12-9-2014

Dynamical X-Ray Diffraction from Arbitrary Semiconductor Heterostructures Containing Dislocations

Paul Rago
Phonon Corporation, paul.rago@gmail.com

Dynamical X-Ray Diffraction from Arbitrary Semiconductor Heterostructures Containing Dislocations

Paul B. Rago

B.S., University of Connecticut, 2008

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Connecticut

2014

Masters of Science Thesis

Dynamical X-Ray Diffraction from Arbitrary Semiconductor Heterostructures Containing Dislocations

Presented by

Paul B. Rago, B.S.

Major Advisor_____

John E. Ayers

Associate Advisor_____

Rajeev Bansal

Associate Advisor_____

Faquir Jain

University of Connecticut

2014

Table of Contents

Abstract

High-resolution x-ray diffraction is a valuable non-destructive tool for the structural characterization of semiconductor heterostructures, and measured diffraction profiles contain information on the depth profiles of strain, composition, and defect densities in device structures. Much of this information goes untapped because the lack of phase information prevents direct inversion of the diffraction profile. A common practice is to use dynamical simulations in conjunction with a curve-fitting procedure to extract the profiles of strain and composition in the depth of the material. Prior to this work, the dynamical simulations have been based on perfect, dislocation-free laminar crystals, and this renders the analysis inapplicable to many real structures which contain dislocation densities greater than about 10^6 cm^{-2}. In this work we present two novel dynamical models for Bragg x-ray diffraction in semiconductor crystals with arbitrary, nonuniform composition, strain, and dislocation density: the Phase Invariant Model for Dynamical Diffraction (PIDDM) and the Mosaic Crystal Model for Dynamical Diffraction (MCDDM), which improves upon the former with greater accuracy and better computation times. The framework for dynamical diffraction calculations is based on the Takagi-Taupin equation, but modified for distorted crystals by accounting for the angular and strain broadening introduced to the lattice by the presence of dislocations.

In this thesis we cover dynamical diffraction from perfect crystals then provide a description of both PIDDM and MCDDM models. We present results for each of these using the $Si_{1-x}Ge_x$ / Si (001), ZnS_ySe_{1-y} / GaAs (001), and $In_xGa_{1-x}As$ / GaAs (001) material systems in multilayer, superlattice, step-graded, and linearly-graded heterostructures. We also compare the MCDDM to experimental measurements of an $In_xAl_{1-x}As$ / GaAs (001) metamorphic device structure. Lastly we provide a comparison of the two models with $In_xGa_{1-x}As$ heterostructures. We show that these two models for defected crystals extend the usefulness of the x-ray diffraction characterization method and should in principle allow depth profiling of dislocation densities in arbitrary heterostructures.

Introduction

X-Ray Diffraction for Characterization of Semiconductor Crystals

High-resolution x-ray diffraction (HRXRD) is an important nondestructive tool for characterization of crystalline materials. XRD measurements can be used to estimate lattice constants, strains, composition, crystallographic orientation, and defect densities in device structures. Furthermore, x-ray diffraction profiles reveal the depth profiles of these material properties. That is, for a heterostructure containing numerous layers, XRD experiments can yield these properties for any and all of the layers in the structure.

Depicted in Figure 1 is a typical high-resolution x-ray diffractometer. The x-ray source yields a broad spectrum of wavelengths and a divergent beam. This beam is usually conditioned using a Bartels-type monochromator[1], which yields a small beam divergence and is tuned to pass only a narrow spectral line of radiation[2]. In the illustration, a diffraction pattern is seen to arise after the radiation contacts the sample. To measure this diffraction profile, the rocking angle θ is simply changed whilst measuring diffracted radiation intensity with the x-ray. This measurement is known as the "θ-scan", and the resulting plot of intensity as a function of rocking angle, is known as an x-ray diffraction profile or rocking curve[†]. This measurement yields a peak in intensity corresponding to each layer of the sample that contains a different lattice parameter, and the positions, intensities, and widths of these peaks can be analyzed to obtain much information about the crystallographic structure of the specimen. However, Halliwell et al.[3] showed that there is not necessarily a one-to-one correspondance between peaks in the diffraction profile and lattice constants in the sample, and it is for this reason that dynamical simulations are used in conjunction with experimental measurements.

[†] Another related measurement, which involves modifying both the sample angle and detector angle, is known as the ω-2θ scan. Though all the data presented herein is of the θ-scan, it is a simple adaptation of this work to obtain ω-2θ-type diffraction profiles.

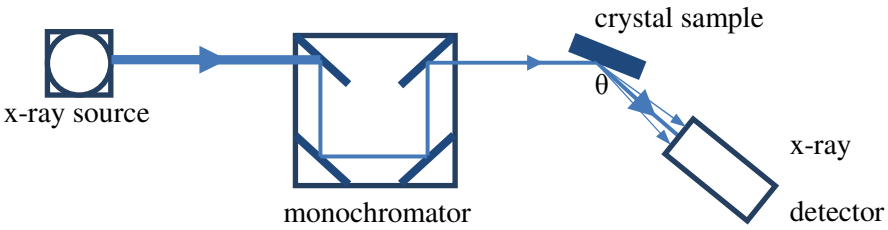

Figure 1. An x-ray diffractometer

Bragg's Law

The basic principle of diffraction from a crystal is described by Bragg's Law[4]. When x-ray radiation contacts an atom, the atom scatters this energy in a process known as x-ray scattering. If the wavelength of the radiation is of a comparable length to the spacing of the planes of atoms in the lattice, interference patterns will result. These patterns contain peaks (and nulls), corresponding to the Bragg condition for constructive (and destructive) interference. Specifically, if the difference in the path length the x-ray travels for beams scattered from different atoms is an integer multiple of the source wavelength, the Bragg condition for diffraction is satisfied. The geometry for this configuration in a few planes of atoms in a simplified crystal is given in Figure 2.

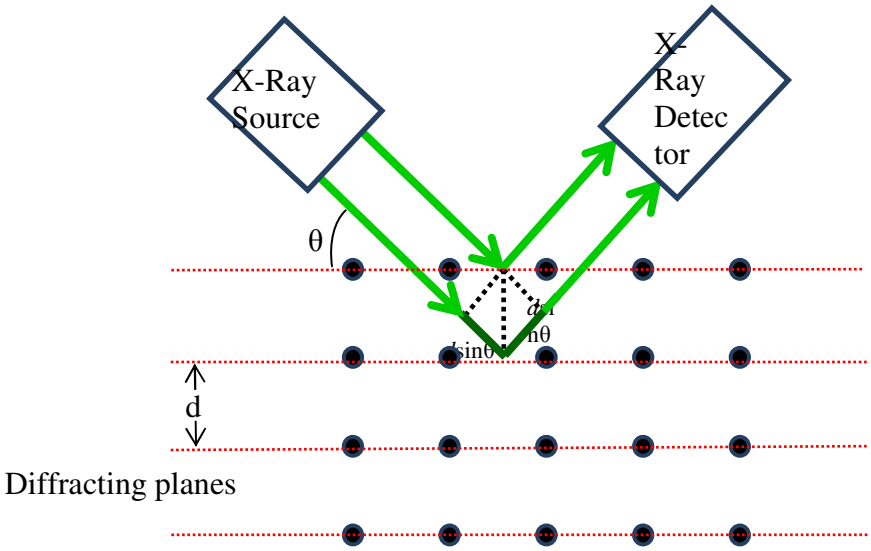

Figure 2. Geometry of Bragg diffraction from a crystal

When the x-ray beam is incident upon the crystal, it contacts the first plane plane of atoms, reflecting back up to the detector, but much of the energy is also transmitted deeper into the crystal. The transmitted energy then contacts the next plane of atoms, and the same things happens: reflection and transmission. (Pictured here, for simplicity, only the first plane of atoms transmits. In reality, transmission continues well into the extinction depth, which is typically tens of microns, or the entirety of epitaxial layers.) When the beam reflecting off the second plane reaches the surface it superimposes as it joins the beam reflected off the first plane. In the geometry shown, the extra length the beam must travel for each additional plane is equal to $2d\sin\theta$. When this extra length is equal to an integer multiple of the wavelenth of the x-ray source λ, constructive interference results. This condition gives the well-known Bragg equation, given by[4]

$$2d\sin\theta_B = n\lambda. \tag{1}$$

Reflections and Symmetry

There are many hkl reflections from a cubic crystal. These reflections correspond to different angles of diffracting planes in the lattice, and they have different properties. For example, some eccentric asymmetric reflections have a weaker intensity but a greater sensitivity to out-of-plane strain.

For a cubic or zincblende crystal with a lattice parameter a, the spacing of the (hkl) planes is given by

$$d(hkl) = \frac{a}{\sqrt{h^2 + k^2 + l^2}} \tag{2}$$

Relating (1) and (2), we find that the Bragg angle of the *hkl* reflection is

$$\theta_B(hkl) = \sin^{-1}\left(n\lambda \cdot \frac{\sqrt{h^2 + k^2 + l^2}}{2a}\right). \tag{3}$$

A reflection can be said to be symmetric if the diffracting planes are parallel to the surface of the crystal. For (001) orientation, this happens when $h = k = 0$. A symmetric reflection is also characterized by the beam's angle of incidence and angle of diffraction being equal to each other (when the Bragg condition is satisfied). This configuration is illustrated in Figure 3.

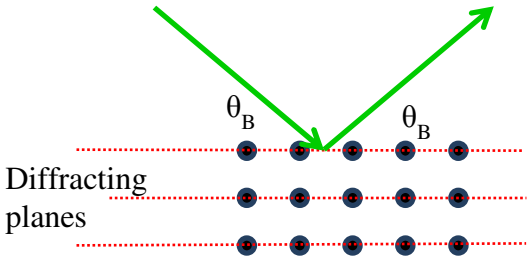

Figure 3. A symmetric reflection.

In an asymmetric reflection, on the other hand, the diffracting planes are inclined to the crystal surface by an angle φ, as illustrated in Figure 4. Common asymmetric reflections include the 135, 044, 026, and others.

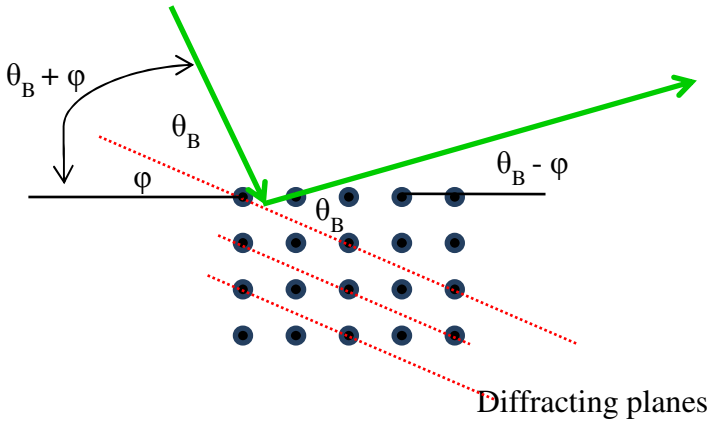

Figure 4. An asymmetric reflection

In this case, the angle between the diffracting planes and the surface is given by

$$\varphi(h,k,l) = \cos^{-1}\left(\frac{l}{\sqrt{h^2+k^2+l^2}}\right). \tag{4}$$

In the presence of biaxial strain, the lattice undergoes a distortion that affects the distance between the diffracting planes, and consequently affects the angle of the diffracting planes. The change in φ due to tetragonal distortion is given by

$$\Delta\varphi_{tet}(h,k,l) = \cos^{-1}\left(\frac{l/c}{c\sqrt{\left(\frac{h}{a}\right)^2+\left(\frac{k}{a}\right)^2+\left(\frac{l}{c}\right)^2}}\right) - \cos^{-1}\left(\frac{l}{c\sqrt{h^2+k^2+l^2}}\right), \tag{5}$$

where a and c are the in- and out-of-plane lattice constants, respectively.

Dynamical Diffraction from Perfect Crystals

To extend the usefulness of x-ray diffraction studies, dynamical[3, 5-9] and kinematical[10-14] simulations can be used in conjunction with a curve-fitting procedure to extract the profiles of strain and composition.

Theory

Dynamical diffraction from a zincblende crystal is described by the Takagi-Taupin equation[5-7],

$$-i\frac{dX}{dT} = X^2 - 2\eta X + 1 \quad , \tag{6}$$

where X is the complex scattering amplitude and η is the deviation parameter, which measures deviation from the Bragg condition. The substrate thickness is usually many times the extinction depth so it can be assumed to be infinitely thick. It is also typical that commercially grown substrate materials have a low enough dislocation density so they behave as a perfect crystals. The solution to (6) given the assumptions of a perfect crystal and infinite thickness is the Darwin-Prins equation[15],

$$X = \eta_S - Sign(\eta_S)\sqrt{\eta_S^2 - 1} \quad . \tag{7}$$

The deviation parameter of the substrate is given by

$$\eta_S = \frac{-(\gamma_0/\gamma_H)(\theta - \theta_{BS})\sin(2\theta_{BS}) - 0.5(1-\gamma_0/\gamma_H)\Gamma F_{0S}}{\sqrt{|\gamma_0/\gamma_H|}C\Gamma\sqrt{F_{HS}F_{\bar{H}S}}} \quad . \tag{8}$$

where θ_{BS} is the Bragg angle for the substrate, θ is the actual angle of incidence on the diffracting planes, F_{0S}, F_{HS} and $F_{\bar{H}S}$ are the substrate structure factors for the 000, hkl, and $\bar{h}\bar{k}\bar{l}$ reflections, respectively, C is the polarization factor, and $\Gamma = r_e\lambda^2/(\pi V)$ where $r_e = 2.818\times10^{-5}\,\text{Å}$ is the classical

electron radius, λ is the x-ray wavelength, V is the unit cell volume, and γ_0 and γ_H are the direction cosines for the incident and reflected beams.

For the purposes of modeling multilayer heterostructures, including those containing graded compositions and superlattices, the epitaxial layers may be broken up into any number of discrete uniform-composition lamina, as depicted in Figure 5.

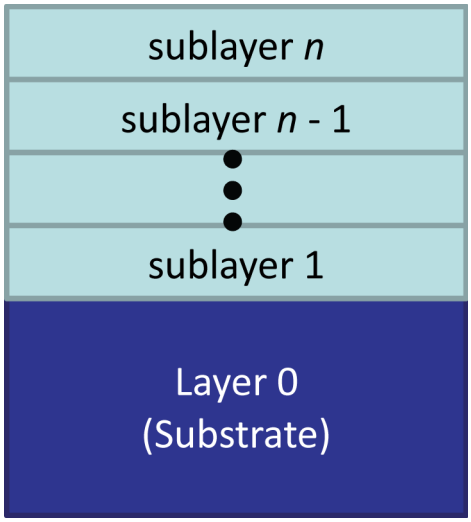

Figure 5. Laminar Modeling of a General Heterostructure

The solution to the Takagi-Taupin equations in this case is recursive, giving the scattering amplitude at the top of the n^{th} sublayer as

$$X_n = \eta_n + \sqrt{\eta_n^2 - 1} \cdot \left(\frac{S_{1n} + S_{2n}}{S_{1n} - S_{2n}} \right), \tag{9}$$

where

$$S_{1n} = \left(X_{n-1} - \eta_n + \sqrt{\eta_n^2 - 1} \right) \exp\left(-iT_n \sqrt{\eta_n^2 - 1} \right), \tag{10}$$

$$S_{2n} = \left(X_{n-1} - \eta_n + \sqrt{\eta_n^2 - 1} \right) \exp\left(iT_n \sqrt{\eta_n^2 - 1} \right), \tag{11}$$

and the thickness parameter of sublayer *n* is given by

$$T_n = h_n \frac{\pi \Gamma \sqrt{F_{Hn} F_{\bar{H}n}}}{\lambda \sqrt{|\gamma_0 \gamma_H|}}, \tag{12}$$

where h_n is the thickness of the n[th] sublayer.

Equations (6-12) can be readily applied to calculate the x-ray diffraction profile for any arbitrary, defect-free, multilayer heterostructure.

MATLAB Program

In this work, we present a MATLAB program[‡] written using the dynamical model for Bragg diffraction from nonuniform semiconductor heterostructures as described in equations (6-12). The program allows calculation of rocking curves (diffraction profiles) for any arbitrary heterostructure specified by the user (within the currently supported material systems), and of any valid *hkl* reflection and *ukv* substrate orientation. Given in Table 1 is a list of substrate materials currently supported and their corresponding lattice constants.

