountVec[4][vecSize-1] +
            GetTwoPointVector5(SizeX,SizeY,vecSize);
        CountVec[5][vecSize-1] = CountVec[5][vecSize-1] +
            GetTwoPointVector6(SizeX,SizeY,vecSize);
    } //End VecSize
    //clear mDataGrid

```

```

        if(SizeX > 1)
        {
            for(int i=0;i<SizeX;i++)
            {
                delete(mDataGrid[i]);
            }
            delete(mDataGrid);
        }
    }
    for(int i=0;i<(mNumOfDisc);i++)
    {
        for(int j=0;j<(mNumOfDisc);j++)
        {
            for(int vec=0;vec<6;vec++)
            {
                for(int vecSize=0;vecSize<mMaxVecSize;vecSize++)
                {
                    mf2[i][j][vec][vecSize] =
                        mf2[i][j][vec][vecSize] /
                        CountVec[vec][vecSize];
                }
            }
        }
    }

    for(int vec=0;vec<6;vec++)
    {
        delete CountVec[vec];
    }
    delete CountVec;
}

double DotAng::GetTwoPointStat(int g1,int g2, int vector,
    int vecsize)
{
    return(mf2[g1][g2][vector][vecsize]);
}

int DotAng::GetMaxVectorLength()
{
    return(this->mMaxVecSize);
}

void DotAng::Printf2ToFile(std::string filename)
{

```

```

//open file
std::ofstream savefile(filename.c_str());
if(savefile==NULL)
//End program if file is not found or if file is empty
{
    std::cout << "Error: Unable to open f2 save file"
        << std::endl;
    exit(1);
}

savefile << "g1" << "\t" << "g2" << "\t" << "Vector"
    << "\t" << "Lenght" << "\t" << "f2" << std::endl;
for(int i=0;i<mNumOfDisc;i++)
{
    for(int j=0;j<mNumOfDisc;j++)
    {
        for(int k=1;k<=mNumOfVectors;k++)
        {
            for(int l=1;l<=mMaxVecSize;l++)
            {
                savefile << i << "\t" << j << "\t" << k
                    << "\t" << l << "\t"
                    << mf2[i][j][k-1][l-1] << std::endl;
            }
        }
    }
}

}

int DotAng::GetTwoPointVector1(int SizeX, int SizeY,
int vecSize)
{
    int xStart,xStop,xStep,yStart,yStop,yStep;

    //Vector 1
    //(i,j) -> (i+1,j)
    int count = 0;
    int CurrentRow = 1;
    xStart = 0;
    xStep = vecSize;
    yStart = 0;
    yStop = SizeY;
    yStep = 0;
    for(int j=0;j<yStop;j++)
    {
        if(CurrentRow%2 != 0) //Odd rows are of Size = SizeX

```

```

    {
        xStop = SizeX-vecSize;
        for(int i=0;i<xStop;i++)
        {
            mf2[mDataGrid[i][j]][mDataGrid[i+xStep][j+yStep]]
                [0][vecSize-1]++;
            count++;
        }
    }
    else //even rows are of Size = SizeX-1
    {
        xStop = SizeX-vecSize-1;
        for(int i=0;i<xStop;i++)
        {
            mf2[mDataGrid[i][j]][mDataGrid[i+xStep][j+yStep]]
                [0][vecSize-1]++;
            count++;
        }
    }
    CurrentRow++;
}
return(count);
//End Vector 1
}

```

```

int DotAng::GetTwoPointVector2(int SizeX, int SizeY,
int vecSize)
{
    int xStart,xStop,xStep,yStart,yStop,yStep;

    //Vector 2
    // odd (i,j) -> (i,j-1)
    // even (i,j) -> (i+1,j-1)
    int count = 0;
    //Grid Searching Parameters
    xStart = 0;
    yStart = vecSize;
    yStop = SizeY;
    yStep = -vecSize;
    int CurrentRow = yStart+1; //1st row = 1
    if(vecSize%2==0) //even vector size
    {
        for(int j=yStart;j<yStop;j++)
        {
            if(CurrentRow%2 == 0)

```

```

//even rows are of Size = SizeX-1
{
    xStop = SizeX-vecSize/2-1;
    xStep = vecSize/2;
    for(int i=xStart;i<xStop;i++)
    {
        mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
            [j+yStep]][1][vecSize-1]++;
        count++;
    }
    CurrentRow++;
}
else //Odd rows are of Size = SizeX
{
    xStop = SizeX-vecSize/2;
    xStep = vecSize/2;
    for(int i=xStart;i<xStop;i++)
    {
        mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
            [j+yStep]][1][vecSize-1]++;
        count++;
    }
    CurrentRow++;
}
}
else //odd vector size
{
    for(int j=yStart;j<yStop;j++)
    {
        if(CurrentRow%2 == 0)
        //even rows are of Size = SizeX-1
        {
            xStop = SizeX-(vecSize+1)/2;
            xStep = (vecSize+1)/2;
            for(int i=xStart;i<xStop;i++)
            {
                mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                    [j+yStep]][1][vecSize-1]++;
                count++;
            }
        }
        else //Odd rows are of Size = SizeX
        {
            xStop = SizeX-(vecSize+1)/2;

```

```

        xStep = (vecSize-1)/2;
        for(int i=xStart;i<xStop;i++)
        {
            mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                [j+yStep]][1][vecSize-1]++;
            count++;
        }
        currentRow++;
    }
}
return(count);
//End Vector 2
}

```

```

int DotAng::GetTwoPointVector3(int SizeX, int SizeY,
int vecSize)
{
    int xStart,xStop,xStep,yStart,yStop,yStep;

    //Vector 3
    // odd (i,j) -> (i-1,j-1)
    // even (i,j) -> (i,j-1)
    int count = 0;
    //Grid Searching Parameters
    yStart = vecSize;
    yStop = SizeY;
    yStep = -vecSize;
    int CurrentRow = yStart+1; //1st row = 1
    if(vecSize%2==0) //even vector size
    {
        xStart = vecSize/2;
        xStep = -vecSize/2;
        for(int j=yStart;j<yStop;j++)
        {
            if(CurrentRow%2 == 0)
            //even rows are of Size = SizeX-1
            {
                xStop = SizeX-1;
                for(int i=xStart;i<xStop;i++)
                {
                    mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                        [j+yStep]][2][vecSize-1]++;
                    count++;
                }
            }
        }
    }
}

```