Substrate Material	Lattice Parameter (Å)
GaAs	5.65340
Si	5.43108
InP	5.86900
InSb	6.47940
CdTe	6.48100
GaSb	6.0960

Table 1. Substrate materials supported

[‡] Source code given in Appendix B: MATLAB Source Code

Table 2 gives the list of material systems supported for the epitaxial layers. Note that for ternary compounds, a value for a_b is also given, and the lattice parameter for a given composition x is calculated using Vegard's law for linear interpolation[16], such that

$$a(x) = xa_b + (1-x)a_a. \tag{13}$$

Epitaxial Layer Material	Lattice Parameter a_a (Å)	Lattice Parameter a_b (Å)
ZnSe	5.6687	
ZnS	5.4105	
ZnS_xSe_{1-x}	5.6687	-0.2582
$In_xGa_{1-x}As$	5.6534	0.405
$Si_{1-x}Ge_x$	5.43108	0.2265
$Hg_xCd_{1-x}Te$	6.481	-0.02
$In_xGa_{1-x}Sb$	6.096	0.3834
$Al_xGa_{1-x}As$	5.6534	0.0066
$In_xAl_{1-x}As$	5.66	0.3984

Table 2. Epitaxial layer materials supported

Atomic scattering factors for each material were obtained from Volume III of The International Tables of X-Ray Crystallography[17]. The structure factor can be calculated from the atomic scattering factor, for the hkl reflection, using

$$F_{hkl} = \sum_{n=1}^{N} f_n \exp\{2\pi i(hu_n + kv_n + lw_n)\}, \tag{14}$$

where f_n is the atomic scattering factor for the n[th] atom in a unit cell containing N atoms, (u_n, v_n, w_n) is the position of the n[th] atom normalized to the primitive unit cell vectors, and $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, and which, for the zincblende crystal structure, reduces to

$$F_{hkl} = 4\{f_A + f_B \exp[(i\pi/2)(h+k+l)]\}. \tag{15}$$

The 000 and $\bar{h}\bar{k}\bar{l}$ structure factors are then given respectively by

$$F_0 = 4\left(f_A + f_B\right), \text{ and} \tag{16}$$

$$F_{hkl} = 4\left\{f_A + f_B \exp\left[\left(i\pi/2\right)\left(-h-k-l\right)\right]\right\} \tag{17}$$

The lattice parameters and elastic constants entered for each material were derived from Heteroepitaxy of Semiconductors[18]. With these parameters, additional materials can be added in a straightforward fashion.

The program supports uniform-composition layers, linearly graded (LG) composition layers, and superlattice (SL) layers. Also, strain in each layer of the structure can be specified as completely relaxed (none), pseudomorphic (coherent strain), custom constant strain, and custom constant percent residual strain (LG only).

Epilayer tilt is an important physical consideration in the analysis of x-ray diffraction rocking curves, and it is also accounted for by allowing the entry of each epilayer's tilt in arc seconds. In diffraction experiments, it is common to measure this effect so as to not draw erroneous conclusions from the data, and so we have also included the ability to calculate the rocking curve at any azimuth. (The axis of rotation of the "azimuth" is the normal to the sample surface being probed.)

To account for typical instrumental effects, calculated rocking curves may be convolved with an instrumental broadening function whose width depends on the geometry as well as the instrument[1, 2, 19]. The program currently supports Gaussian and rectangular broadening functions of a specified width.

A number of commonly-used x-ray source radiations are supported, and these are enumerated in Table 3.

Radiation Line	Wavelength (Å)
Cr $k\alpha_1$	2.28970
Fe $k\alpha_1$	1.936042
Cu $k\alpha_1$	1.540594
Mo $k\alpha_1$	0.70930
Ag $k\alpha_1$	0.5594075

Table 3. X-ray source radiations supported

Results

In this section, we present results for heterostructures in the $Si_{1-x}Ge_x$ / Si (001) material system. The x-ray source radiation corresponds to the Cu kα_1 ($\lambda = 0.1540594 nm$) line. Linearly graded layers were broken up into 31 sublayers. In general, the necessary number of sublayers for accurate modeling of the structure depends on the hkl reflection, layer thickness, compositional profile, and dislocation density in a complex manner. A uniform layer of finite thickness is modeled without division into sublayers. In a layer with nonuniform composition, strain, defect density—or any other parameter that one performing a simulation might like to model—the necessary number of sublayers increases with maximum gradient in that particular parameter. For example, a 1.0-µm LG layer with grading coefficient of 0.2% cm^{-1} would need to be modeled with more sublayers than a 1.0-µm LG layer with grading coefficient 0.1% cm^{-1}. When considering different hkl reflections, the necessary number of sublayers increases with Bragg angle but this is a weaker effect than that mentioned above. However, because of the complex interplay of these factors in determining the necessary number of sublayers, good practice dictates that one increase the number of sublayers until the diffraction profile is no longer affected by further increases. Here, we present unbroadened rocking curves, which exhibit complex interference fringe (Pendellosung) structure, as well as rocking curves broadened by convolution with an instrumental broadening function in cases where Pendellosung interfere with presentation. In such cases, a Gaussian with a width of 12 arc seconds was used, which represents the divergence for a 2-crystal, 4-reflection Ge(220) Bartels monochromator[1, 2].

Substrate

First we present results for a Si (001) substrate. This calculation is made by the Darwin-Prins equation[15] (7), which assumes the substrate to be perfect (free of crystallographic defects) and infinitely thick (this assumption is safe as the substrate is typically many times the x-ray extinction depth)[18]. In this example, we have also assumed the substrate to be completely relaxed. Given in Figure 6 is the unbroadened 004 rocking curve for this structure, and this simulation gives the intrinsic full width at half maximum (FWHM) to be 2.33 arc seconds.

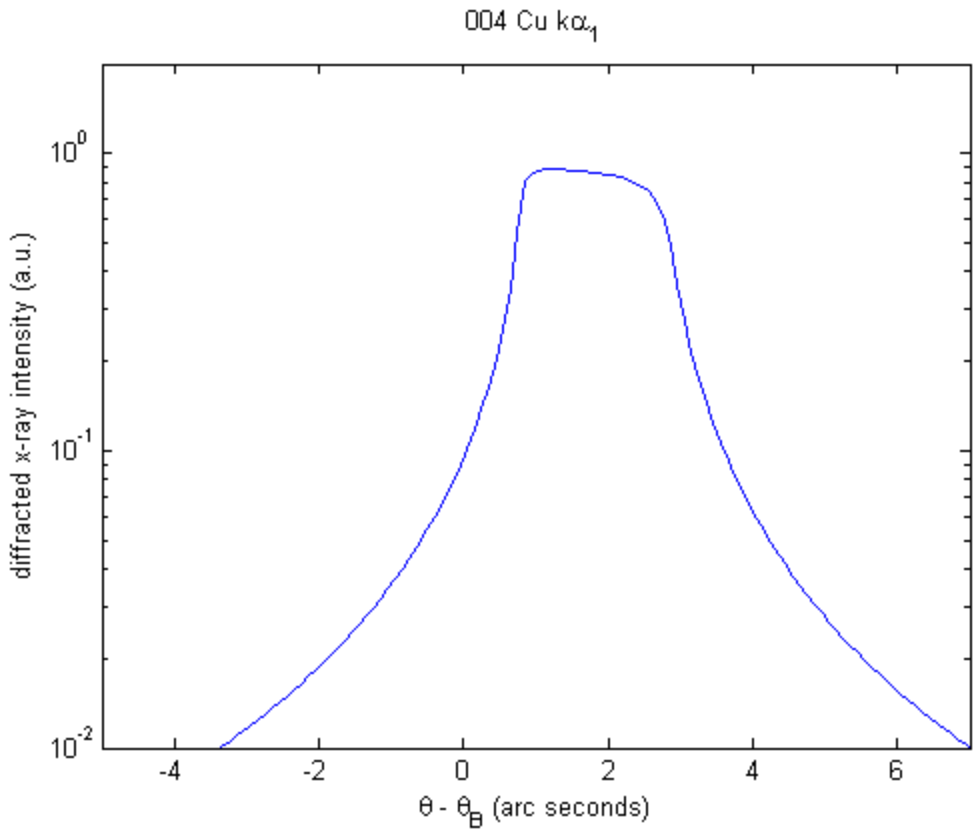

Figure 6. 004 Rocking curve for Si substrate, unbroadened. FWHM = 2.33''

Figure 7 gives the rocking curve broadened by convolution with a 12 arc second Gaussian instrumental broadening function, and this results in a 12.38 arc second peak width. (Note the x-scale is increased greatly over that in Figure 6.) The asymmetry of the substrate peak is also reduced substantially by the convolution process (but in this case only because the broadening function, being 6 times the width of the substrate peak, dominates the convolved product).

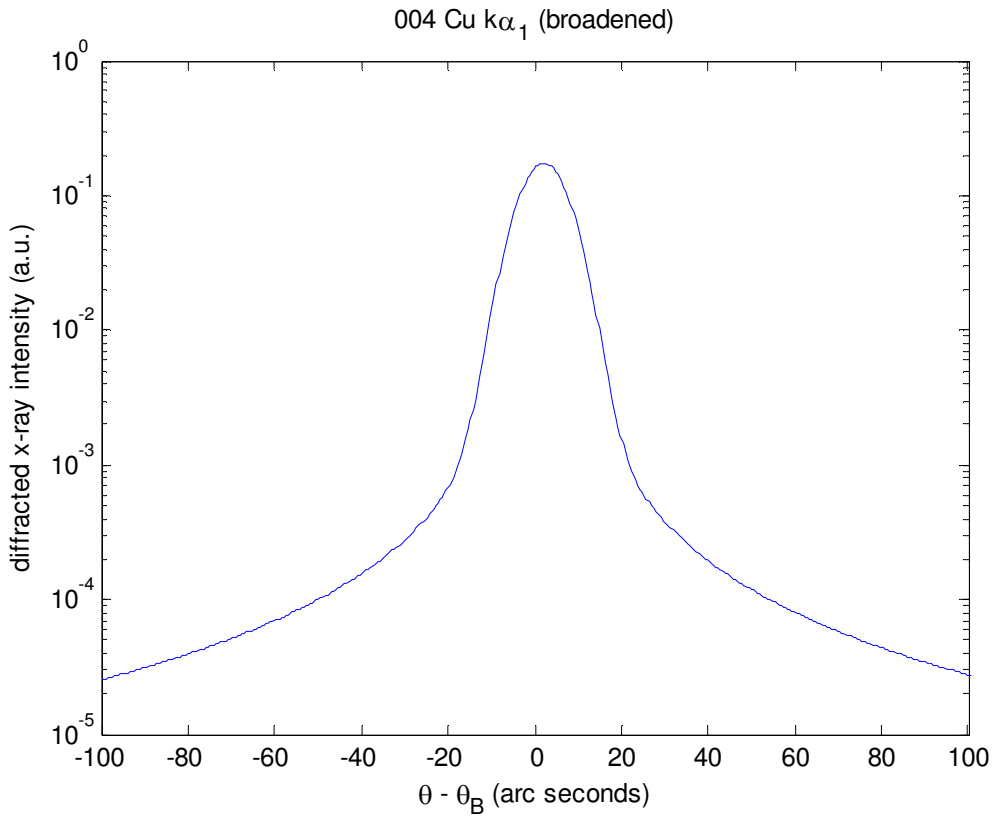

Figure 7. 004 Rocking curve for Si substrate, broadened. FWHM = 12.38".

Uniform Layers

Next we consider a 1.0-µm layer of $Si_{0.9}Ge_{0.1}$ on the same Si (001) substrate as the preceding example. As before, this is calculated using the Darwin-Prins equation, except this time it is followed by one recursion of the Takagi-Taupin equation for laminar modeling, given by equations (9-12). The epitaxial layer is assumed to be compositionally uniform and unstrained (completely relaxed). Given in Figure 8 is the 004 rocking curve for this structure. As with all rocking curve plots presented in this work, the x-axis is referenced to the Bragg angle of the substrate, so that 0 corresponds to the substrate peak. The FWHM of the epitaxial layer is 17.3 arc seconds (and the substrate as before). A very structured diffraction pattern of interference fringes is noted.

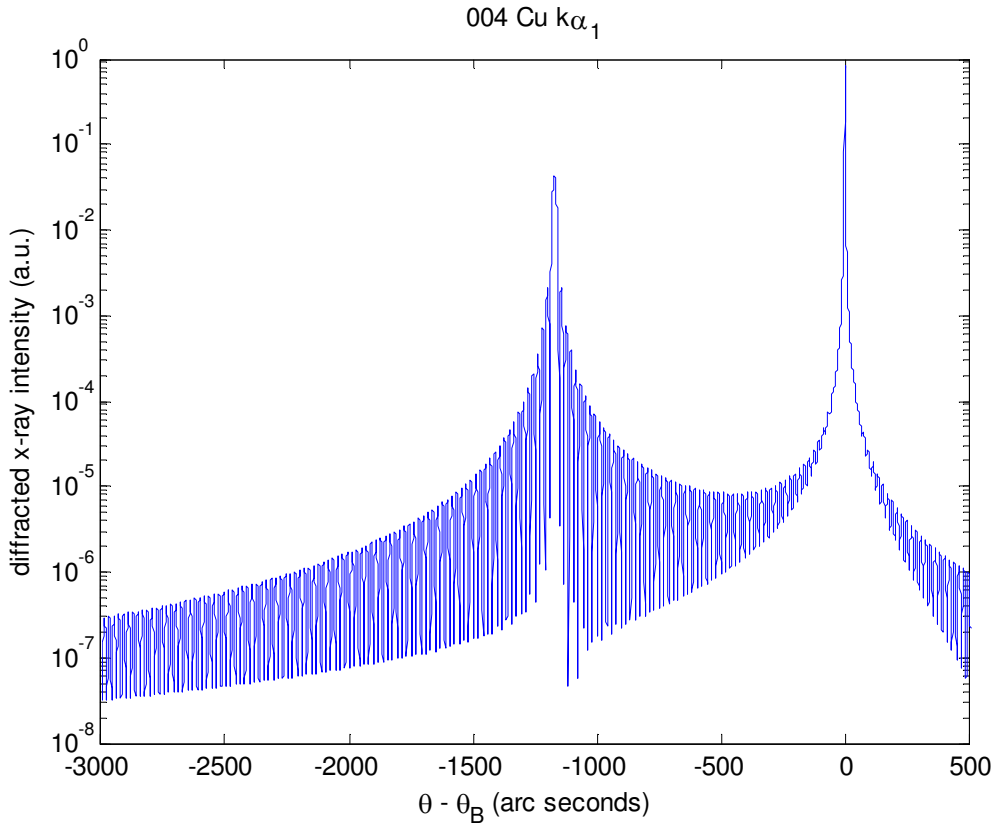

Figure 8. 004 Rocking curve for $Si_{0.9}Ge_{0.1}$ / Si (001) structure, unbroadened.

Given in Figure 9 is the broadened rocking curve for this structure, and the FWHM of the epitaxial layer in this case is 20.3 arc seconds. This is fairly consistent with the prediction where the shape of the peak is modeled as a Gaussian,

$$\beta_{broadened} = \sqrt{\beta_{unbroadened}^2 + \beta_{broad.fn.}^2}, \tag{18}$$

which predicts a width of 21 arc seconds. Again we note numerous Pendellosung in the unbroadened structure, but in the broadened structure which accounts for instrumental effects of broadening, we see that the fringes are attenuated significantly, as they are in experimental diffraction measurements.

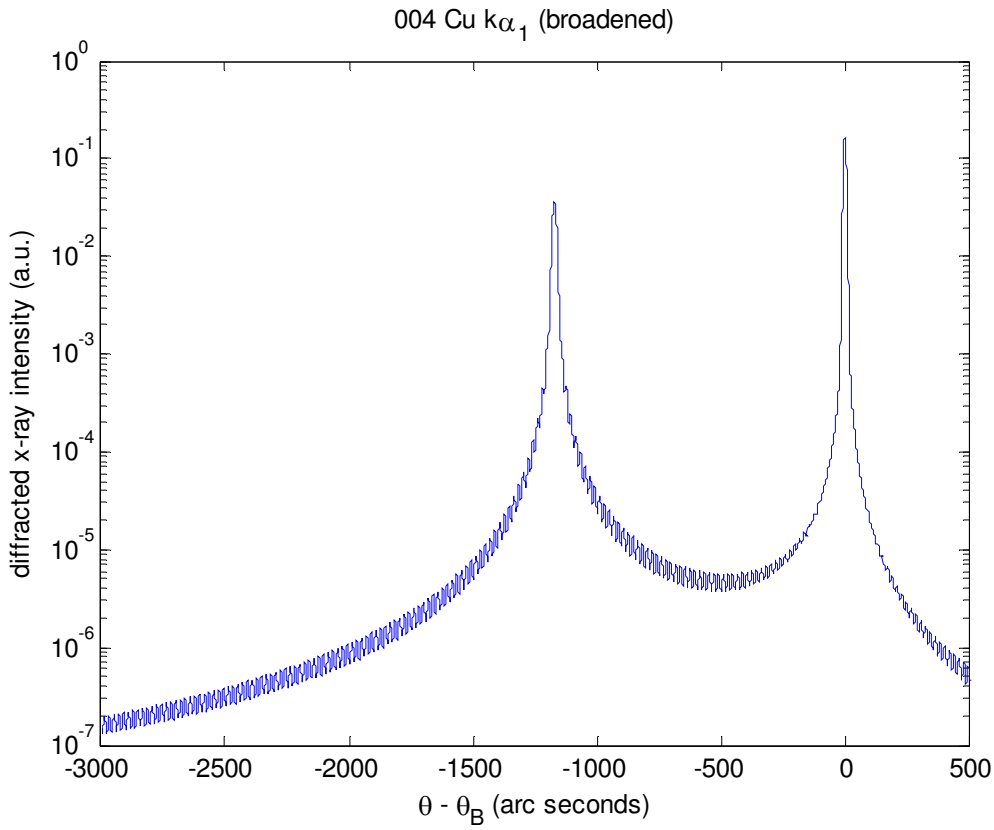

Figure 9. 004 Rocking curve for $Si_{0.9}Ge_{0.1}$ / Si (001) structure, broadened.

Next we demonstrate what happens as strain is introduced to the epitaxial layer. Here, we assume the pseudomorphic case, or coherent strain. In this case the in-plane strain is equal to the lattice mismatch,

$$\varepsilon_{\parallel} = f \ . \tag{19}$$

This is the opposite extreme from the completely relaxed case, in which all the mismatch of the substrate-epitaxial layer is taken up by biaxial strain in the epitaxial layer, and the unit cell is tetragonally distorted. In real structures, this type of growth only tends to happen when there is low mismatch and when the epilayer thickness is less than the critical layer thickness. Above this thickness, the strain energy becomes too high and misfit dislocations nucleate to relieve the strain.

The structure is the same uniform 1.0-µm layer of $Si_{0.9}Ge_{0.1}$ on a Si (001) substrate, except the epilayer is under coherent strain. The strain profile for this structure is given in Figure 10 (this plot is automatically produced if specified).

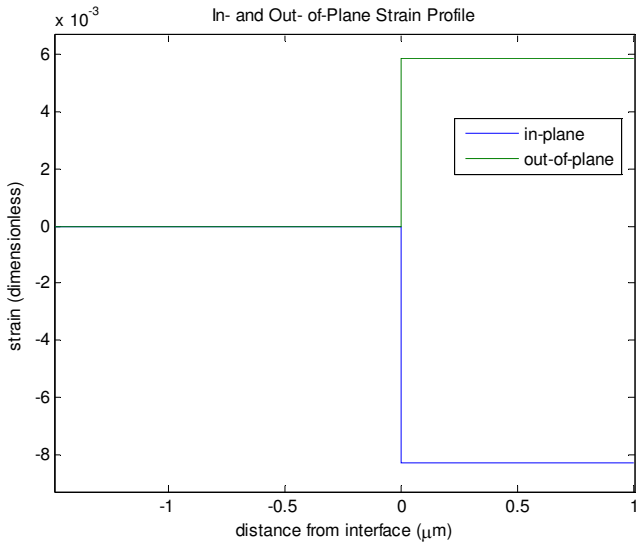

Figure 10. Strain profile for coherently strained $Si_{0.9}Ge_{0.1}$ / Si (001) structure.

The broadened 004 rocking curve for this structure is given in Figure 11.

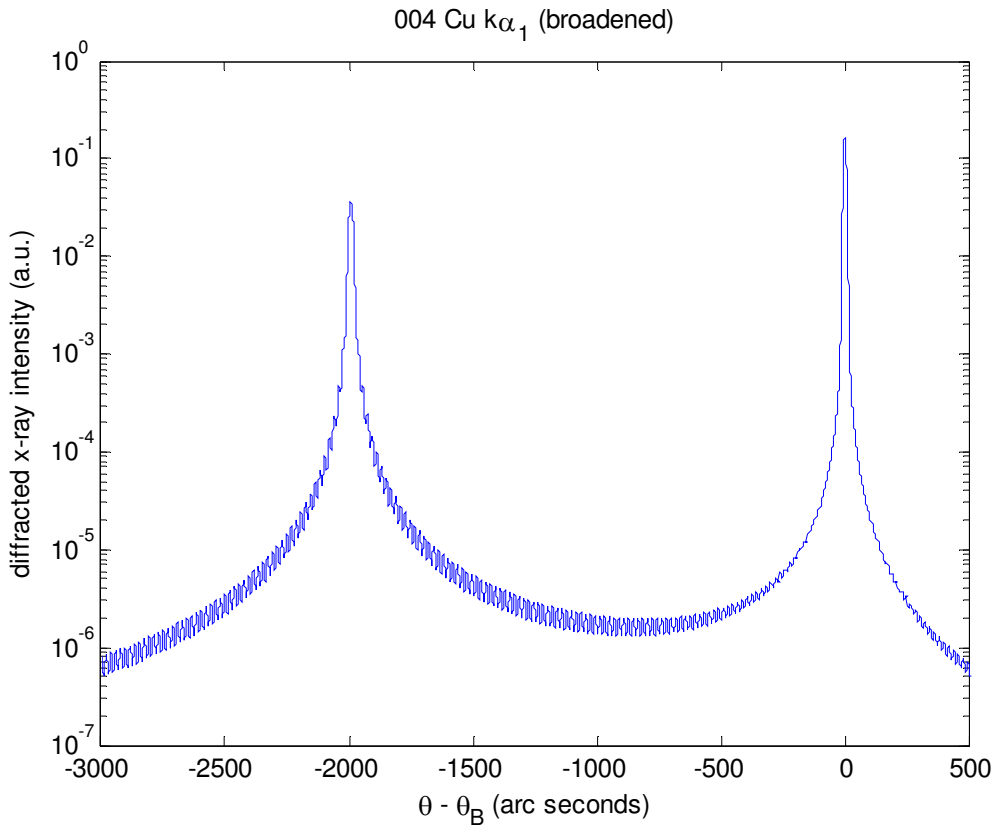

Figure 11. 004 Rocking curve for coherently strained $Si_{0.9}Ge_{0.1}$ / Si (001) structure.

The FWHM of the epitaxial layer peak is again 20.3 arc seconds, but the peak has been shifted from $\theta - \theta_B = -1170$ arc seconds to $\theta - \theta_B = -1992$ arc seconds. This shift in the Bragg angle happens because the tetragonal distortion of the unit cell means the out-of-plane lattice constant is modified, and this results in a change in the spacing of the diffracting planes. Therefore, with respect to the substrate, the strain depicted in Figure 10 yields a change in the 004 Bragg angle of 822 arc seconds.

Introducing an epilayer tilt to the structure would have a similar effect to this, and experimentally, measurement at multiple azimuths would be needed to separate these effects.

Linearly Graded Layers

Graded buffer layers are used to accommodate the mismatch between the substrate and device layers. The composition of the material is graded, from a starting value at the substrate interface, to a final value at the interface with the device layers. Traditionally, this has been done in a linear manner, so that

the lattice mismatch is $f = C_f y$, where y is the distance from the substrate interface and C_f is the grading coefficient. The linear compositional grading scheme has been used since at least the sixties[20, 21] and today remains a commonly employed approach used to control defect densities in the upper (device) layers.

We present a simple graded structure comprising a Si (001) substrate and a relaxed 1.0-µm $Si_{1-x}Ge_x$ buffer layer with its composition graded linearly from x=0 at the substrate interface and x=10% at the surface. For this structure, no significant change was found in the diffraction profile when modeling this LG layer with more than 81 sublayers, indicating that this number was sufficient. The 004 rocking curve for this structure is given in Figure 12.

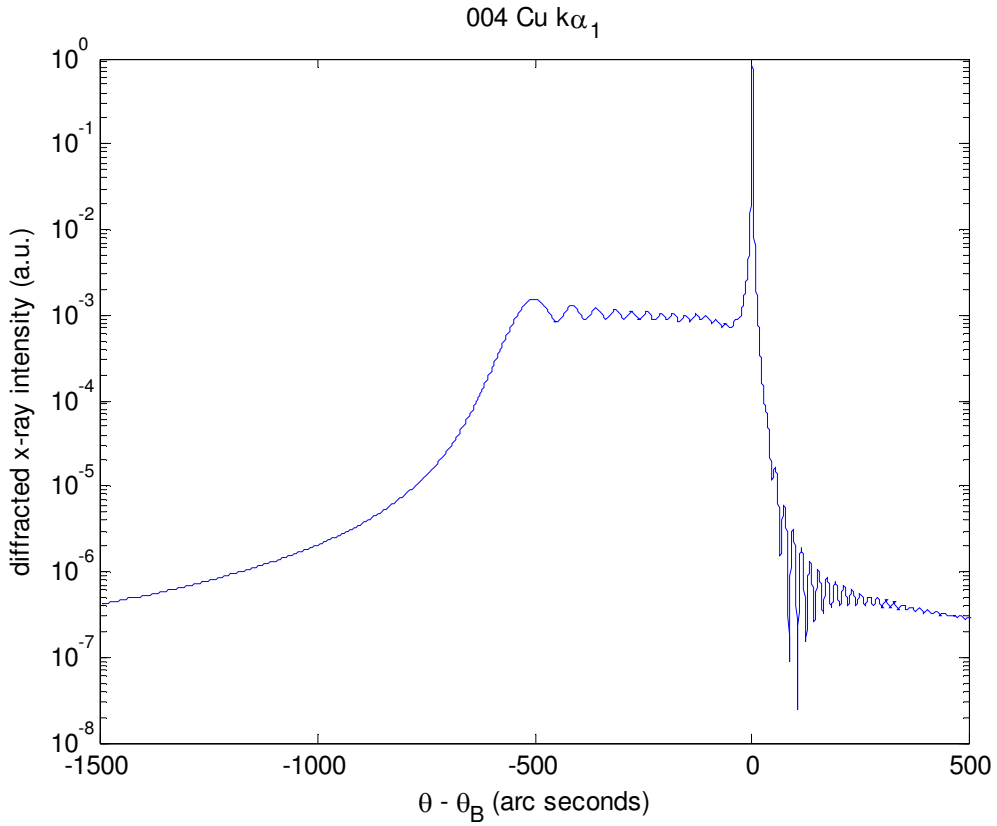

Figure 12. 004 Rocking curve for the linearly graded $Si_{1-x}Ge_x$ / Si (001), x=0-10%, structure

Typical interference fringes are noted to the right of the substrate peak, and these are related to dynamical interactions in the substrate. However, to the left of the substrate, they are washed out by the presence of the LG buffer, and instead an interesting pattern of fringes is present. This pattern is the result of dynamical phase interactions in the buffer, and is a diffraction characteristic typically exhibited by LG layers.

Superlattice Structure

A superlattice structure is a periodic structure of two or more materials, with thin sublayers that alternate composition back and forth. They are sometimes used to control defect densities in device layers through interactions of threading dislocations at the boundaries of the periods (between the thin sublayers). Such interactions that reduce defect densities that have been observed experimentally[22] are annihilation, coalescence, and escape to the edge of the sample. They are also of interest in other device applications and characterization experiments.

Here we present results for a superlattice structure examined in a 2013 work by Rago and Ayers[23] which comprises a 10-period, 10-nm $In_{0.03}Ga_{0.97}As$/10-nm $In_{0.04}Ga_{0.96}As$ superlattice on GaAs (001). For clarity, the lattice profile for the structure is given in Figure 13. This structure was assumed to be completely relaxed. The 044 rocking curve for the structure is given in Figure 14. The superlattice structure alone produces a complex diffraction behavior, with a centrally located 0^{th}-order superlattice peak located around $\theta - \theta_B = -600$ arc seconds and satellite peaks of decaying intensity on either side of the 0^{th}-order peak. Interestingly, some of the periodicity originating from the superlattice layer even diffracts at angles greater than θ_B.

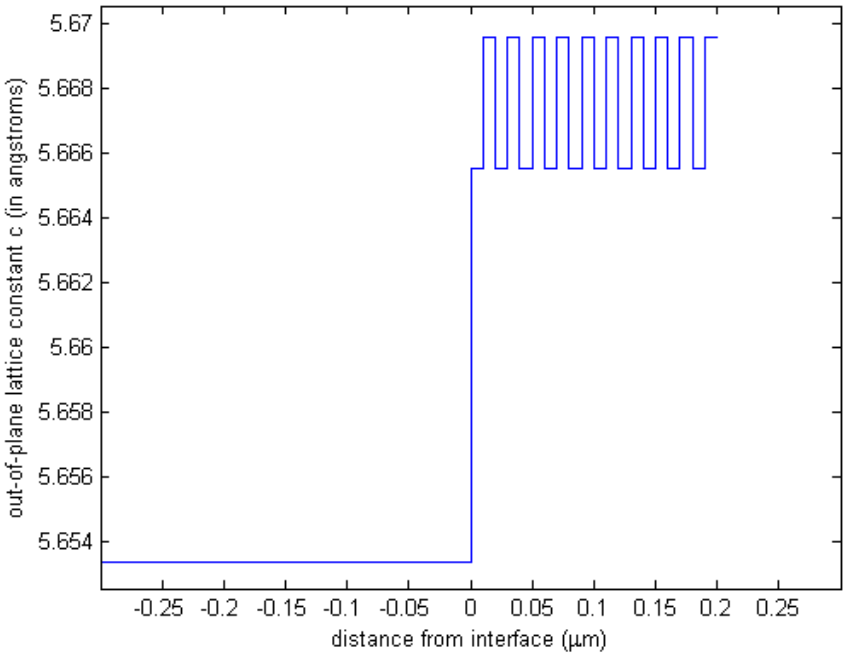

Figure 13. Lattice profile for the $In_xGa_{1-x}As$ / GaAs (001) superlattice structure.

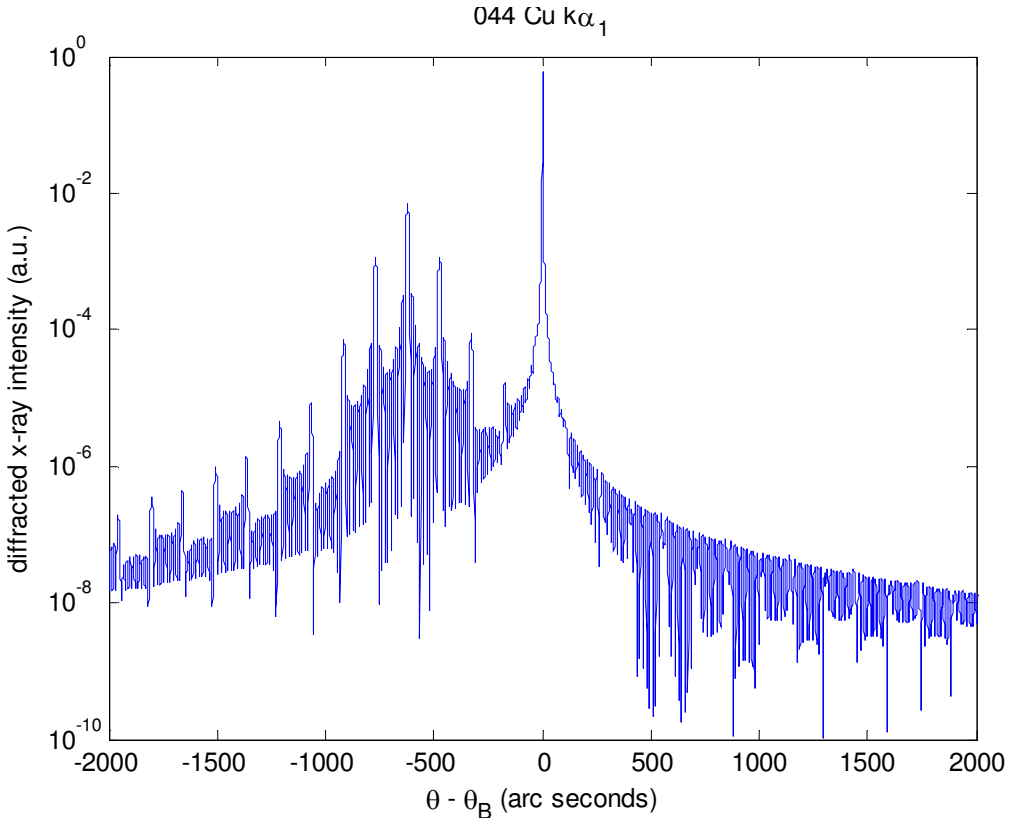

Figure 14. 044 Rocking curve for the $In_xGa_{1-x}As$ / GaAs (001) superlattice structure.

The results from the preceding structures demonstrate just a few examples of an infinite number of simulations possible using the dynamical diffraction model outlined by Takagi and Taupin. This method is commonly used to gather additional information from x-ray diffraction experiments, but a major limitation is its inability to model dislocation density, which is of great interest to crystal growers, defect engineers, and heteroepitaxial device designers.

Phase-Invariant Dynamical Diffraction Model

Dynamical simulations are commonly used in conjunction with a process of curve-fitting to experimentally measured rocking curves to indirectly estimate the depth profiles of strain and composition. Because these simulations are based on perfect, dislocation-free crystals, the analysis is inapplicable to many real structures, namely those containing dislocation densities greater than 10^6 cm^{-2}. In this work we present a dynamical model for Bragg x-ray diffraction in heteroepitaxial structures with nonuniform strain, composition, and dislocation density. It is based on the Takagi-Taupin equation for distorted crystals as outlined in equations (6-12), but it accounts for the diffuse scattering that arises from mosaicity of strain and angular misorientation associated with the presence of dislocations. We show that the diffraction profiles from ZnS_xSe_{1-x} and $In_xGa_{1-x}As$ / GaAs (001) heterostructures are affected strongly by the presence of dislocations, so that dislocation density can be extracted as well as strain and composition through the analysis experimentally measured rocking curves.