```

        CurrentRow++;
    }
    else //Odd rows are of Size = SizeX
    {
        xStop = SizeX;
        for(int i=xStart;i<xStop;i++)
        {
            mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                [j+yStep]][2][vecSize-1]++;
            count++;
        }
        CurrentRow++;
    }
}
else //odd vector size
{
    for(int j=yStart;j<yStop;j++)
    {
        if(CurrentRow%2 == 0) //even rows are of Size = SizeX-1
        {
            xStart = (vecSize-1)/2;
            xStop = SizeX-1;
            xStep = -(vecSize-1)/2;
            for(int i=xStart;i<xStop;i++)
            {
                mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                    [j+yStep]][2][vecSize-1]++;
                count++;
            }
        }
        else //Odd rows are of Size = SizeX
        {
            xStart = (vecSize+1)/2;
            xStop = SizeX;
            xStep = -(vecSize+1)/2;
            for(int i=xStart;i<xStop;i++)
            {
                mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                    [j+yStep]][2][vecSize-1]++;
                count++;
            }
        }
        CurrentRow++;
    }
}

```



```

    }
    return(count);
    //End Vector 3
}

int DotAng::GetTwoPointVector4(int SizeX, int SizeY,
    int vecSize)
{
    int xStart,xStop,xStep,yStart,yStop,yStep;

    //Vector 4
    //(i,j) -> (i-1,j)
    int count = 0;
    int CurrentRow = 1;
    xStart = vecSize;
    xStep = -vecSize;
    yStart = 0;
    yStop = SizeY;
    yStep = 0;
    for(int j=0;j<yStop;j++)
    {
        if(CurrentRow%2 != 0)
            //Odd rows are of Size = SizeX
            {
                xStop = SizeX;
                for(int i=xStart;i<xStop;i++)
                {
                    mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                        [j+yStep]][3][vecSize-1]++;
                    count++;
                }
            }
        else //even rows are of Size = SizeX-1
            {
                xStop = SizeX-1;
                for(int i=xStart;i<xStop;i++)
                {
                    mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                        [j+yStep]][3][vecSize-1]++;
                    count++;
                }
            }
        CurrentRow++;
    }
    return(count);
}

```

```

        //End Vector 4
    }

    int DotAng::GetTwoPointVector5(int SizeX, int SizeY,
        int vecSize)
    {
        int xStart,xStop,xStep,yStart,yStop,yStep;

        //Vector 5
        // odd (i,j) -> (i-1,j+1)
        // even (i,j) -> (i,j+1)
        int count = 0;
        //Grid Searching Parameters
        yStart = 0;
        yStop = SizeY-vecSize;
        yStep = vecSize;
        int CurrentRow = yStart+1; //1st row = 1
        if(vecSize%2==0) //even vector size
        {
            xStart = vecSize/2;
            xStep = -vecSize/2;
            for(int j=yStart;j<yStop;j++)
            {
                if(CurrentRow%2 == 0)
                //even rows are of Size = SizeX-1
                {
                    xStop = SizeX-1;
                    for(int i=xStart;i<xStop;i++)
                    {
                        mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                            [j+yStep]][4][vecSize-1]++;
                        count++;
                    }
                    CurrentRow++;
                }
            }
            else //Odd rows are of Size = SizeX
            {
                xStop = SizeX;
                for(int i=xStart;i<xStop;i++)
                {
                    mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                        [j+yStep]][4][vecSize-1]++;
                    count++;
                }
                CurrentRow++;
            }
        }
    }

```

```

        }
    }
}
else //odd vector size
{
    for(int j=yStart;j<yStop;j++)
    {
        if(CurrentRow%2 == 0)
            //even rows are of Size = SizeX-1
            {
                xStart = (vecSize-1)/2;
                xStop = SizeX-1;
                xStep = -(vecSize-1)/2;
                for(int i=xStart;i<xStop;i++)
                {
                    mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                        [j+yStep]][4][vecSize-1]++;
                    count++;
                }
            }
        else //Odd rows are of Size = SizeX
        {
            xStart = (vecSize+1)/2;
            xStop = SizeX;
            xStep = -(vecSize+1)/2;
            for(int i=xStart;i<xStop;i++)
            {
                mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                    [j+yStep]][4][vecSize-1]++;
                count++;
            }
        }
        CurrentRow++;
    }
}
return(count);
//End Vector 5
}

int DotAng::GetTwoPointVector6(int SizeX, int SizeY,
    int vecSize)
{
    int xStart,xStop,xStep,yStart,yStop,yStep;

    //Vector 6

```

```

// odd (i,j) -> (i,j+1)
// even (i,j) -> (i+1,j+1)
int count = 0;
//Grid Searching Parameters
xStart = 0;
yStart = 0;
yStop = SizeY-vecSize;
yStep = vecSize;
int CurrentRow = yStart+1; //1st row = 1
if(vecSize%2==0) //even vector size
{
    for(int j=yStart;j<yStop;j++)
    {
        if(CurrentRow%2 == 0)
            //even rows are of Size = SizeX-1
            {
                xStop = SizeX-vecSize/2-1;
                xStep = vecSize/2;
                for(int i=xStart;i<xStop;i++)
                {
                    mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                        [j+yStep]][5][vecSize-1]++;
                    count++;
                }
                CurrentRow++;
            }
        else //Odd rows are of Size = SizeX
        {
            xStop = SizeX-vecSize/2;
            xStep = vecSize/2;
            for(int i=xStart;i<xStop;i++)
            {
                mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                    [j+yStep]][5][vecSize-1]++;
                count++;
            }
            CurrentRow++;
        }
    }
}
else //odd vector size
{
    for(int j=yStart;j<yStop;j++)
    {
        if(CurrentRow%2 == 0)

```

```

        //even rows are of Size = SizeX-1
        {
            xStop = SizeX-(vecSize+1)/2;
            xStep = (vecSize+1)/2;
            for(int i=xStart;i<xStop;i++)
            {
                mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                    [j+yStep]][5][vecSize-1]++;
                count++;
            }
        }
        else //Odd rows are of Size = SizeX
        {
            xStop = SizeX-(vecSize+1)/2;
            xStep = (vecSize-1)/2;
            for(int i=xStart;i<xStop;i++)
            {
                mf2[mDataGrid[i][j]][mDataGrid[i+xStep]
                    [j+yStep]][5][vecSize-1]++;
                count++;
            }
        }
        CurrentRow++;
    }
}
return(count);
//End Vector 6
}

```

E.5 TPvarianc.hpp

```

//-----
// MODULE: TPvariance.hpp
//
// DESCRIPTION:
//   The is the class TPvariance which calculates the variance
//   of the Taylor factor.
//   This ONLY works for Hexagonal grid type scans where
//   Number of vectors = 6
//
// AUTHOR:
//   Craig P. Przybyla
//

```

```

// REVISION HISTORY:
//     July 1, 2005 - created
//-----

#ifndef TPVARIANCE_H
#define TPVARIANCE_H

//C header Files
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <vector>
#include "dotAng.h"
#include "tokenize.h"