Krivoglaz & Ryaboshapka[24] and Levine & Thomson[25] have analyzed the line profiles of Bragg diffraction peaks from crystals containing straight, parallel screw dislocations with precisely specified atomic displacements. Their treatment is mathematically rigorous, but it is not possible to determine all of the atomic displacements in a real crystal containing a diverse network of dislocation distributions, orientations, shapes, and Burgers vectors. Indeed, it would be impossible to completely describe atomic positions of a real structure with its complex network of dislocations. Because of this, in this work we consider the ensemble of dislocations in a defected semiconductor heterostructure, and we describe the dislocations as characteristic statistical distributions associated with the ensemble average angular and strain broadening of the Bragg profiles by the dislocations, which are the two main ways that dislocations distort the otherwise orderly lattice. As illustrated in Figure 15, some mosaic blocks have angular misorientation (a) while others have a modified spacing of the diffracting planes (b). Here we see that both effects serve to modify the angle at which that block diffracts. We take advantage of this effect to adapt the theory of dynamical diffraction laid out by Takagi and Taupin.

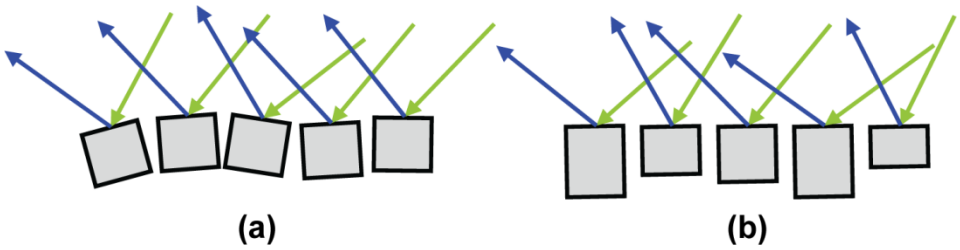

Figure 15. The effect of angular (a) and (b) interplanar spacing variations of mosaic blocks on diffraction.

This approach is quite general but for its application here we have considered the specific case of Gaussian distributions for the angular and mosaic spread to calculate the Bragg diffraction profiles from semiconductor structures with nonuniform strain, composition, and dislocation density. It has been shown in previous work that curve-fitting to dynamical simulations can yield indirect measurements of the strain profiles in semiconductor heterostructures, and so for this research we focused on the dislocation density and composition profiles. Therefore, to simplify their interpretation, we present results for unstrained structures, but it is important to note that the general conclusions still hold for structures containing residual strain.

Theory

We define the statistical distributions of angular orientation as $P_\alpha(\theta)$ and the distribution of interplanar spacing variation as $P_\varepsilon(\theta)$. We then modify the deviation parameter given by equation (8) to include these statistical distributions, to give an effective deviation parameter

$$\eta_n = \left\{ \sqrt{|\gamma_0 / \gamma_H|}\, C\Gamma \sqrt{F_{Hn} F_{\overline{H}n}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{P_\alpha(\alpha) P_\varepsilon(\beta)\, d\alpha d\beta}{-(\gamma_0/\gamma_H)(\theta+\alpha-\theta_{Bn}-\beta)\sin(2\theta_{Bn}) - 0.5(1-\gamma_0/\gamma_H)\Gamma F_{0n}} \right\}^{-1}, \qquad (20)$$

where θ_{Bn} is the Bragg angle for the nth sublayer and θ is the actual angle of incidence on the diffracting planes, F_{0n}, F_{Hn} and $F_{\overline{H}n}$ are the structure factors for the 000, hkl, and $\overline{h}\,\overline{k}\,\overline{l}$ reflections, respectively, C is the polarization factor, and $\Gamma = r_e \lambda^2 / (\pi V)$ where $r_e = 2.818 \times 10^{-5}$ Å is the classical electron

radius, λ is the x-ray wavelength, and V is the volume of the unit cell. This effective deviation parameter is discretized and the integrals become a summation for the purposes of computer calculation. The effective deviation parameter (20) is calculated then used in the recursion formulae given earlier for the perfect case but restated here for clarity, where the scattering amplitude is given by

$$X_n = \eta_n + \sqrt{\eta_n^2 - 1}\,\frac{(S_{1n} + S_{2n})}{S_{1n} - S_{2n}}, \tag{21}$$

where

$$S_{1n} = \left(X_{n-1} - \eta_n + \sqrt{\eta_n^2 - 1}\right)\exp\left(-iT_n\sqrt{\eta_n^2 - 1}\right) \tag{22}$$

and

$$S_{2n} = \left(X_{n-1} - \eta_n + \sqrt{\eta_n^2 - 1}\right)\exp\left(iT_n\sqrt{\eta_n^2 - 1}\right). \tag{23}$$

As before, the scattering amplitude is first calculated for the substrate, then recursively for each sublayer until the surface of the structure being modeled has been reached. The approach described by (20-23) is quite general, but for its application here we have considered the specific case of Gaussian distributions for the angle-scale distributions associated with the variations of orientation and interplanar spacing of the mosaic crystal. In 1953 Gay Hirsch and Kelly[26] showed for metals that the distribution of angular variation of mosaic blocks may be modeled as a Gaussian distribution, and in 1994, Ayers later adapted this work for semiconductors[2], giving the distribution for the angular orientation to be

$$P_\alpha(\theta) = (1/(\sigma_\alpha\sqrt{2\pi}))\exp\left(-\theta^2/(2\sigma_\alpha^2)\right), \tag{24}$$

where the standard deviation is

$$\sigma_\alpha = b\sqrt{\pi D}/2, \tag{25}$$

D is the dislocation density, and b is the Burgers vector for the dislocations.

On the basis of the mosaic block model, it was also shown for metals in 1961 by Hordon and Averbach[27], and later adapted for semiconductors by Ayers[2], that for a symmetric reflection from a (001) semiconductor heterostructure with 60° dislocations the scale distribution $P_\varepsilon(\theta)$ may be modeled as a Gaussian distribution given by

$$P_\varepsilon(\theta) = (1/(\sigma_\varepsilon\sqrt{2\pi}))\exp\left(-\theta^2/(2\sigma_\varepsilon^2)\right), \tag{26}$$

for which the standard deviation is

$$\sigma_\varepsilon = 0.127b\sqrt{D\left|\ln\left(2\times10^{-7}\,cm\sqrt{D}\right)\right|}\tan\theta_B. \tag{27}$$

Although we have considered Gaussian distributions in this study, the dynamical model presented here may be readily applied using other types of distributions, such as those suggested by Ivanov et al. for metal crystals[28]. Other statistical distributions could be determined either experimentally or by use of the Krivoglaz & Ryaboshapka[24] theoretical framework by making detailed calculations for particular dislocation configurations.

Presently a significant gap exists between the rigorous modeling of long, straight dislocations with precisely known atomic displacements and the treatment of real semiconductor structures with complex dislocation networks. It will be necessary to apply rigorous models, as was done by Krivoglaz & Ryaboshapka and Hordon & Averbach, to random dislocation networks so that the expressions (24-27) may be refined further. It is hoped that with the ever-increasing computational power this may soon become feasible. Though this endeavor could lead to refined models described by equations (24-27), the general approach described here by equations would remain unchanged.

To determine the in-plane lattice constant and strain requires x-ray diffraction measurement of asymmetric reflections. Depicted in Figure 3 is a symmetric reflection, showing interplanar spacing *d*, while Figure 4 depicts an asymmetric reflection for which the angle between the diffracting planes and

the surface is φ. For more information and the expressions describing key concepts of asymmetric reflections, see the section entitled Reflections and Symmetry.

For an asymmetric reflection, (20) becomes

$$\eta_n = \left\{ \sqrt{|\gamma_0 / \gamma_H|} C\Gamma \sqrt{F_{Hn} F_{\bar{H}n}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{P_\alpha(\alpha) P_\varepsilon(\beta) d\alpha d\beta}{-(\gamma_0 / \gamma_H)(\theta + \alpha - \theta_{Bn} - \beta + \Delta\varphi_{tet}) \sin(2\theta_{Bn}) - 0.5(1 - \gamma_0 / \gamma_H)\Gamma F_{0n}} \right\}^{-1}, \quad (28)$$

where $\Delta\varphi_{tet}$ is the change in φ due to tetragonal distortion.

The framework presented here applies to any hkl reflection from a (001) oriented diamond or zincblende crystal, and our results include both symmetric and asymmetric reflections. It is therefore applicable to III-V/Si (001), II-VI/Si (001), SiGe/Si (001), and II-VI/III-V (001) heterostructures with any arbitrary laminar configuration. It can also be readily extended to other crystal orientations with the appropriate modification of (14-17) and (28).

Results

Here we apply the modified dynamical model for Bragg diffraction from nonuniform semiconductor structures as outlined in equations (20-28) to calculate diffraction profiles of various reflections for $In_xGa_{1-x}As$ / GaAs (001) heterostructures. The x-ray source wavelength used corresponds to the Cu kα_1 line ($\lambda = 0.1540594 nm$). To account for typical instrumental effects, all rocking curves were convolved with a Gaussian instrumental broadening function with a 20 arc second full width at half maximum (FWHM), which corresponds to a standard deviation for the Gaussian of 8.5 arc seconds.

First we consider structures containing a 1.0-µm uniform layer of $In_{0.05}Ga_{0.95}As$ with a GaAs (001) substrate and various values of assumed threading dislocation density D. The 002, 004, and 006 diffraction profiles are given in Figure 16, Figure 17, and Figure 18, respectively, for these samples. Dynamical interactions with the diffracted intensity corresponding to the substrate yield an asymmetric

$In_{0.05}Ga_{0.95}As$ Bragg peak profile, and this is most pronounced in the case of the 002 reflection. For all three reflections the FWHM of the $In_{0.05}Ga_{0.95}As$ peak increases monotonically as *D* is increased from 0 to $5x10^7$ cm^{-2}.

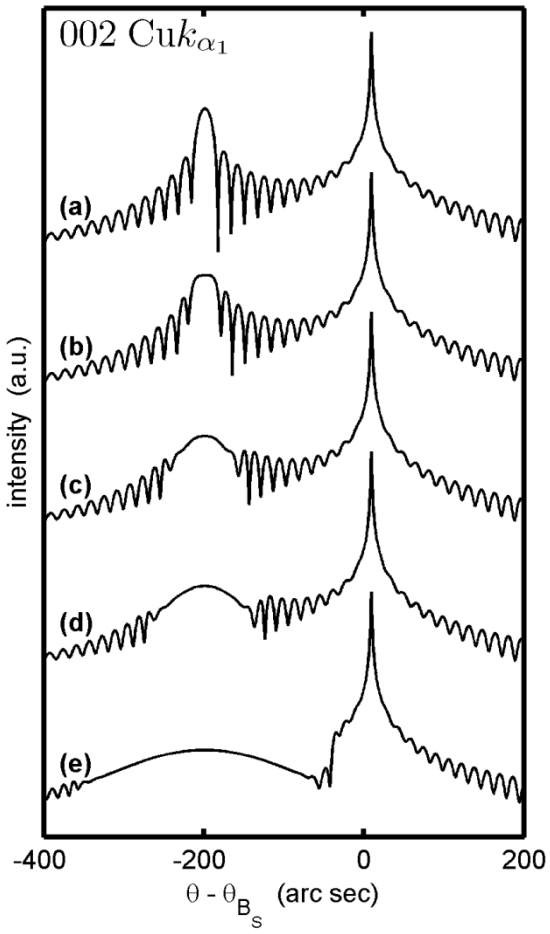

Figure 16. 002 rocking curves for uniform 1.0-μm $In_{0.05}Ga_{0.95}As$ / As (001) with D equal to (a) 0 cm-2, (b) 10^6 cm^{-2}, (c) $5x10^6$ cm^{-2}, (d) 10^7 cm^{-2}, and (e) $5x10^7$ cm^{-2}.

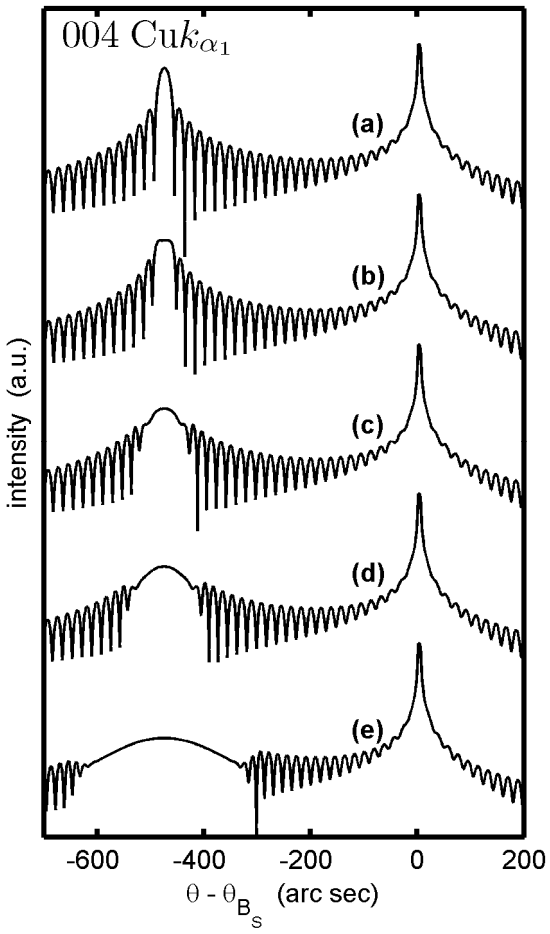

Figure 17. 004 rocking curves for uniform 1.0-µm $In_{0.05}Ga_{0.95}As$ / As (001) with D equal to (a) 0 cm-2, (b) 10^6 cm^{-2}, (c) $5x10^6$ cm^{-2}, (d) 10^7 cm^{-2}, and (e) $5x10^7$ cm^{-2}.

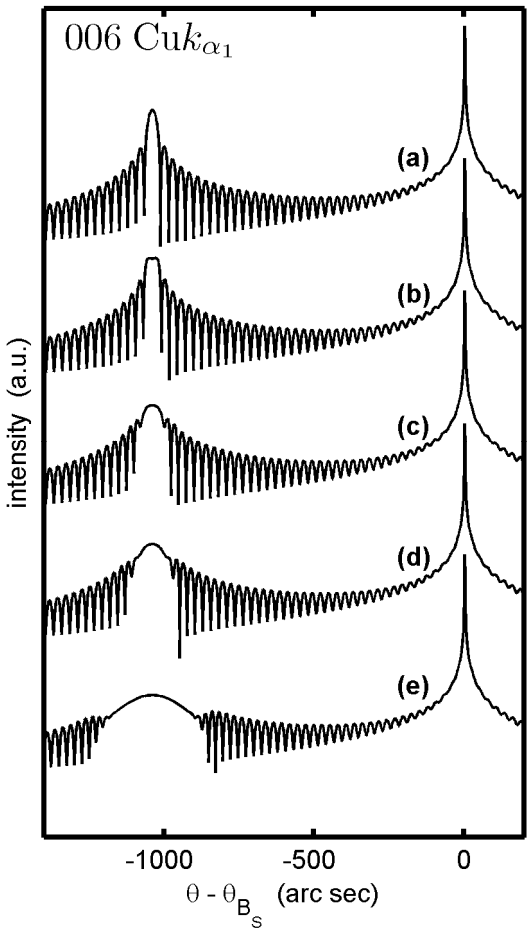

Figure 18. 006 rocking curves for uniform 1.0-µm $In_{0.05}Ga_{0.95}As$ / As (001) with D equal to (a) 0 cm-2, (b) 10^6 cm^{-2}, (c) $5x10^6$ cm^{-2}, (d) 10^7 cm^{-2}, and (e) $5x10^7$ cm^{-2}.

The convolution model[2] provides a simple hand calculation for estimating dislocation density from the width of the Bragg peak for a uniform layer, and it is based on the same mosaic block model as the phase-invariant model for dynamical diffraction. To compare the detailed dynamical simulations to the convolution model, Figure 19 gives the rocking curve FWHM for the 002, 004, and 006 Bragg reflections as a function of the dislocation density D. Also shown in Figure 19 is the rocking curve width calculated using the convolution model[2], for which

$$\beta^2 = \beta_0^2 + \beta_\alpha{}^2 + \beta_\varepsilon{}^2 = \beta_0^2 + 2\pi b^2 D \ln 2 + 0.16 b^2 D \left|\ln\left(2\times 10^{-7}\sqrt{D}\right)\right| \tan^2\theta_B , \tag{29}$$

where β_0 is the intrinsic rocking curve width, determined using a dynamical simulation for a perfect crystal with the same thickness, and β_α, and β_ε are the FWHMs of the angle-scale distributions associated with angular and strain mosaic spread, respectively. It is seen that there is good agreement between the dynamical calculations and the convolution model (within 20%) , even at the lower dislocation densities which are near the lower limit for the use of x-ray characterization to measure dislocation density.

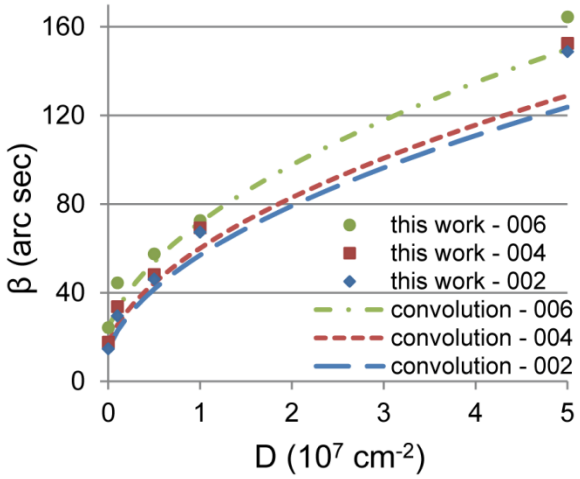

Figure 19. Comparison of Bragg peak width to that predicted by the convolution model[2].

Next, to further compare the detailed dynamical simulations to the convolution model, we consider seven reflections from a 1.0-µm uniform layer of $In_{0.05}Ga_{0.95}As$ / GaAs (001) with constant D = $5x10^6$ cm^{-2}. Shown in Figure 20 are the 002, 004, 044, 135, 006, 026, and 335 reflections, which are in order of increasing Bragg angle.

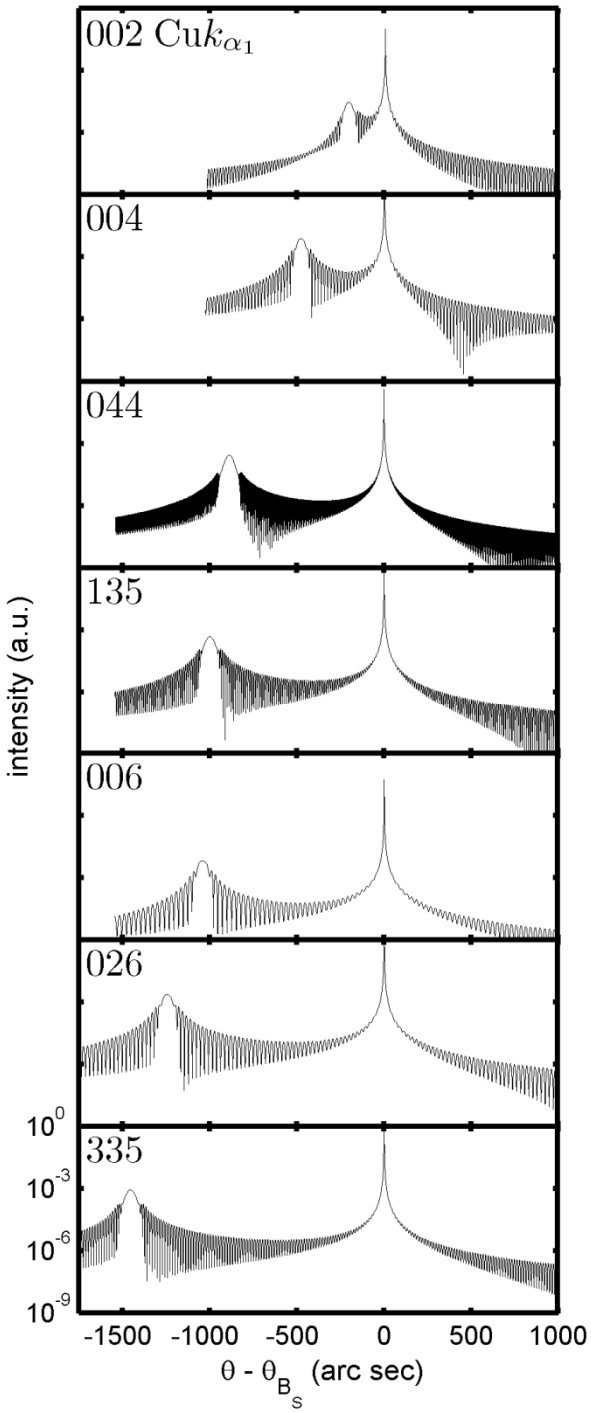

Figure 20. Rocking curves for various reflections from 1.0µm $In_{0.05}Ga_{0.95}As$ / GaAs (001) with D = $5x10^{6}$ cm^{-2}. Both the intensity and angular scales are the same in all plots.

In the simplest form, the convolution model is applied by neglecting the intrinsic width of the rocking curve width β_o, including only the widths due to angular (β_α) and strain (β_ε) broadening. This is

appropriate for uniform samples with high, uniform dislocation densities, for which $\beta_\alpha{}^2 + \beta_\varepsilon{}^2 >> \beta_o^2$ and yields a linear relationship between the square width β^2 and $\tan^2\theta_B$, which is given by

$$\beta^2 = \beta_\alpha{}^2 + \beta_\varepsilon{}^2 = 2\pi b^2 D \ln 2 + 0.16 b^2 D \left|\ln\left(2\times10^{-7}\sqrt{D}\right)\right| \tan^2\theta_B . \tag{30}$$

Plotted in Figure 21 are the widths from the various reflections of the D = 5x10^6 cm^{-2} sample, as calculated by this work, as well the approximate analysis given by equation (30) excluding the intrinsic rocking curve width.

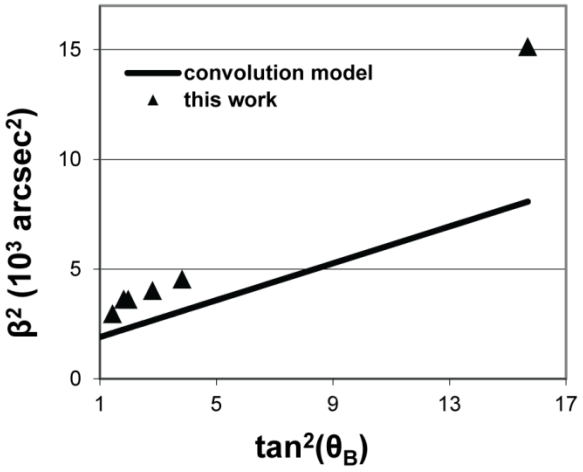

Figure 21. Comparison of squared width vs. tan^2(θ_B) for various reflections to the convolution model[2] from a 1.0-µm $In_{0.05}Ga_{0.95}As$ /GaAs (001) structure with D = 5x10^6 cm^{-2}.

It is seen in Figure 21 that the convolution model greatly underestimates the width when the intrinsic width is excluded. This is likely due to the relatively low dislocation density studied here, as that term will be a greater contributor to β^2. When including the intrinsic width of the epilayer, the convolution model agrees to within ~20%. Also, an important advantage of the present model is the ability to analyze nonuniform structures for which the convolution model is not applicable.

To demonstrate this we calculated 044 diffraction profiles from 1.0-µm $In_{0.07}Ga_{0.93}As$ / GaAs (001) structures with the epilayer containing linear dislocation density profiles. In these structures, D varied linearly from D_1 at the substrate interface to D_2 at the surface. Figure 22 shows the calculated rocking curves for the structures with (D_1, D_2) equal to ($2x10^7$, 0), ($1.5x10^7$, $0.5x10^7$), ($1.2x10^7$, $0.8x10^7$), and ($1x10^7$, $1x10^7$) cm^{-2}. All four samples have the same average defect density of 10^7 cm^{-2}, yet the rocking curve peak intensity, width, and shape are all influenced by the dislocation distribution. This demonstrates that it should be possible to extract this distribution by curve-fitting to a measured high-resolution x-ray diffraction profile.

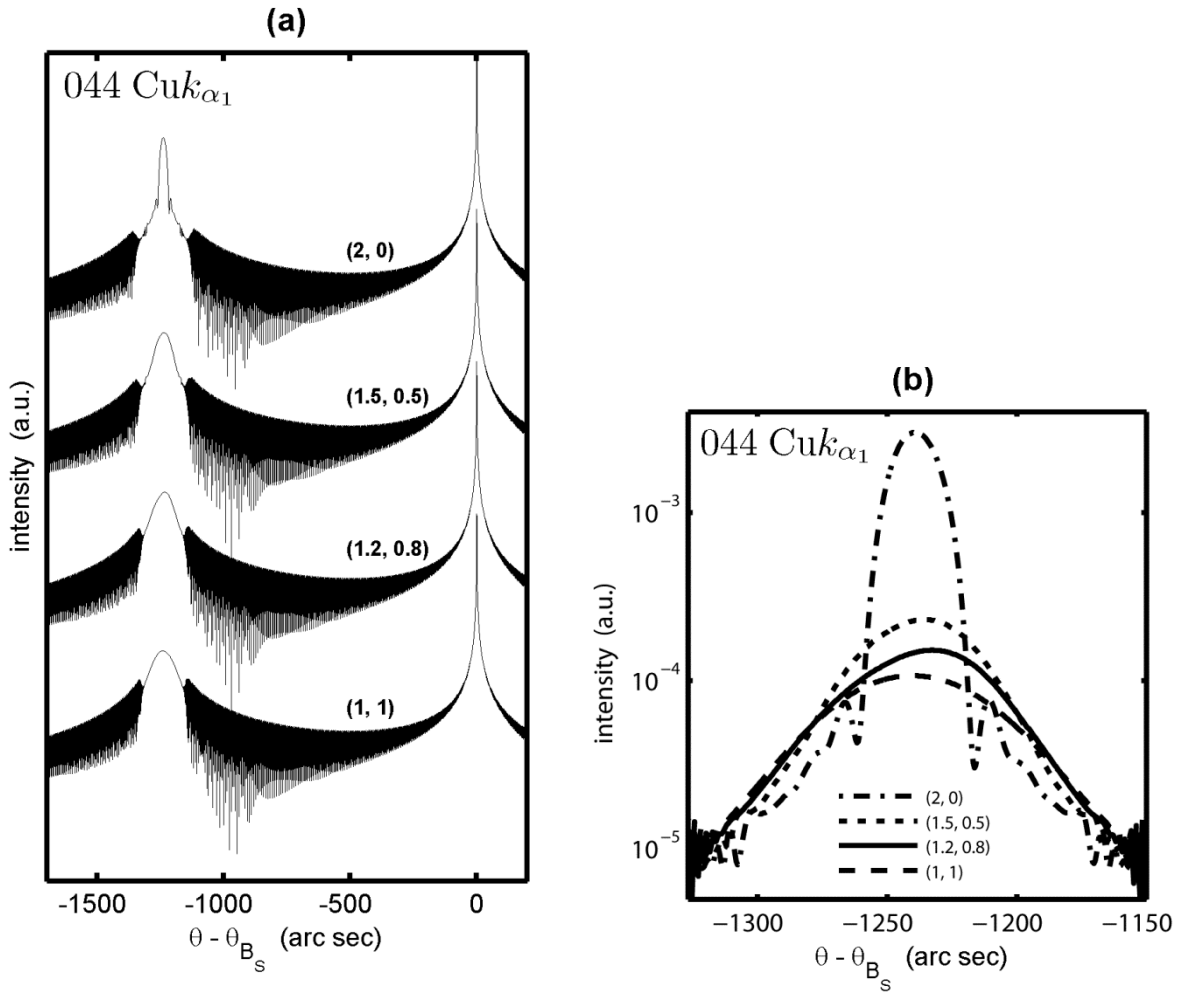

Figure 22. 044 rocking curves for a 1µm $In_{0.07}Ga_{0.93}As$ / GaAs (001) structure with linear dislocation density profiles given by (D_1, D_2) in 10^7 cm^{-2}, for (a) both the substrate and epitaxial Bragg peaks, and (b) detail of the epilayer peak. D_1 is the dislocation density at the substrate interface and D_2 is the dislocation density at the surface.

Given this result, we were interested to see if this would also hold for structures having nonuniform composition. To address this we calculated diffraction profiles for multilayer structures comprising 0.3-μm $In_{0.06}Ga_{0.94}As$ / 0.3-μm $In_{0.07}Ga_{0.93}As$ / 0.3-μm $In_{0.08}Ga_{0.92}As$ / 0.3-μm $In_{0.09}Ga_{0.91}As$ / GaAs (001), with linear depth profiles of dislocation throughout the entire epitaxial structure, and they are given in Figure 23. We calculated rocking curves for linear dislocation profiles (D_1,D_2) of ($2x10^7$, 0), ($1.5x10^7$, $0.5x10^7$), ($1.2x10^7$, $0.8x10^7$), and ($1x10^7$, $1x10^7$) cm^{-2}. Again, the value of D_1 corresponds to the defect density at the substrate interface and D_2 corresponds to that at the surface. For reference, we also show the diffraction profile for the dislocation-free structure $D_1 = D_2 = 0$. We find that the rocking curve varies with the dislocation distribution even though the average dislocation density is fixed in all structures (not including the perfect case). From this we conclude that in principle we can distinguish between the various dislocation distributions through the depth of the epitaxial layers on the basis of experimental measurements.

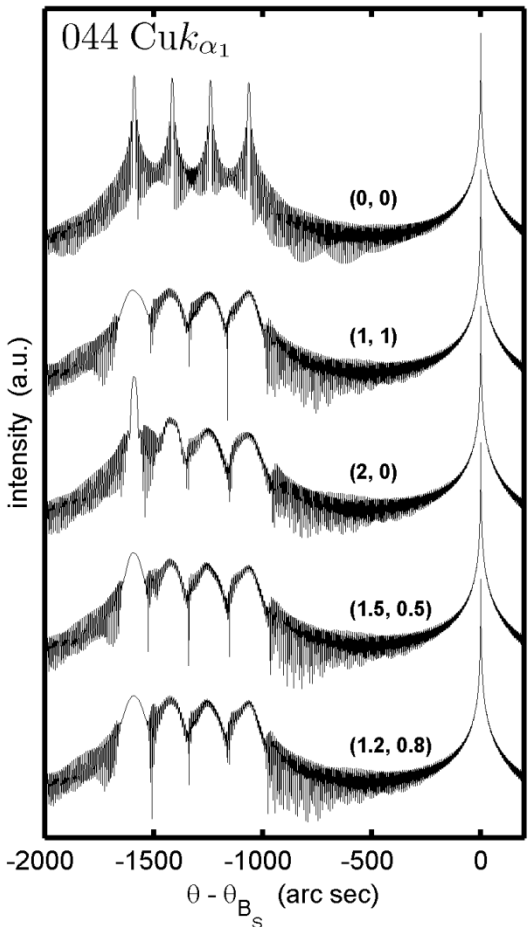

Figure 23. 044 rocking curves for $In_xGa_{1-x}As$ / GaAs (001) multilayer structure with linear dislocation density depth profiles (D_1, D_2) in 10^7 cm^{-2}.

To determine if these findings would hold true in another material system, we then considered a ZnS_xSe_{1-x} / GaAs (001) multilayer structure, comprised of 0.4-µm $ZnS_{0.16}Se_{0.84}$ / 0.4-µm $ZnS_{0.14}Se_{0.86}$ / 0.4-µm $ZnS_{0.12}Se_{0.88}$ / 0.4-µm $ZnS_{0.10}Se_{0.90}$ top layers on a GaAs (001) substrate. These structures, all identical in their compositional profile, were calculated using the phase invariant model with linear dislocation density depth profiles (D_1,D_2) equal to (a) (0,0), (b) ($2x10^7$, 0), (c) ($1.5x10^7$, $0.5x10^7$), (d) ($1.2x10^7$, $0.8x10^7$), and (e) ($1x10^7$, $1x10^7$) cm^{-2}, where D_1 is the dislocation density at the substrate interface and D_2 is the dislocation density at the surface. The 004 diffraction profiles for this structure are given in Figure 24. As we found for the $In_xGa_{1-x}As$ / GaAs (001) material system, we see that the rocking curve varies with the depth profile of dislocation density even though the average dislocation density is

fixed at $1x10^7$ cm^{-2}. This confirms that, the phase invariant model allows us to distinguish between the various dislocation distributions on the basis of experimental measurements.

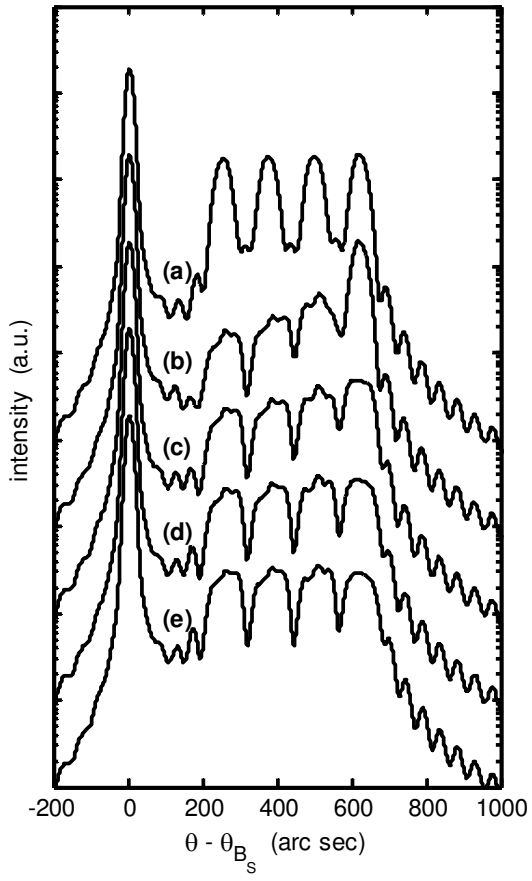

Figure 24. 004 rocking curves for ZnSySe1-y/GaAs (001) multilayers with various linear distributions of dislocation density in which (D1,D2) are equal to (a) (0,0), (b) ($2x10^7$, 0), (c) ($1.5x10^7$, $0.5x10^7$), (d) ($1.2x10^7$, $0.8x10^7$), and (e) ($1x10^7$, $1x10^7$) cm^{-2}.