//Define x and y dimensions of Omega
#define OMEGAX 1.2 //in mm
#define OMEGAY 1.2 //in mm

class TPvariance : public DotAng
{
public:
    TPvariance(void);
    ~TPvariance();

    double CalcTPvariance(int LambdaIndex);
    double absvalue(double number);
    void ReadTaylorFactors(std::string TaylorFactorFile);
    void ReadMacroTaylorFactors(std::string filename);
    void WriteVariance(std::string filename);
    int GetNumOfLambda(void);

private:

    double SumOverRSpace(int g1,int g2);
    void Assignf2InRspace(void);
    double Interp(int sextant,double xgrid,double ygrid);
    double CalculateVolumeofOverlap(double rx, double ry);
    bool CheckTFFiles(void);

    //Rspace variables
    double mStepSize; //in microns
    //Step size must be same for every scan
    int mMaxVectorLength;

```

```

//Size of Omega (predefined above)
double mOmegaX; //in microns
double mOmegaY;
double mVolOmega;
int mNumX;
int mNumY;

//two-point array
double ** mf2vecs;

//Rspace grids
double** mf2Rspace;
double* mRgridX;
double* mRgridY;

//Variance
void VarCalcSetup(void);
bool mSetup;
bool mReadMacroTF;
int mNumOfLambda;
std::vector<double> mMacroTF;
unsigned int mNumOfTessera;
std::vector<std::vector<double> > mTaylorFactors;
std::vector<double> mVariance;
};

#endif // TPvariance

```

E.6 TPvariance.cpp

```

//-----
// MODULE: TPvariance.cpp
//
// DESCRIPTION:
//   This file contains all of the functions in order to
//   calculate the Variance of the Taylor Factor
//
// AUTHOR:
//   Craig P. Przybyla
//
// REVISION HISTORY:
//   July 1, 2005 - created

```

```

//-----

//Header File
#include "TPvariance.h"

TPvariance::TPvariance(void)
{
    mSetup = false;
    mReadMacroTF = false;
    mNumOfTessera = GetNumOfTessera();
}

TPvariance::~TPvariance()
{
    //clear pointers
    if(mSetup == true)
    {
        for(int i=0;i<6;i++)
        {
            delete mf2vecs[i];
        }
        delete mf2vecs;

        delete mRgridX;
        delete mRgridY;

        for(int i=0;i<mNumX;i++)
        {
            delete mf2Rspace[i];
        }
        delete mf2Rspace;
    }
}

double TPvariance::absvalue(double number)
{
    if(number<0)
    {
        number=(-1*number);
    }
    return(number);
}

void TPvariance::VarCalcSetup(void)
{

```



```

if(this->mAngFileRead == false)
{
    std::cout << "Error: Must read .ang file
        before variance calculation" << std::endl;
    exit(1);
}

if(this->mCalTwoPointStats == false)
{
    this->CalculateTwoPointStatistics();
}

if(this->mCalODF == false)
{
    this->CalculateODF();
}

if(CheckTFFiles() == false)
{
    std::cout << "Error: Number of lambda for Macro
        Taylor Factors does not match number for Taylor
        Factors Read" << std::endl;
    exit(1);
}

mStepSize = this->GetScanStepSize(0);
//Assume step size is same for every scan
//in microns

//Set up parameters for variance calculation
//Get parameters to set up grid in R(omega) space
mMaxVectorLength = this->GetMaxVectorLength();

//Round Size Omega to nearest stepsize
double remX = fmod(OMEGAX*1e3,mStepSize);
double remY = fmod(OMEGAY*1e3,mStepSize);
mOmegaX = (OMEGAX*1e3-remX); //in microns
mOmegaY = (OMEGAY*1e3-remY);

//Determine grid size in R(omega) space
mNumX = 2*(mOmegaX/mStepSize);
mNumY = 2*(mOmegaY/mStepSize);

//determine reduced volume Omega
mVolOmega = mOmegaX*mOmegaY; //in microns

```

```

//Check to see if f2 is sufficient for Omega
double MaxRadius = sqrt((mOmegaX-mStepSize/2) *
    (mOmegaX-mStepSize/2) + (mOmegaY-mStepSize/2) *
    (mOmegaY-mStepSize/2));
if(MaxRadius > mMaxVectorLength*mStepSize)
{
    std::cout << "Error: Two-point statistics not
        sufficient for defined size Omega" << std::endl;
    exit(1);
}

//Create Array for previously obtained two-point statistics
mf2vecs = new double* [6];
//Only works for hexagonal grids with 6 principal directions
for(int i=0;i<6;i++)
{
    mf2vecs[i] = new double [mMaxVectorLength+1];
}

//Set up position and value grids in R space
mf2Rspace = new double* [mNumX];
mRgridX = new double [mNumX];
mRgridY = new double [mNumY];
for(int i=0;i<mNumX;i++)
{
    mf2Rspace[i] = new double [mNumY];
}

//Set position values
for(int i=0;i<mNumX;i++)
{
    mRgridX[i] = -(mOmegaX-mStepSize/2)+i*mStepSize;
}
for(int j=0;j<mNumY;j++)
{
    mRgridY[j] = -(mOmegaY-mStepSize/2)+j*mStepSize;
}
mSetup = true;
}

double TPvariance::CalcTPvariance(int LambdaIndex)
{
    std::cout << "Calculating variance..." << std::endl;

```

```

    if(mSetup == false)
    {
        this->VarCalcSetup();
    }

    //Calculate Variance
    double Sum = 0.0;
    for(unsigned int i=0;i<mNumOfTessera;i++)
    {
        for(unsigned int j=0;j<mNumOfTessera;j++)
        {
            Sum = Sum + mTaylorFactors[LambdaIndex][i] *
                      mTaylorFactors[LambdaIndex][j] * SumOverRSpace(i,j);
        }
    }

    double tvariance;
    tvariance = (mStepSize*mStepSize)/(mVolOmega*mVolOmega) *
                Sum - mMacroTF[LambdaIndex] * mMacroTF[LambdaIndex];

    mVariance.push_back(tvariance);

    return(tvariance);
}

double TPvariance::SumOverRSpace(int g1,int g2)
{
    double RspaceSum = 0.0;
    //Obtain previous two-point statistics
    int zerocheck = 0;
    for(int i=0;i<6;i++)
    {
        if(g1 == g2)
        {
            mf2vecs[i][0] = this->GetODFPoint(g1);
        }
        else
        {
            mf2vecs[i][0] = 0;
        }
        for(int j=1;j<=mMaxVectorLength;j++)
        //Modified to include zero vector
        {
            double temp = this->GetTwoPointStat(g1,g2,i,(j-1));
            if(temp == 0.0)

```