Next we considered superlattice structures, which are of interest for device applications, control of defect density, and growth characterization experiments. We found that $In_xGa_{1-x}As$ superlattices on GaAs (001) substrates exhibit rather complex diffraction behavior with nonuniform dislocation distributions, even using simple linear profiles for D. Figure 25 shows 044 rocking curves for a 10-period 10-nm $In_{0.03}Ga_{0.97}As$ / 10-nm $In_{0.04}Ga_{0.96}As$ superlattice on GaAs (001) with four different linear profiles of dislocation density in which (D_1,D_2) are equal to (a) (0,0), (b) ($1x10^7$, $1x10^7$), (c) ($2x10^7$, 0), (d) ($1.5x10^7$, $0.5x10^7$), and (e) ($1.2x10^7$, $0.8x10^7$) cm^{-2}.

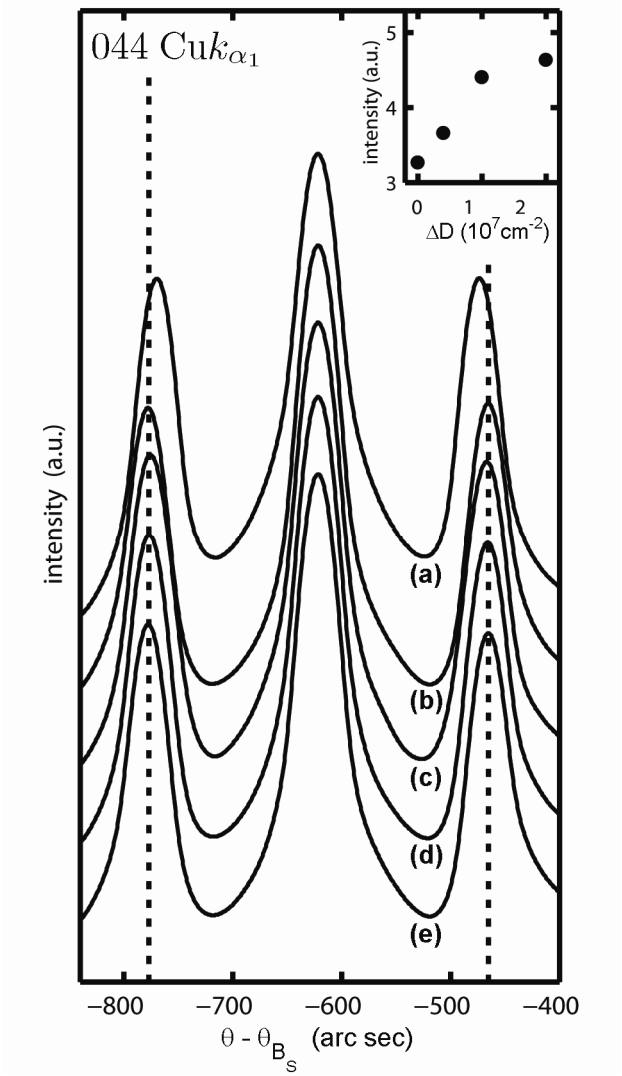

Figure 25. 044 diffraction profiles for a 10-period 10-nm $In_{0.03}Ga_{0.97}As$ / 10-nm $In_{0.04}Ga_{0.96}As$ superlattice on GaAs (001) with different linear distributions of dislocation density in which (D_1, D_2) are equal to (a) (0,0), (b) ($1x10^7$, $1x10^7$), (c) ($2x10^7$, 0), (d) ($1.5x10^7$, $0.5x10^7$), and (e) ($1.2x10^7$, $0.8x10^7$) cm^{-2}. Inset shows average peak intensity *I* of the +1 and -1 superlattice peaks as a function of $\Delta D = D_2 - D_1$; reference angle lines show variation of angle of superlattice peaks.

Much more research will be needed to make general conclusions about the modeling of nonuniform dislocation densities in superlattice structures with the phase-invariant model for dynamical diffraction. This notwithstanding, our findings indicate that the diffraction patterns exhibited by superlattice structures are influenced by the dislocation distribution so that in principle it will be possible to extract the distribution by curve fitting to experimental measurements.

By calculating diffraction profiles from $In_xGa_{1-x}As$ and ZnS_ySe_{1-y} uniform layers, multilayers, and superlattices using the phase-invariant model for dynamical diffraction, we show that the rocking curves

are sensitive to dislocation density, the depth distribution of the dislocation density, as well as composition and strain, indicating that it should be possible to extract all three distributions by the analysis of measured diffraction profiles.

Mosaic Crystal Dynamical Diffraction Model

The phase invariant model for dynamical diffraction presented previously is broadly applicable to semiconductor heterostructures and extends the x-ray analysis to determination of the depth profile of the dislocation density as well as the strain and composition. Although the approximate phase-invariant model is applicable to many metamorphic semiconductor heterostructures, it does not account for phase differences between the crystallites of the mosaic. This leads to incorrect predictions of certain dynamic effects including extinction and Pendellosung interference fringes. The model therefore lacks accuracy in certain structures, such as those containing either closely-lattice matched layers or layers with very high threading dislocation densities. The inaccuracy in some technically important cases of metamorphic structures was the motivation for developing a refined mosaic crystal model which is more generally applicable to semiconductor device structures. The model is more appropriate for step-graded metamorphic semiconductor structures, as well as any other type of semiconductor heterostructure containing closely-lattice matched layers or dislocation densities higher than 10^8 cm^{-2}. Additionally, the model is significantly more computationally efficient, so complex structures with many sublayers or high dislocation densities can be computed in much less time. Here we present the physical and computational details of the mosaic crystal model, and we demonstrate its application in step-graded $In_xGa_{1-x}As/GaAs$ (001) and $In_xAl_{1-x}As/GaAs$ (001) device structures.

Theory

As mentioned previously in Section Phase-Invariant Dynamical Diffraction Model, an imperfect crystal is distorted by the presence of dislocations in two key ways which affect the diffraction profile, and these effects may be modeled by treating the crystal as a mosaic of uniform blocks. First, the mosaic blocks exhibit angular variations, which cause diffraction at different angles relative to the crystal surface. Second, localized strain about the dislocations gives rise to variations in the spacing of the diffracting planes, which alters the Bragg angles for the blocks. These mosaic block variations each bear statistical

distributions, which we define, for angular and interplanar spacing, as $P_\alpha(\theta)$ and $P_\varepsilon(\theta)$, respectively. The general approach two the PIDDM and MCDDM are as follows: in the phase invariant model, the angular and mosaic spread $P_\alpha(\theta)$ and $P_\varepsilon(\theta)$ are integrated to the deviation parameter, and an effective deviation parameter is computed through the double integration with respect to these distributions, and the rocking curve is computed with this effective deviations parameter. In the mosaic crystal model, the scattering amplitude is calculated for every mosaic block in the crystal, and the intensities are weighted and summed.

More specifically, the structure is considered to comprise a mosaic of $N_\alpha \times N_\beta$ crystallites in α and β space. The dynamic scattering amplitude is computed for each of these crystallites iteratively. For the n[th] lamina of one such crystallite, the deviation parameter is

$$\eta_{nij} = \frac{-(\gamma_0/\gamma_H)(\theta-\theta_{Bn}+\alpha_i-\beta_j)\sin(2\theta_{Bn})-0.5(1-\gamma_0/\gamma_H)\Gamma F_{0n}}{\sqrt{|\gamma_0/\gamma_H|}C\Gamma\sqrt{F_{Hn}F_{\bar{H}n}}}, \tag{31}$$

where $\alpha_i = -N_\sigma\sigma_\alpha + iN_\alpha\sigma_\alpha/2N_\sigma$ and $\beta_j = -N_\sigma\sigma_\beta + jN_\beta\sigma_\beta/2N_\sigma$, where N_σ is the number of standard deviations used in the two distributions and i and j are integers. The scattering amplitudes are calculated iteratively by

$$X_{nij} = \eta_{nij} + \sqrt{\eta_{nij}^2-1}\frac{(S_{1nij}+S_{2nij})}{S_{1nij}-S_{2nij}}, \tag{32}$$

where

$$S_{1nij} = \left(X_{nij-1}-\eta_{nij}+\sqrt{\eta_{nij}^2-1}\right)\exp\left(-iT_n\sqrt{\eta_{nij}^2-1}\right) \tag{33}$$

and

$$S_{2nij} = \left(X_{nij-1}-\eta_{nij}+\sqrt{\eta_{nij}^2-1}\right)\exp\left(iT_n\sqrt{\eta_{nij}^2-1}\right), \tag{34}$$

η_n is the deviation parameter and the thickness parameter is

$$T_n = h_n \frac{\pi \Gamma \sqrt{F_{Hn} F_{\bar{H}n}}}{\lambda \sqrt{|\gamma_0 \gamma_H|}}, \tag{35}$$

where h_n is the thickness and F_{0n}, F_{Hn} and $F_{\bar{H}n}$ are the 000, hkl, and $\bar{h}\bar{k}\bar{l}$ structure factors for the nth sublayer. The x-ray diffraction profile is calculated by adding the weighted intensity contributions from the $N_\alpha \times N_\beta$ crystallites in α and β space with

$$I = \sum_i \sum_j \left| X_{Nij} \right|^2 \cdot W_{\alpha i} \cdot W_{\varepsilon j}, \tag{36}$$

where the weighting functions are given by

$$W_{\alpha i} = \exp\left(-\alpha_i^2 / 2\sigma_\alpha^2\right), \text{ and} \tag{37}$$

$$W_{\varepsilon i} = \exp\left(-\beta_i^2 / 2\sigma_\varepsilon^2\right), \tag{38}$$

and, as before, the standard deviations related to the dislocation density as

$$\sigma_\alpha = b\sqrt{\pi D}/2, \text{ and} \tag{39}$$

$$\sigma_\varepsilon = 0.127 b \sqrt{D \left| \ln\left(2 \times 10^{-7} cm \sqrt{D}\right) \right|} \tan\theta_B . \tag{40}$$

This approach accounts for the incoherency of the phase within crystallites of the mosaic, and strictly speaking is a more physically accurate model of diffraction within a distorted crystal with mosaic block angular misorientations and interplanar variations. The mosaic crystal model may be applied to semiconductor heterostructures with arbitrary profiles of composition and dislocation density.

Results

We have applied the mosaic crystal model to calculate x-ray diffraction profiles for step-graded $In_xGa_{1-x}As/GaAs$ (001) structures. The x-ray wavelength was used corresponded to the Cu $k\alpha_1$ radiation line, $\lambda = 0.1540594$ nm. The layers of the step-graded structure were assumed to have uniform dislocation density. Layers with constant composition may be broken up into a series of sublayers for the purpose of x-ray simulation. In this work we chose $N_\sigma = 3$ because we found that a larger number of standard deviations did not provide an appreciable increase in accuracy. For implementation of the mosaic crystal model, we set a lower limit on the number of crystallites used as $N_\alpha = 21$ and $N_\beta = 21$, resulting in a model involving 441 crystallites. The required number of crystallites for accurate modeling depends on the maximum dislocation density in the structure, and therefore the best practice is to increase the number of crystallites until the diffraction profile exhibits no change. For structures with an assumed dislocation density of D = 10^8 cm^{-2}, we found that $N_\alpha = N_\beta = 31$ was necessary. Though it is possible to simulate instrumental effects through convolution with a broadening function, here we present unbroadened rocking curves.

The first structure we considered contains 1.0-μm step-graded layers of $In_xGa_{1-x}As/GaAs$ (001), with the composition graded from 0% to 50% indium in five steps. The 004 diffraction profiles for this structure with five different assumed values of the dislocation density, and the perfect case, is given in Figure 26. Although five diffraction peaks are easily resolved in the dislocation-free structure, these diffraction peaks broaden and begin to overlap as the dislocation density increases. In addition, the Pendellosung begin to extinguish in cases where the dislocation density exceeds 10^7 cm^{-2}.

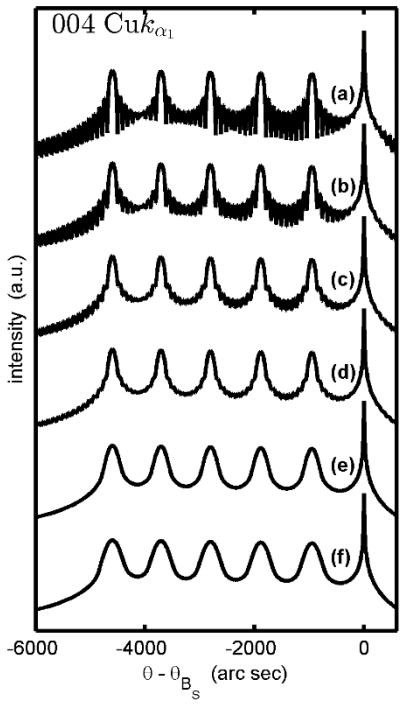

Figure 26. 004 Diffraction Profile for Step Graded InGaAs/GaAs (five steps) with Dislocation Density of (a) zero, (b) 10^6 cm^{-2}, (c) 5 x 10^6 cm^{-2}, (d) 10^7 cm^{-2}, (e) 5 x 10^7 cm^{-2}, and (f) 10^8 cm^{-2}.

Next, we investigated step-graded structures with ten steps instead, but with the same thickness and final composition as the previous structure. The 004 diffraction profiles for these structures are given in Figure 27 for five uniform dislocation densities, as well as the perfect case. Here we see that the ten diffraction peaks associated with the compositions in the step-graded structure become increasingly difficult to resolve with higher dislocation density. Again we also note the washout of Pendellosung with increasing D.

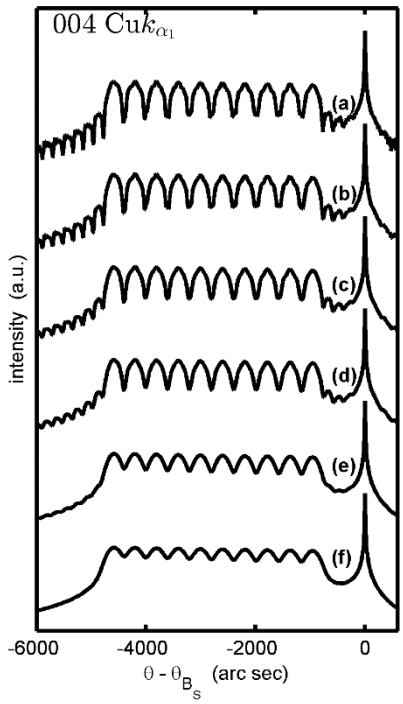

Figure 27. 004 Diffraction Profile for Step Graded InGaAs/GaAs (ten steps) with Dislocation Density of (a) zero, (b) 10^6 cm^{-2}, (c) 5 x 10^6 cm^{-2}, (d) 10^7 cm^{-2}, (e) 5 x 10^7 cm^{-2}, and (f) 10^8 cm^{-2}.

Finally we considered the case of nonuniform dislocation density. To study this, we considered 1.0-µm step-graded layers of $In_xGa_{1-x}As/GaAs$ (001) with the composition graded from 0% to 50% indium in five steps, and with the dislocation density decreased stepwise, in a linear fashion, in the step-graded layers. The dislocation density in the first layer (the layer contacting the substrate) was assumed to be 10^8 cm^{-2} in all cases, and the dislocation density in the succeeding layers decreased linearly to a top layer value of (a) 10^8, (b) 5x10^7, (c) 10^7, (d) 5x10^6, (e) 10^6, and (f) 0 cm^{-2} as indicated in Figure 28. Differences are noted in the diffraction from each step in the graded buffer, and as in the uniform case, we observe Pendellosung in layers containing the lower dislocations densities, and the fringes become extinguished as D increases. These results indicate that the x-ray diffraction profile curve is sensitive to the profile of the dislocation density, as well as the average value.

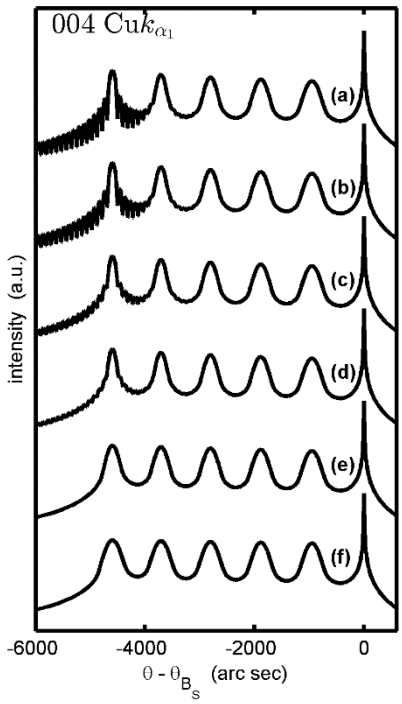

Figure 28. 004 Diffraction Profile for Step Graded InGaAs/GaAs (five steps) with D = 10^8 cm^{-2} at the interface varied stepwise linearly to (a) zero, (b) 10^6 cm^{-2}, (c) 5 x 10^6 cm^{-2}, (d) 10^7 cm^{-2}, (e) 5 x 10^7 cm^{-2}, and (f) 10^8 cm^{-2} at the top layer. (Case (f) represents a uniform dislocation density.)

Much more research will be needed for a comprehensive evaluation of mosaic crystal model and its application to experimental structures. As an initial study, we calculated rocking curves for a heterostructure grown by Jiang et al.[29] in a 2008 work. The structure comprises a GaAs substrate, a 900nm overshoot step-graded $In_xAl_{1-x}As$ buffer with nine steps from x=0.05 to x=0.85, and a 1-µm $In_{0.75}Al_{0.25}As$ top layer (see the lattice profile for this structure in Figure 29). For the purpose of simulation, the structure was assumed to be completely relaxed, as suggested by Jiang et al. Shown in Figure 30 is the experimental 004 rocking curve along with that calculated using the mosaic crystal model, and the top layer was assumed to have a uniform dislocation density of (a) 10^9 cm^{-2} and (b) 2x10^8 cm^{-2} in the top layer, and the entire step-graded buffer was calculated with D = 10^9 cm^{-2} in both cases.

In the case where the dislocation density is assumed to be 10^9 cm^{-2}, the predicted profile of the 1-µm $In_{0.75}Al_{0.25}As$ layer exhibits a width (~600 arc seconds) and a normalized intensity (normalized by dividing by the substrate peak intensity, ~0.06) that is in excellent agreement to the experimental profile

as shown in Figure 30(a). On the other hand, if the top $In_{0.75}Al_{0.25}As$ layer is assumed to have a lower dislocation density then the peak width and intensity will be over and underestimated, respectively. As an example, Figure 30(b) shows the case in which the top layer is assumed to have a dislocation density of D = 2 x 10^8 cm^{-2}. The resulting width and normalized intensity are 280" and 0.15, respectively. These observations suggest that the actual dislocation density in the top uniform layer is approximately 10^9 cm^{-2}.

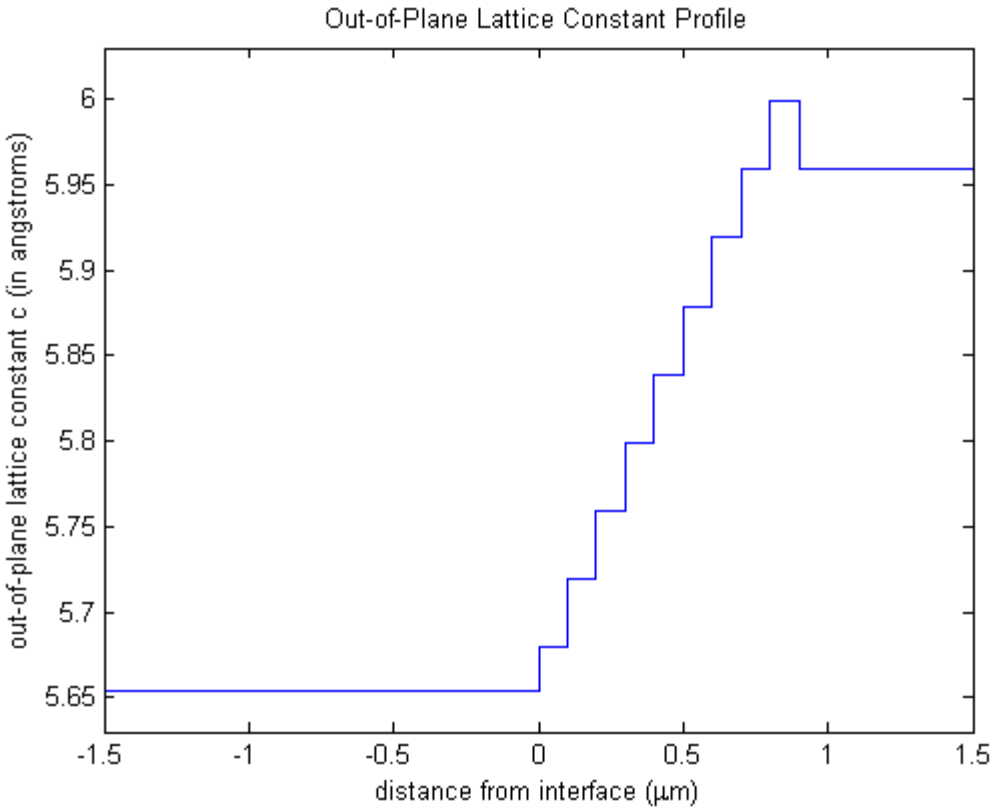

Figure 29. Lattice profile for the $In_xAl_{1-x}As$ / GaAs (001) structure grown by Jiang et al.[29]

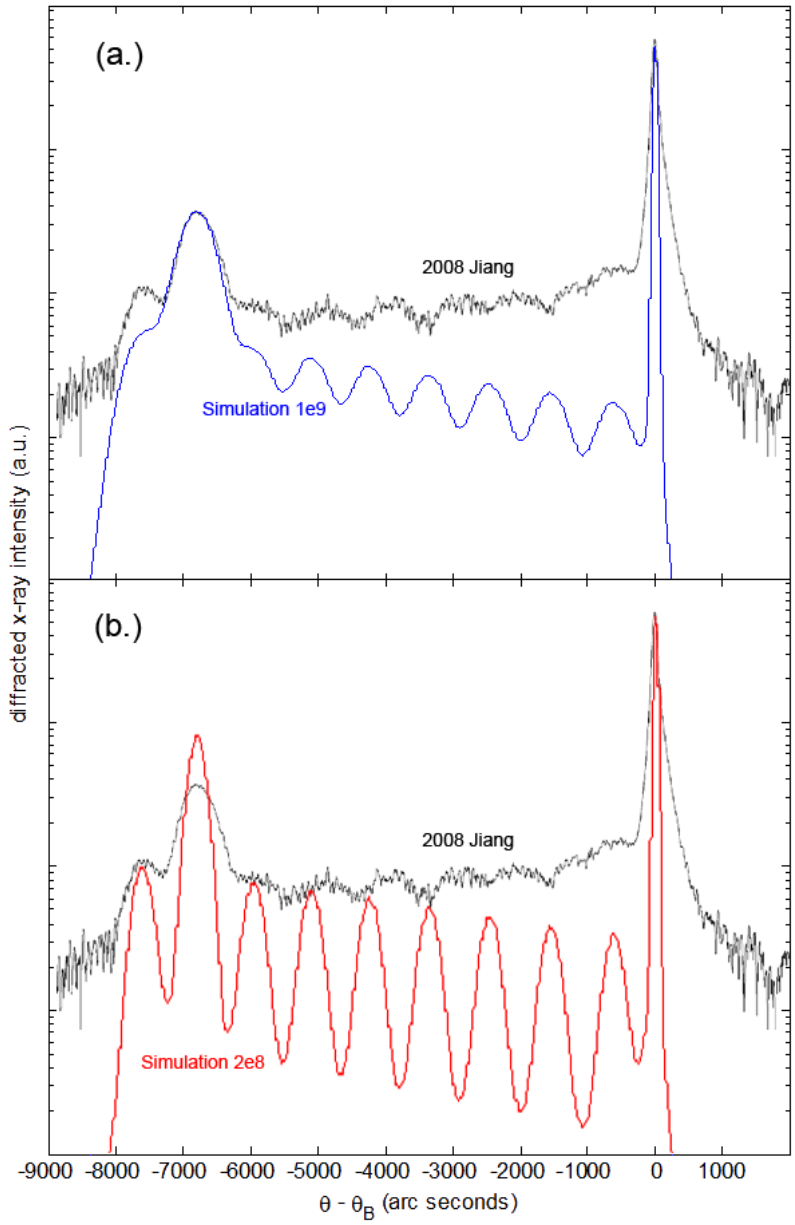

Figure 30. Comparison of an experimentally measured 004 diffraction profile[29] to diffraction profiles calculated with the mosaic crystal model. The dislocation density in the top layer assumed to be (a) D = 10^9 cm^{-2} and (b) D = $2x10^8$ cm^{-2}, and the dislocation density in the graded buffer was assumed to be 10^9 cm^{-2} in both cases.

In the graded buffer, on the other hand, neither calculated diffraction profile achieves good agreement with the measurement. Firstly, the experimental profile exhibits shifts in the peak positions. Perhaps the most obvious explanation for this is the presence of residual strain in the crystal sample, though Jiang et al. concluded it was completely relaxed. However, the peak shifts are not uniform, that is,

each step in the buffer is shifted by a different amount. This could indicate that the depth profile of residual strain that is also nonuniform. If this is the correct explanation, the layers closer to the substrate appear to include tensile in-plane strain; however, the mismatch strain in the structure is compressive. Another possible explanation is the tolerance on compositional control during growth. If the layers closer to the substrate were indium-deficient compared to their target composition, then this could cause the observed peak shift. Finally, it is also possible that crystallographic tilting of the layers caused the peak shift. Tilted epitaxial growth is known to occur, with several mechanisms of action proposed[30]. It is possible that the steps in the graded buffer contain different tilts, shifting their Bragg peaks accordingly. To separate these three effects—residual strain, compositional tolerance, and epitaxial tilt—one could measure asymmetric hkl reflections and vary the azimuth. Another disagreement between the simulation and experiment is the difference in peak intensities of the peaks associated with the graded buffer. The noise floor of the experimental data could explain this in part, but not entirely. Also, unexpected variations in the growth rate would lead to different thicknesses of the steps, and this could help explain the differences observed in intensity.

In applying the mosaic crystal dynamical diffraction model to $In_xGa_{1-x}As$/GaAs (001) and $In_xAl_{1-x}As$/GaAs (001) structures, we demonstrate the usefulness of the mosaic crystal model in the estimation of threading dislocation densities as well as compositions and strains, as well as their depth profiles, in device structures.

Comparison of Phase-Invariant and Mosaic Crystal Models

Here we compared the phase-invariant and mosaic crystal models by calculating x-ray rocking curves for several $In_xGa_{1-x}As$ /GaAs (001) heterostructures of interest. In all simulations, the x-ray wavelength used corresponds to the Cu $K\alpha_1$ radiation line, $\lambda = 0.1540594$ nm. In this work layers linearly graded layers were approximated by 30 sublayers. We used $N_\sigma = 3$ in the implementation of both models, and for implementation of the mosaic crystal model, we used $N_\alpha = N_\beta = 21$, resulting in a model involving 441 crystallites. To account for typical instrumental effects of the diffractometry setup, we present rocking curves broadened by a Gaussian function with a FWHM of 12 arc seconds, which represents the divergence of a 2-crystal, 4-reflection Ge(220) Bartels monochromator[1, 2].

First we considered 1.0-µm, uniform-composition layers of $In_{0.05}Ga_{0.95}As$ / GaAs (001) with various, uniform values of threading dislocation density D. The 004 diffraction profiles for these structures are given in Figure 31, calculated using (a) the phase-invariant model and (b) the mosaic crystal model.

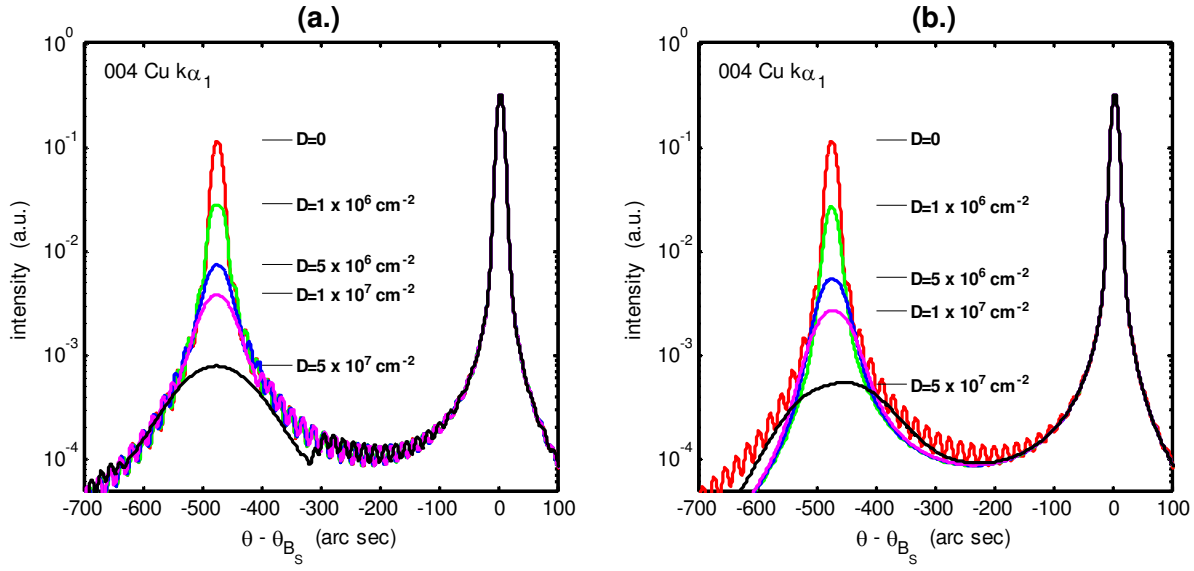

Figure 31. 004 Rocking curves for the 1.0-µm $In_{0.05}Ga_{0.95}As$/GaAs (001) structure calculated with the (a) phase invariant and (b) mosaic crystal models.

It is seen that the two models predict slightly different peak intensities for the epilayer. Also, the mosaic crystal model predicts the washing out of interference fringes as dislocation density increases above 10^6 cm^{-2}, while the phase-invariant model predicts the interference fringes for all calculated dislocation densities, despite the fact that experimental measurements show these fringes only in cases of extremely low dislocation densities. To demonstrate this more clearly, these same data are presented differently in Figure 32, wherein the two models are overlaid.

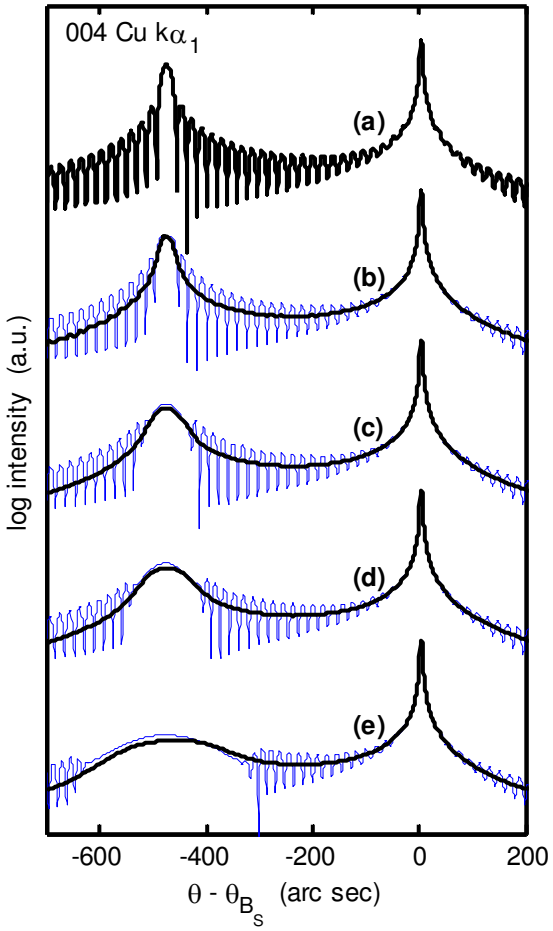

Figure 32. 004 Rocking curves for the 1.0-µm $In_{0.05}Ga_{0.95}As/GaAs$ (001) structure with calculated with the phase-invariant model (blue) and the mosaic crystal model (black), with assumed dislocation densities of (a) D = 0, (b) 10^6 cm^{-2}, (c) $5x10^6$ cm^{-2}, (d) 10^7 cm^{-2}, and (e) $5x10^7$ cm^{-2}.

A comparison of the widths of the epilayer Bragg peak as predicted by the phase-invariant model, mosaic crystal model, and convolution model[2] is given in Figure 33. The rocking curve width associated with the convolution model was found using equation (29).

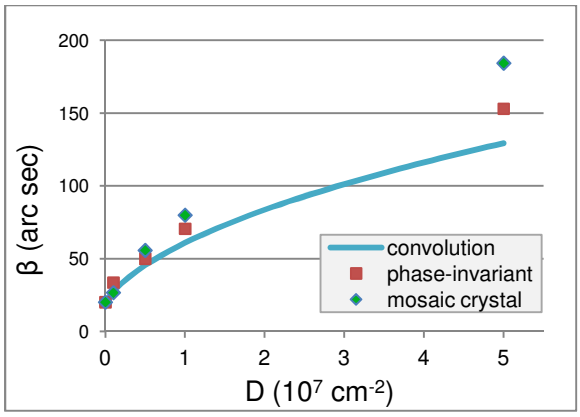

Figure 33. Comparison of epilayer peak width as predicted by the phase-invariant model, the mosaic crystal model, and the convolution model[2].