```

        {
            mf2vecs[i][j] = temp;
        }
        else
        {
            mf2vecs[i][j] = temp;
            zerocheck++;
        }
    }
}
if(zerocheck == 0)
{
    RspaceSum = 0.0;
    return(RspaceSum);
}
else
{
    Assignf2InRspace();
}

for(int i=0;i<mNumX;i++)
{
    for(int j=0;j<mNumY;j++)
    {
        RspaceSum = RspaceSum + mf2Rspace[i][j] *
            CalculateVolumeofOverlap(absvalue(mRgridX[i]),
            absvalue(mRgridY[j]));
    }
}
return(RspaceSum);
}

void TPvariance::Assignf2InRspace(void)
{
    //Set f2 values in grid
    for(int i=0;i<mNumX;i++)
    {
        double x = mRgridX[i];
        for(int j=0;j<mNumY;j++)
        {
            double y = mRgridY[j];
            //Determine in which sextant point x and y lie
            int sextant = -555; //0 through 5

            //1st quadrant

```

```

if(x > 0 && y > 0)
{
    if(y < x*tan(PI/3))
    {
        sextant = 0;
    }
    else
    {
        sextant = 1;
    }
}
//2nd quadrant
else if(x < 0 && y > 0)
{
    if(y < -x*tan(PI/3))
    {
        sextant = 2;
    }
    else
    {
        sextant = 1;
    }
}
//3rd quadrant
else if(x < 0 && y < 0)
{
    if(y > x*tan(PI/3))
    {
        sextant = 3;
    }
    else
    {
        sextant = 4;
    }
}
//4th quadrant
else if(x > 0 && y < 0)
{
    if(y > -x*tan(PI/3))
    {
        sextant = 5;
    }
    else
    {
        sextant = 4;
    }
}

```

```

        }
    }
    mf2Rspace[i][j] = Interp(sextant,x,y);
}
}
return;
}

double TPvariance::Interp(int sextant,double x,double y)
{
    if(sextant == -555)
    {
        std::cout << "Error: Sextant not assigned"
        << std::endl;
        exit(1);
    }

    //find interp points along vectors
    double radius = sqrt(x*x+y*y);
    int point = 1;
    double length = point*mStepSize;
    while(radius > length)
    {
        point++;
        length = point*mStepSize;
    }
    int upperpoint = point; //check values
    int lowerpoint = point-1;
    double theta = atan2(y,x);
    double b = upperpoint*mStepSize-radius;
    double a = mStepSize-b;
    double percenta = a/mStepSize;
    double percentb = b/mStepSize;

    //interp variables
    double f2point;
    int vector1;
    int vector2;
    double deltatheta;
    //sextant 0
    if(sextant == 0)
    {
        vector1 = 0;
        vector2 = 1;
        deltatheta = 0;
    }

```

```

}
//sextant 1
else if(sextant == 1)
{
    vector1 = 1;
    vector2 = 2;
    deltatheta = PI/3;
}
//sextant 2
else if(sextant == 2)
{
    vector1 = 2;
    vector2 = 3;
    deltatheta = 2*PI/3;
}
//sextant 3
else if(sextant == 3)
{
    vector1 = 3;
    vector2 = 4;
    deltatheta = -PI;
}
//sextant 4
else if(sextant == 4)
{
    vector1 = 4;
    vector2 = 5;
    deltatheta = -2*PI/3;
}
//sextant 5
else if(sextant == 5)
{
    vector1 = 5;
    vector2 = 0;
    deltatheta = -PI/3;
}
//Interpolation
double Vec1value = percenta*mf2vecs[vector1][upperpoint] +
    percentb*mf2vecs[vector1][lowerpoint];
double Vec2value = percenta*mf2vecs[vector2][upperpoint] +
    percentb*mf2vecs[vector2][lowerpoint];
//determine arclengths
double ArcVec2 = (theta-deltatheta)*radius;
double ArcVec1 = PI/3*radius-ArcVec2;
double percentArcVec1 = 3*ArcVec2/PI/radius;

```

```

        double percentArcVec2 = 3*ArcVec1/PI/radius;
        f2point = percentArcVec1*Vec1value+percentArcVec2 *
            Vec2value;
        return(f2point);
    }

double TPvariance::CalculateVolumeofOverlap(double rx, double ry)
//volume is in sq microns
{
    double volume = (mOmegaX - rx)*(mOmegaY - ry); //in sq microns
    return(volume);
}

void TPvariance::ReadTaylorFactors(std::string filename)
{
    //open file
    std::ifstream taylorfile(filename.c_str());
    if(taylorfile==NULL)
    //End program if file is not found or if file is empty
    {
        std::cout << "Error: Taylor Factor file not
            found or is empty" << std::endl;
        exit(1);
    }

    std::string data;
    std::vector<std::string> tokens;

    //Read first line
    std::getline(taylorfile, data, '\n');
    tokenize(data,tokens);
    //Set Number of Lambdas
    mNumOfLambda = tokens.size();
    for(int i=0;i<mNumOfLambda;i++)
    {
        double tftemp;
        if(sscanf(tokens[i].c_str(),"%lf",&tftemp)!=1)
        {
            std::cout << "Error: Taylor Factors file data
                is not formatted correctly" << std::endl;
            exit(1);
        }
        std::vector<double> tempTF;
        tempTF.push_back(tftemp);
        mTaylorFactors.push_back(tempTF);
    }
}

```



```

}
//data.clear();
tokens.clear();
while(taylorfile.eof()==0)
{
    std::getline(taylorfile, data, '\n');
    tokenize(data,tokens);
    if(tokens.size() != mNumOfLambda)
    {
        if(data.size() == 0)
        {
            continue;
        }
        else
        {
            std::cout << "Error: Taylor Factors file
                formatted incorrectly" << std::endl;
            exit(1);
        }
    }
    for(int i=0;i<mNumOfLambda;i++)
    {
        double tftemp;
        if(sscanf(tokens[i].c_str(),"%lf",&tftemp)!=1)
        {
            std::cout << "Error: Taylor Factors file data
                is not formatted correctly" << std::endl;
            exit(1);
        }
        mTaylorFactors[i].push_back(tftemp);
    }
    tokens.clear();
}
if(mTaylorFactors[0].size() < mNumOfTessera)
{
    std::cout << "Error: Insufficient number of Taylor Factors
        read for number of tessera" << std::endl;
    exit(1);
}
}

void TPvariance::ReadMacroTaylorFactors(std::string filename)
{
    //open file
    std::ifstream taylorfile(filename.c_str());

```

```

    if(taylorfile==NULL)
    //End program if file is not found or if file is empty
    {
        std::cout << "Error: Macro Taylor Factor file not
            found or is empty" << std::endl;
        exit(1);
    }

    while(taylorfile.eof()==0)
    {
        std::string data;
        double macroTaylorFactor;
        //Read data
        std::getline(taylorfile, data, '\n');
        if(sscanf(data.c_str(), "%lf", &macroTaylorFactor) != 1)
        {
            continue;
        }

        mMacroTF.push_back(macroTaylorFactor);
    }
    mReadMacroTF = true;
}

void TPvariance::WriteVariance(std::string filename)
{
    //open file
    std::ofstream variancefile(filename.c_str());
    if(variancefile==NULL)
    //End program if file is not found or if file is empty
    {
        std::cout << "Error: variance file could not be
            opened" << std::endl;
        exit(1);
    }

    for(unsigned int i=0; i<mMacroTF.size(); i++)
    {
        variancefile << mVariance[i] << std::endl;
    }
    variancefile.close();
}

bool TPvariance::CheckTFFiles(void)
{

```

```

        if(mMacroTF.size() == mTaylorFactors.size())
        {
            return(true);
        }
        else
        {
            return(false);
        }
    }

int TPvariance::GetNumOfLambda(void)
{
    return(mNumOfLambda);
}

```

E.7 tokenize.hpp

```

//-----
// MODULE: tokenize.hpp
//
// Description : This is the header for tokenize.cpp
//
// Author : Craig Przybyla
//
// Date: September 14, 2004
//-----
#ifndef TOKENIZE_HPP
#define TOKENIZE_HPP

#include <string>
#include <vector>
#include <algorithm>
#include <iostream>

//function Prototypes
void tokenize(const std::string& str,
             std::vector<std::string>& tokens);

#endif //TOKENIZE_HPP

```

E.8 tokenize.cpp

```

//-----
// MODULE: tokenize.cpp
//
// Description : This function shows how to take a standard C++
// string and parse it into tokens specified by a set of delimiters.
// This method is preferred over using the C version of strtok()
// on the c_str() data in the standard string.
//
// This string tokenizer and example were extracted from
//   http://www.tldp.org/HOWTO/C++Programming-HOWTO-7.html
// Author : Al Dev (Alavoor Vasudevan)
//
//-----

// Header Files
#include "tokenize.h"

void tokenize(const std::string& str,
              std::vector<std::string>& tokens)
{
    // define the list of delimiters
    std::string delimiters("\t");

    // Skip delimiters at beginning.
    std::string::size_type lastPos =
        str.find_first_not_of(delimiters, 0);

    // Find first "non-delimiter".
    std::string::size_type pos =
        str.find_first_of(delimiters, lastPos);

    while (std::string::npos != pos ||
           std::string::npos != lastPos)
    {
        // Found a token, add it to the vector.
        tokens.push_back(str.substr(lastPos, pos - lastPos));

        // Skip delimiters. Note the "not_of"
        lastPos = str.find_first_not_of(delimiters, pos);

        // Find next "non-delimiter"
        pos = str.find_first_of(delimiters, lastPos);
    }
}

```


Appendix F

Matlab Taylor Factor Code

F.1 TaylorFactor.m

```
%Author: Craig Przybyla
%Date: June 2005

clc;clear all;close all;

%savefile Name
savefile = 'TF1728ChinMamv2.mat';
%PreSet Variables
tau knot = 1;
RateSensitivity = 65; %max power of n
StrainRateMag = 1; %strain-rate magnitude

%%%%%%%% File Parameters

%Load .ang file
load('LoafTessCn1728.txt');
phi1(1:length(LoafTessCn1728)) =
    LoafTessCn1728(1:length(LoafTessCn1728),2);
PHI(1:length(LoafTessCn1728)) =
    LoafTessCn1728(1:length(LoafTessCn1728),3);
phi2(1:length(LoafTessCn1728)) =
    LoafTessCn1728(1:length(LoafTessCn1728),4);
clear('LoafTessCn1728');

%Set up Lambda Array
deltalambda = 0.05;
NumLambda = 1/deltalambda-1;
```

```

for i=1:NumLambda
    lambda(i) = i*deltalambda;
end

%Read in Slip Systems and create slip tensors
[NumOfSys,slipsys] = GetContractedSlipTensors();

%Iterate through lambda
for i=1:length(lambda)
    %Strain rate tensor in macroscopic frame
    epsilondot(1:3,1:3) = 0.0; %Initialize array
    epsilondot(1,1) = StrainRateMag;
    epsilondot(2,2) = -lambda(i)*StrainRateMag;
    epsilondot(3,3) = (lambda(i)-1)*StrainRateMag;

    %Set up waitbar
    h = waitbar(0,['Calculating Taylor Factors for lambda',num2str(i),' ...']);
    for j=1:length(phi1)
        %g defined as direction cosines as in Bunge
        g(1,1)=cos(phi1(j))*cos(phi2(j)) -
            sin(phi1(j))*sin(phi2(j))*cos(PHI(j));
        g(1,2)=sin(phi1(j))*cos(phi2(j)) +
            cos(phi1(j))*sin(phi2(j))*cos(PHI(j));
        g(1,3)=sin(phi2(j))*sin(PHI(j));
        g(2,1)=-cos(phi1(j))*sin(phi2(j)) -
            sin(phi1(j))*cos(phi2(j))*cos(PHI(j));
        g(2,2)=-sin(phi1(j))*sin(phi2(j)) +
            cos(phi1(j))*cos(phi2(j))*cos(PHI(j));
        g(2,3)=cos(phi2(j))*sin(PHI(j));
        g(3,1)=sin(phi1(j))*sin(PHI(j));
        g(3,2)=-cos(phi1(j))*sin(PHI(j));
        g(3,3)=cos(PHI(j));
        %rotate strain rate into crystal frame
        epsilondotC = g*epsilondot*g';

        %Determine stress tensor
        [sigmaC] = StrainToStress(epsilondotC,RateSensitivity,
            tauknot,NumOfSys,slipsys,lambda(i),j);

        %Determine stress (sigma) in macroscopic frame
        sigma = g'*sigmaC*g;

        %calculate Taylor Factor

```

```

        [TF(i,j)] = GetChinMammelTaylorFactor(StrainRateMag,g,
            sigma,RateSensitivity,tauknot);

        %increment waitbar
        waitbar(j/length(phi1),h)
    end
    close(h);
    %save info
    save(savefile,'TF');
end

```

F.2 GetSlipTensors.m

```

%The data file contains the slip systems for the BCC
%crystal structure. The file is formatted such that each line
%contains n1 n2 n3 m1 m2 m3 where n is the normal to the slip
%plane and m is the slip direction
%Author: Craig Przybyla
%Date: May 2004
%systems: 12

%n = direction normal to slip plane
%m = slip direction

function [NumOfSys,slipsys] = GetSlipTensors()

NumOfSys = 12;

load('bccSlipSystems.txt', 'n1', 'n2', 'n3', 'm1', 'm2', 'm3');
for sys=1:NumOfSys
    n(1:3) = bccSlipSystems(sys,1:3);
    m(1:3) = bccSlipSystems(sys,4:6);
    mag = sqrt(n(1)^2+n(2)^2+n(3)^2)*sqrt(m(1)^2+m(2)^2+m(3)^2);
    for i=1:3
        for j=1:3
            slipsys(sys).mu(i,j) = 0.5/mag*(m(i)*n(j)+m(j)*n(i));
        end
    end
end
end

```

F.3 GetContractedSlipTensors.m


```

%The data file contains the slip systems for the BCC crystal
%structure. The file is formatted such that each line contains
%n1 n2 n3 m1 m2 m3 where n is the normal to the slip plane and
% m is the slip direction
%Author: Craig Przybyla
%Date: May 2004
%systems: 12

%n = direction normal to slip plane
%m = slip direction

function [NumOfSys,slipsys] = GetContractedSlipTensors()

NumOfSys = 12;

load('bccSlipSystems.txt', 'n1', 'n2', 'n3', 'm1', 'm2', 'm3');
for sys=1:NumOfSys
    n(1:3) = bccSlipSystems(sys,1:3);
    m(1:3) = bccSlipSystems(sys,4:6);
    mag = sqrt(n(1)^2+n(2)^2+n(3)^2)*sqrt(m(1)^2+m(2)^2+m(3)^2);
    for p=1:6
        if p == 1
            i=1;
            j=1;
        elseif p == 2
            i=2;
            j=2;
        elseif p == 3
            i=3;
            j=3;
        elseif p == 4
            i=2;
            j=3;
        elseif p == 5
            i=3;
            j=1;
        elseif p == 6
            i=1;
            j=2;
        end
        slipsys(sys).mur(p) = 0.5/mag*(m(i)*n(j)+m(j)*n(i));
    end
end
end

```

F.4 StrainToStress.m

%Author: Craig Przybyla

%Date: June 2005

```
function [sigmaC] = StrainToStress(epsilondotC,RateSensitivity,
    tauknot,NumOfSys,slipsys,lambda,point)
%Constants for Power Law relation
alpha = 1;
dn = 1/10; %Step in n for Newton's Method
tolerance = 1.0e-12;
convergelimit = 500;

%Contract Epsilon into a 5-Dimentional Column Vector
eC(1) = epsilondotC(1,1);
eC(2) = epsilondotC(2,2);
eC(3) = epsilondotC(2,3);
eC(4) = epsilondotC(3,1);
eC(5) = epsilondotC(1,2);

%%NEWTONS METHOD TO SOLVE FOR STRESS%%

%sigma @ n=1
%Creep Compliance Tensor
M(1:6,1:6) = 0;
for k=1:NumOfSys
    for i=1:6
        for j=1:6
            slipsys(k).M(i,j) = slipsys(k).mur(i)*slipsys(k).mur(j);
        end
    end
    M = M + slipsys(k).M;
end
M=(alpha/tauknot)*M;

%Reduce Creep Compliance Tensor to 5x5
Mr = ReduceCreepTo5D(M);

sC = Mr\transpose(eC); %C denotes Contracted Tensor in crystal frame
%Solving the problem Ax=B by x = A\B uses a particular function in
%Matlab that solves this using Gaussian elimination instead of
%calculating the inverse such as x = inv(A)*B

%Iterative Newton Method to find stress
```

```

n = 1 + dn;
while n < RateSensitivity
    exit = 1;
    converge = 1;
    while exit == 1
        %%%Calculate Updated Creep Compliance Tensor%%%
        %Calculate Resolved Shear Stress for Each Slip System
        for k=1:NumOfSys
            slipsys(k).tauRSS = sC(1)*(slipsys(k).mur(1)-...
                slipsys(k).mur(3))+sC(2)*(slipsys(k).mur(2)-...
                slipsys(k).mur(3))+sC(3)*slipsys(k).mur(4)+...
                sC(4)*slipsys(k).mur(5)+sC(5)*slipsys(k).mur(6);
        end

        %Creep Compliance Tensor
        M(1:6,1:6) = 0; %Initialize Creep Compliance
        for k=1:NumOfSys
            M = M + abs(slipsys(k).tauRSS)^(n-1)*slipsys(k).M;
        end
        M = (alpha/tauknot^n)*M;
        Mr = ReduceCreepTo5D(M); %Reduce M to 5x5

        %Determine Stress Correction
        ndeltaSigmaTemp = Mr\transpose(eC);
        deltaSigma = (1/n)*(ndeltaSigmaTemp-sC);
        sC=sC+deltaSigma;
        if max(abs(deltaSigma)) < tolerance
            exit = 0;
            break;
        end
        converge = converge+1;
        if converge > convergelimt
            disp('Did not converge');
            disp(['lambda: ',num2str(lambda),' n: ',num2str(n)]);
            disp(['Point: ',num2str(point)]);
            break;
        end
    end
    n = n + dn; %Step in n = dn where dn is specified above
    if converge > convergelimt
        break;
    end
end

%stress matrix

```

```

sigmaC(1,1) = sC(1);
sigmaC(1,2) = sC(5)/2;
sigmaC(1,3) = sC(4)/2;
sigmaC(2,1) = sC(5)/2;
sigmaC(2,2) = sC(2);
sigmaC(2,3) = sC(3)/2;
sigmaC(3,1) = sC(4)/2;
sigmaC(3,2) = sC(3)/2;
sigmaC(3,3) = -(sC(1)+sC(2));

```

F.5 ReduceCreep5D.m

```

%Author: Craig Przybyla

function Mr = ReduceCreep5D(M)

%Reduce Creep Compliance Tensor to 5x5
for m=1:2
    for n=1:2
        Mr(m,n) = M(m,n) - M(m,3);
    end
end
for m=3:5
    for n=1:2
        Mr(m,n) = M(m+1,n) - M(m+1,3);
    end
end
for m=1:2
    for n=3:5
        Mr(m,n) = M(m,n+1);
    end
end
for m=3:5
    for n=3:5
        Mr(m,n) = M(m+1,n+1);
    end
end
end

```

F.6 ChinMammelTaylorFactor.m

```

%Author: Craig Przybyla

function [ChinMammelTaylorFactor] =

```

```

    GetChinMammelTaylorFactor(StrainRateMag,g,sigma,n,tauknot)

%Read in Slip Systems and create slip tensors
[NumOfSys,slipsys] = GetSlipTensors();

%Rotate Slip tensors into Sample frame
for k=1:NumOfSys
    slipsys(k).muSF = g'*slipsys(k).mu*g;
end

%Construction of Shear Stresses
for k=1:NumOfSys
    slipsys(k).tauRSS = 0.0;
    for i=1:3
        for j=1:3
            slipsys(k).tauRSS = slipsys(k).tauRSS +
                sigma(i,j)*slipsys(k).muSF(i,j);
        end
    end
end

gamma = 0.0;
for k=1:NumOfSys
    gamma = gamma + abs((slipsys(k).tauRSS/tauknot)^(n));
end
ChinMammelTaylorFactor = gamma/StrainRateMag;

% %%%
% %Test taylor factor
% gamma = 0.0;
% for k=1:NumOfSys
%     k
%     gamma = gamma + abs(slipsys(k).tauRSS/tauknot)^(n-1);
%     a = slipsys(k).tauRSS
% end
% Mtest = gamma/StrainRateMag
% %%%

```

Appendix G

Matlab Taylor Factor Variance Random Sampling Code

G.1 RandVariance.m

```
%Author: Craig Przybyla

clc; close all; clear all;

%Define Omega in millimeters
OmegaX = 0.8; %in mm
OmegaY = 0.8; %in mm
%Define Samples Size
SampleSize = 500;

%Read in Scans -- Step Size must all be the same
%Load .ang file Scan1b
load('FeSiScan1b.txt');
ang(1).phi1(1:length(FeSiScan1b)) =
    FeSiScan1b(1:length(FeSiScan1b),1);
ang(1).PHI(1:length(FeSiScan1b)) =
    FeSiScan1b(1:length(FeSiScan1b),2);
ang(1).phi2(1:length(FeSiScan1b)) =
    FeSiScan1b(1:length(FeSiScan1b),3);
ang(1).x(1:length(FeSiScan1b)) =
    FeSiScan1b(1:length(FeSiScan1b),4);
ang(1).y(1:length(FeSiScan1b)) =
    FeSiScan1b(1:length(FeSiScan1b),5);
%Determine StepSize
StepSize = (ang(1).x(2)-ang(1).x(1));
delx = StepSize;
dely = StepSize/2*tan(pi/3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

ang(1).length = length(FeSiScan1b);
for i=1:length(FeSiScan1b)-1
    x = ang(1).x(i);
    if(x > ang(1).x(i+1))
        ang(1).SizeX = x/delx;
        break;
    end
end
ang(1).SizeY = ang(1).y(length(FeSiScan1b))/dely;
clear('FeSiScan1b');
load('TFScan1bChinMamfv.mat');
ang(1).TF = TF';
clear('TF');

%Load .ang file FeSiScan2
load('FeSiScan2.txt');
ang(2).phi1(1:length(FeSiScan2)) =
    FeSiScan2(1:length(FeSiScan2),1);
ang(2).PHI(1:length(FeSiScan2)) =
    FeSiScan2(1:length(FeSiScan2),2);
ang(2).phi2(1:length(FeSiScan2)) =
    FeSiScan2(1:length(FeSiScan2),3);
ang(2).x(1:length(FeSiScan2)) =
    FeSiScan2(1:length(FeSiScan2),4);
ang(2).y(1:length(FeSiScan2)) =
    FeSiScan2(1:length(FeSiScan2),5);
ang(4).length = length(FeSiScan2);
for i=1:length(FeSiScan2)-1
    x = ang(2).x(i);
    if(x > ang(2).x(i+1))
        ang(2).SizeX = x/delx;
        break;
    end
end
ang(2).SizeY = ang(2).y(length(FeSiScan2))/dely;
clear('FeSiScan2');
load('TFScan2ChinMamfv.mat');
ang(2).TF = TF';
clear('TF');

%Load .ang file FeSiScan3
load('FeSiScan3.txt');
ang(3).phi1(1:length(FeSiScan3)) =
    FeSiScan3(1:length(FeSiScan3),1);
ang(3).PHI(1:length(FeSiScan3)) =

```

```

    FeSiScan3(1:length(FeSiScan3),2);
ang(3).phi2(1:length(FeSiScan3)) =
    FeSiScan3(1:length(FeSiScan3),3);
ang(3).x(1:length(FeSiScan3)) =
    FeSiScan3(1:length(FeSiScan3),4);
ang(3).y(1:length(FeSiScan3)) =
    FeSiScan3(1:length(FeSiScan3),5);
ang(4).length = length(FeSiScan3);
for i=1:length(FeSiScan3)-1
    x = ang(3).x(i);
    if(x > ang(3).x(i+1))
        ang(3).SizeX = x/delx;
        break;
    end
end
ang(3).SizeY = ang(3).y(length(FeSiScan3))/dely;
clear('FeSiScan3');
load('TFScan3ChinMamfv.mat');
ang(3).TF = TF';
clear('TF');

%Load .ang file FeSiScan4
load('FeSiScan4.txt');
ang(4).phi1(1:length(FeSiScan4)) =
    FeSiScan4(1:length(FeSiScan4),1);
ang(4).PHI(1:length(FeSiScan4)) =
    FeSiScan4(1:length(FeSiScan4),2);
ang(4).phi2(1:length(FeSiScan4)) =
    FeSiScan4(1:length(FeSiScan4),3);
ang(4).x(1:length(FeSiScan4)) =
    FeSiScan4(1:length(FeSiScan4),4);
ang(4).y(1:length(FeSiScan4)) =
    FeSiScan4(1:length(FeSiScan4),5);
ang(4).length = length(FeSiScan4);
for i=1:length(FeSiScan4)-1
    x = ang(4).x(i);
    if(x > ang(4).x(i+1))
        ang(4).SizeX = x/delx;
        break;
    end
end
ang(4).SizeY = ang(4).y(length(FeSiScan4))/dely;
clear('FeSiScan4');
load('TFScan4ChinMamfv.mat');
ang(4).TF = TF';

```



```

clear('TF');
clear('x');

%Resize Omega to be a multiple of StepSize
OmegaX = OmegaX*1e3 - mod(OmegaX*1e3,dely); %convert to microns
OmegaY = OmegaY*1e3 - mod(OmegaY*1e3,dely); %convert to microns
NumOmegaX = OmegaX/dely;
NumOmegaY = OmegaY/dely;

%Calculate Variance
for lambda=1:19
    for sample=1:SampleSize
        scan = ceil((4*rand(1))); %Randomly Select Scan
        StartX = ceil(((ang(scan).SizeX-NumOmegaX)*rand(1)));
        StartY = ceil(((ang(scan).SizeY-NumOmegaY)*rand(1)));
        if(mod(StartY,2) == 0)
            index = (ang(scan).SizeX-1)*(StartY/2-1) +
                    ang(scan).SizeX*(StartY/2) + StartX;
            currentRow = 2;
        else
            index = ((ang(scan).SizeX-1)*(StartY-1)/2 +
                    ang(scan).SizeX*(StartY-1)/2) + StartX;
            currentRow = 1;
        end
        count = 1;
        for j=1:NumOmegaY
            if(mod(currentRow,2) == 0)
                xStop = NumOmegaX-1;
            else
                xStop = NumOmegaX;
            end
            for i=1:xStop
                Omega(count) = ang(scan).TF(index,lambda);
                index = index + 1;
                count = count + 1;
            end
            currentRow = currentRow + 1;
            index = index + ang(scan).SizeX-NumOmegaX;
        end
        SampleMeans(sample) = mean(Omega);
    end
    stdev(lambda) = std(SampleMeans)
    clear('SampleMeans');
    variance(lambda) = stdev(lambda)^2
end

```

Bibliography

- Adams, B. L. (1993). Orientation imaging microscopy: application to the measurement of grain boundary structure. *Materials Science and Engineering A166*, 59–66.
- Adams, B. L., E. Homer, and J. Basinger (2003, November). Consideration of microstructure variability in the grain boundary character distribution. In *Materials Science And Technology*, Chicago, Illinois. TMS.
- Adams, B. L., S. I. Wright, and K. Kunze (1993, April). Orientation imaging: The emergence of a new microscopy. *Metallurgical Transactions A 24A*, 819–831.
- Akpokodje, E. G. and P. P. Hudec (1992, February). Factors controlling properties and durability of concretionary laterite gravel aggregates. *Journal of Materials in Civil Engineering 4*(1), 58–70.
- Bishop, J. F. W. and R. Hill (1951a). *Philosophical Magazine 42*, 414–427.
- Bishop, J. F. W. and R. Hill (1951b). *Philosophical Magazine 42*, 1298–1307.
- Bunge, H. (1982). Mean values of physical properties. *Quantitative Texture Analysis*, 383–406.
- Bunge, H. J. (1993). *Texture Analysis in Materials Science* (English ed.). Cuvillier Verlag, Göttingen.
- Chanat, S. and D. Zipperian (1992, August). Microstructural preparation and analysis of advanced composites. In *Metallographic Characterization of Metals After*

- Welding, Processing, and Service*, Denver, Colorado, pp. 547–558. ASM International.
- Cheng, X. and A. M. Sastry (1999). On transport in stochastic, heterogeneous fibrous domains. *Mechanics of Materials* 31, 765–786.
- Chin, G. Y. and W. L. Mammel (1967, September). Computer solutions of the taylor analysis for axisymmetric flow. *Transactions of the Metallurgical Society of AIME* 239, 1400–1405.
- Chin, G. Y. and W. L. Mammel (1969). Generalization and equivalence of the minimum work (taylor) and maximum work (bishop-hill) principles for crystal plasticity. *Transactions of Metallurgical Society AIME* 245, 1211–1214.
- Choi, S. H. and F. Barlat (1999). Prediction of macroscopic anisotropy in rolled aluminum-lithium sheet. *Scripta Materialia* 41(9), 981–987.
- Hosford, W. F. (2005). *Mechanical Behavior of Materials*. Cambridge.
- Hutchinson, J. W. (1976, February). Bounds and self-consistent estimates for creep of polycrystalline materials. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* 348(1652), 101–127.
- J. L. Duncan, E. C. and W. Johnson (1967). Measurement of normal plastic anisotropy in sheet metal. *Journal of the Australian Institute of Metals* 12, 127–138.
- Jesson, T. L., V. A. Greenhut, L. David, III, and J. J. Friel (1999, October). Effect of microstructure on the mechanical behavior of continuous-fiber-reinforced ceramic-matrix composites. *Journal of the American Ceramic Society* 82(10), 2753–2761.
- Lee, P. S., H. R. Piehler, A. D. Rollett, and B. L. Adams (2002, December). Texture clustering and long-rang disorientation representation methods: Application to

- 6022 aluminium sheet. *Metallurgical and Materials Transactions 33A*, 3709–3718.
- Liu, Y. C. (1983, June). On the r-value measurements. *Metallurgical Transactions A 14A*, 1199–1205.
- Lu, B. L. and S. Torquato (1990). Local volume fraction fluctuations in heterogeneous media. *Journal of Chemical Physics 93*, 3453–3459.
- Mason, T. A. and B. L. Adams (1999, April). Use of microstructural statistics in predicting polycrystalline material properties. *Metallurgical and Materials Transactions 30A*, 969–979.
- Morawiec, A. (2003). *Orientations and Rotations: Computations in Crystallographic Textures*. Springer.
- Przybyla, C. P., B. L. Adams, and M. P. Miles (2004, June). A method for determining property variance in polycrystalline materials. In S. Ghosh, J. M. Castro, and J. K. Lee (Eds.), *Materials Processing and Design: Modeling, Simulation and Applications*, Columbus, Ohio, pp. 1760–1764. Numiform.
- Quintanilla, J. and S. Torquato (1998). Local volume fraction fluctuations in random media. *Journal of Chemical Physics 110*, 3215–3219.
- Skrotzki, B., E. A. Starke, Jr., and G. J. Shiflet (1994, September). Effect of texture and precipitates on mechanical property anisotropy of al-cu-mg-x alloys. In *The 4th International Conference on Aluminium Alloys: Their Physical and Mechanical Properties*, Volume II, pp. 40–47.
- Taylor, G. I. (1938). Plastic strain in metals. *Journal of the Institute of Metals* (62), 307–324.
- Torquato, S. (2002). *Random Heterogeneous Materials*. New York: Springer-Verlag.

- W. T. Lankford, S. C. S. and J. A. Bauscher (1950). New criteria for predicting the press performance of deep drawing sheets. *Transactions of ASM* 42, 1197–1232.
- Welch, P. I. and H. J. Bunge (1986, April). Influence of grain size on plastic anisotropy in low-carbon steels. *Materials Science and Technology* 2(4), 354–362.
- Wright, S. I. and D. P. Field (1998, November). Recent studies of local texture and its influence on failure. *Materials Science & Engineering A: Structural Materials: Properties, Microstructure and Processing* A257(1), 165–170.
- X. Gao, C. P. Przybyla, B. L. A. (2005, September). Methodology for recovering and analyzing two-point pair correlations functions in polycrystalline materials. *Metallurgical Transactions A*. Submitted.
- Yadov, D. and N. Verma (1997, March-April). Buckling of composite circular cylindrical shells with random material properties. *Composite Structures* 37(3-4), 385–391.