All three models predict similar rocking curve widths, within 20%, so that any of the three models provide a fair prediction of the peak width for a uniform layer with moderate lattice mismatch. However, as illustrated in Figure 32, the mosaic crystal model is preferred for these structures because it correctly predicts the weakening of the interference fringes with increasing dislocation density, and this illustrates the importance of accounting for phase differences between crystallites. This also suggests that the mosaic crystal model will provide better accuracy in epitaxial layer peak intensity.

We expected that these considerations are even more important in structures with closely-lattice matched layers. To study this, we considered 1.0-µm thick $In_{0.01}Ga_{0.99}As$ / GaAs (001) layers with various values of threading dislocation density, and calculated the 004 rocking curves using both models. Figure 34 gives the rocking curves as calculated with (a) the phase-invariant model and (b) the mosaic crystal model. In this case of closely-lattice matched (to the substrate) layers, the phase-invariant model incorrectly predicts an extinguishing of the substrate intensity ("peak washout") with high threading dislocation densities. This result is nonphysical, as the substrate peak is almost always present in real x-ray measurements, and indicates that the phase-invariant model accounts for extinction incorrectly in structures with closely-lattice matched materials and high dislocation densities. The mosaic crystal model does not exhibit this issue, and appears to correctly predict intensity in cases with close-lattice matching.

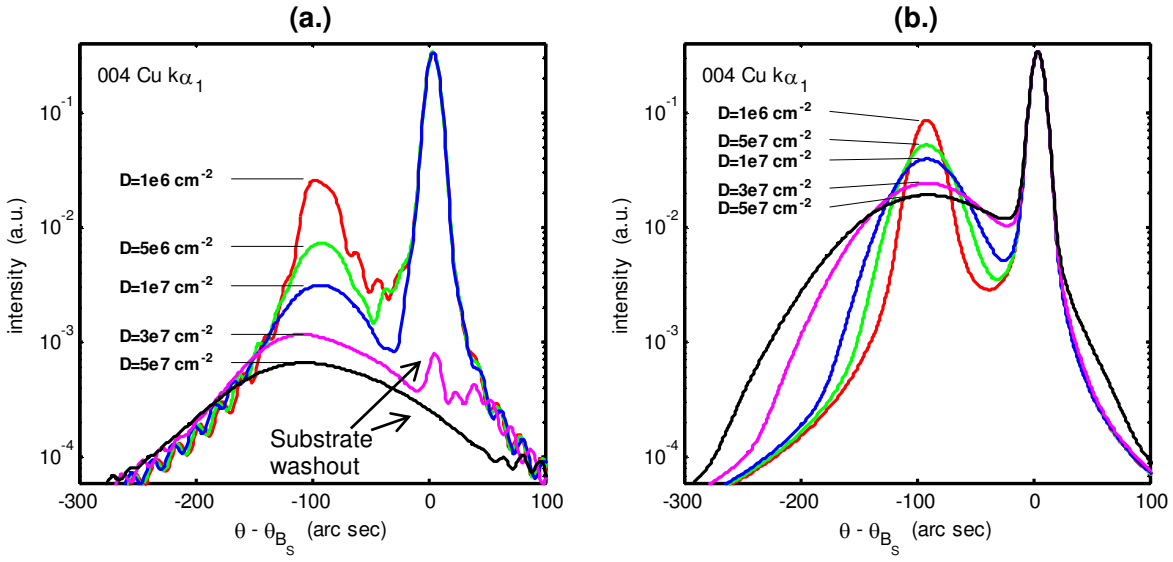

Figure 34. 004 Rocking curves for the closely-lattice matched 1µm uniform $In_{0.01}Ga_{0.99}As$/GaAs (001) structure calculated with (a) the Phase Invariant Model and (b) the Mosaic Crystal Model.

After seeing this, we then questioned whether and how the peak washout might appear in graded structures, which are commonly employed in metamorphic devices for the control of the dislocation density, strain relaxation, and crystallographic tilting[31-38]. We considered step-graded $In_xGa_{1-x}As$ / GaAs (001) structures in which the $In_xGa_{1-x}As$ buffer was 2.0 µm-thick and contained five equal-thickness layers (steps) with indium compositions of 1%, 3%, 5%, 7%, and 9%, with the 1% layer at the substrate interface and the 9% layer at the surface. Figure 35(a) and 5(b) show the predictions of the phase-invariant and mosaic crystal models for cases of various uniform dislocation densities as indicated in the figures. In the case with $D=5x10^7$ cm^{-2}, we again see that the substrate peak is completely extinguished in the phase-invariant model. As well, there is a distortion of the shape of the diffraction peaks from all but the top and bottom layers when using the phase-invariant model. We also note that the epilayer peak intensities are incorrectly predicted by the phase-invariant model.

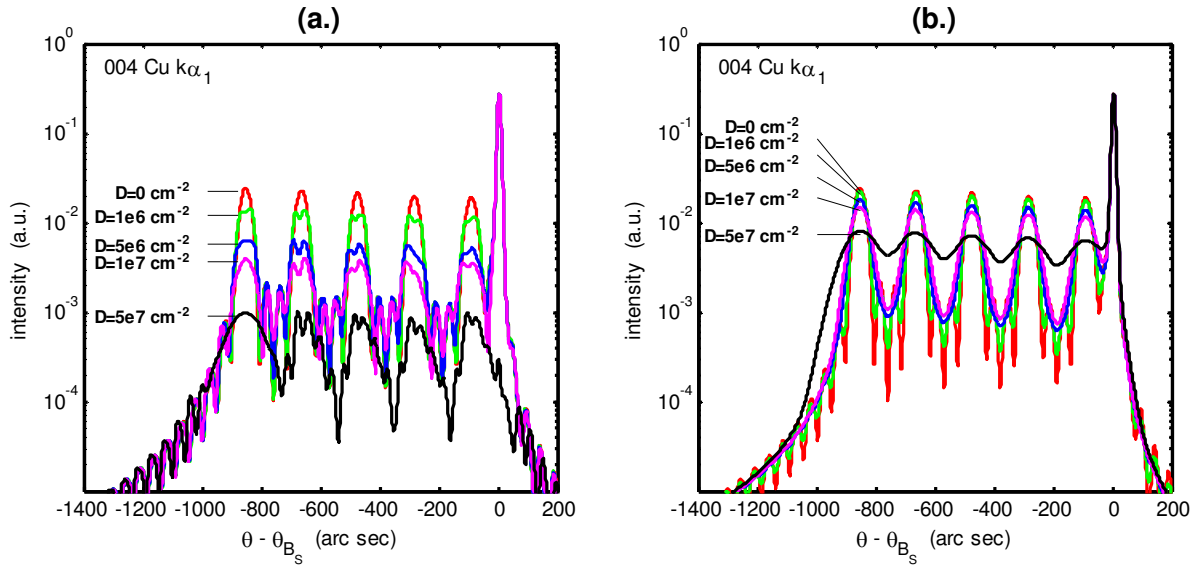

Figure 35. 004 Rocking curves for the step-graded 2.0-µm $In_xGa1-xAs/GaAs$ (001) structure calculated with (a) the Phase Invariant Model and (b) the Mosaic Crystal Model.

Finally, we wanted to see how these conclusions would hold when applied to linearly graded structures. We considered a 2.0-µm $In_xGa_{1-x}As$ graded buffer on a GaAs (001) substrate, with x graded from 0 to 10%. We calculated the diffraction profiles for this structure for both models and various dislocation densities, depicted in Figure 36. In the phase-invariant model, the substrate peak becomes washed out with higher dislocation densities, and interestingly much of the intensity of the graded buffer is also extinguished. In the mosaic crystal model, no such washout is observed and the rocking curves are qualitatively in agreement with experimental measurements for structures of this type.

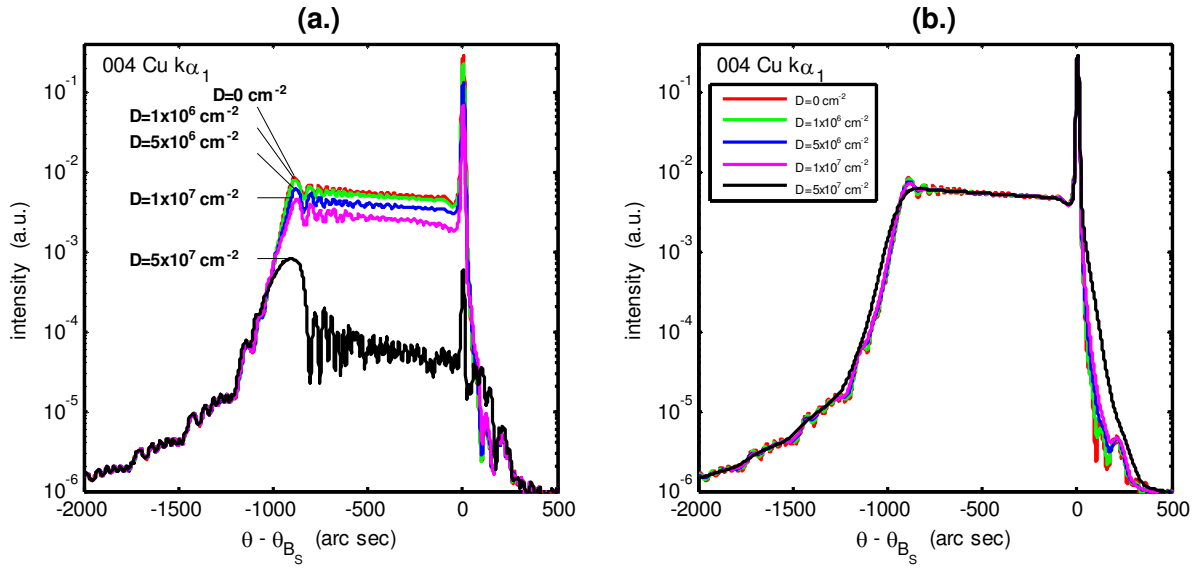

Figure 36. 004 Rocking curves for the linearly-graded 2µm $In_xGa1-xAs/GaAs$ (001) structure calculated with (a) the Phase Invariant Model and (b) the Mosaic Crystal Model.

Conclusions

We have developed two new dynamical models for Bragg x-ray diffraction from arbitrary semiconductor heterostructures with nonuniform composition, strain, and dislocation density. The models are based on the Takagi-Taupin equation for distorted crystals and account for the angular and strain variations in the lattice induced by the presence of dislocations.

We demonstrate dynamical diffraction from perfect crystals by computing rocking curves from $Si_{1-x}Ge_x$ / Si (001) structures. We then calculate diffraction profiles for symmetric and asymmetric $In_xGa_{1-x}As$ / GaAs (001) , and ZnS_ySe_{1-y} / GaAs (001) heterostructures, including uniform layers, step-graded and linearly-graded buffer layers, multilayers and superlattices, using both the phase-invariant and mosaic crystal models. In so doing, we show that the diffraction profiles are strongly affected by the depth profiles of dislocation density, as well as composition and strain, and demonstrate the application of the models for the estimation of these three distributions in device structures. We also compare the mosaic crystal model to experimental measurements of an $In_xAl_{1-x}As$ / GaAs (001) heterostructure and through a curve-fitting process conclude that the top layer dislocation density is likely about 10^9 cm^{-2}. Finally, we compare the two models, demonstrating that the mosaic crystal model offers several improvements over the phase-invariant model in the prediction of x-ray diffraction profiles. We show that, by accounting for phase differences between crystallites, the mosaic crystal model gives improved predictions of intensities and Pendellosung interference features. While both models can be applied to determine the depth profiles of strain, composition, and defect density in device structures, we show that in certain cases, such as closely-lattice matched structures and those with very high dislocation density, the mosaic crystal model is preferred.

In this work we have assumed Gaussian distributions for the angular and strain-broadening of dislocations, but the dynamical models presented may be readily applied using other angle-scale distributions, which could be determined either experimentally or by use of the Krivoglaz & Ryaboshapka[24] theoretical framework within assumed dislocation structures.

Future work could include the development of curve-fitting algorithms for computationally efficient determination of the depth profile of strain, composition, and dislocation density. Also, this work could be readily adapted to compute the ω-2θ scan as well as the θ scan, for which the model is currently written. Finally, we would like to extend the mosaic crystal model for the computation of reciprocal space maps, which have increasing importance in the field of x-ray diffraction characterization.

References

1. W. J. Bartels, J. Vac. Sci. Technol. B 1, 338 (1983).

2. J. E. Ayers, J. Cryst. Growth 135, 71 (1994).

3. M. A. G. Halliwell, M. H. Lyons and M. J. Hill, J. Cryst. Growth 68, 523 (1984).

4. W. L. Bragg, Proceedings of the Cambridge Philosophical Society 17, 43 (1913).

5. S. Takagi, Acta Cryst. 15, 1311 (1962).

6. D. Taupin, C. R. Acad. Sci. 256, 4881 (1963).

7. S. Takagi, J. Phys. Soc. Jpn. 26, 1239 (1969).

8. C. R. Wie, T. A. Tombrello and T. Vreeland, Jr. , J. Appl. Phys. 59, 3743 (1986).

9. W. J. Bartels, J. Hornstra and D. J. Lobeek, Acta. Cryst. A42, 539 (1986).

10. V. S. Speriosu, J. Appl. Phys. 52, 6094 (1981).

11. V. S. Speriosu and T. Vreeland, Jr. , J. Appl. Phys. 56, 1591 (1984).

12. L. Tapfer and K. Ploog, Phys. Rev. B 40, 9802 (1989).

13. C. R. Wie, J. Appl. Phys. 65, 1036 (1989).

14. C. R. Wie and H. M. Kim, J. Appl. Phys. 69, 6406 (1991).

15. J. A. Prins, Z. Phys. 63, 477 (1930).

16. L. Vegard, Z. Phys. 79, 1602 (1921).

17. J. A. Ibers and W. C. Hamilton, International Tables for X-Ray Crystallography, (Kluwer, 1989), p. 99.

18. J. E. Ayers, Heteroepitaxy of Semiconductors: Theory, Growth, and Characterization, 1st edn. (CRC Press, Boca Raton, FL, 2007), p. 262.

19. W. J. Bartels, Philips Tech. Rev. 41, 183 (1983/84).

20. R. H. Saul, J. Appl. Phys. 40, 3273 (1969).

21. M. S. Abrahams, L. R. Weisberg, C. J. Buiocchi and J. Blanc, J. Mater. Sci. 4, 223 (1969).

22. N. A. El-Masry, J. C. Tarn and N. H. Karam, J. Appl. Phys. 64, 3672 (1988).

23. P. B. Rago and J. E. Ayers, J. Electron. Mater. 42, 2450 (2013).

24. M. A. Krivoglaz and K. P. Ryaboshapka, Fiz. Metal. Metalloved 15, 18 (1963).

25. L. E. Levine and R. Thomson, Acta Cryst. 53, 590 (1997).

26. P. Gay, P. B. Hirsch and A. Kelly, Acta Met. 1, 315 (1953).

27. M. J. Hordon and B. L. Averbach, Act Met. 9, 237 (1961).

28. A. N. Ivanov, P. Klimanek and A. M. Polyakov, Metal Sci. Heat Tret. 42, 299 (2000).

29. Z. Jiang, W. Wang, H. Gao, L. Liu, H. Chen and J. Zhou, Appl. Surf. Sci. 254, 5241 (2008).

30. J. E. Ayers, S. K. Ghandhi and L. J. Schowalter, J. Cryst. Growth 113, 430 (1991).

31. D. E. Grider, S. E. Swirhun, D. H. Narum, A. I. Akinwande, T. E. Nohava, W. R. Stuart and P. Joslyn, J. Vac. Sci. Technol. B, 8, 301 (1990).

32. M. Haupt, K. Kohler, P. Ganser, S. Emminger, S. Muller and W. Rothemund, Appl. Phys. Lett. 69, 412 (1996).

33. J.-F. He, H.-L. Wang, X.-J. Shang, M.-F. Li, Y. Zhu, L.-J. Wang, Y. Yu, H.-Q. Ni, Y.-Q. Xu and Z.-C. Niu, J. Phys. D: Appl. Phys. 44, (2011).

34. J. Maws, J. D. Kirch, C.-C. Chang, T. Kim, T. Garrod, D. Botez, S. Ruder, T. F. Kuech, T. Earles, R. Tatavarti, N. Pan and A. Wibowo, J. Cryst. Growth 370, 230 (2012).

35. K. Inoue, J. C. Harmand and T. Matsuno, J. Cryst. Growth 111, 313 (1991).

36. G. B. Galiev, I. S. Vasil'evskii, S. S. Pushkarev, E. A. Klimov, R. M. Imamov, P. A. Buffat, B. Dwir and E. I. Suvorova, J. Cryst. Growth 366, 55 (2013).

37. A. Zakaria, R. R. King, M. Jackson and M. S. Goorsky, J. Appl. Phys. 112, 024907 (2012).

38. J.-H. Tsai, J. Electrochem. Soc. 158, 889 (2011).

Appendix A: Config Files

An important concept utilized in this MATLAB program is that of config files. These are files that the user makes to specify (up to) everything about a simulation, such that it can be run entirely without user interaction. This particularly useful in the case of using multisim or some adaptation thereof which runs a batch of simulations.

It is not necessary to use a config file. If none is used, the user must answer every question about the structure and simulation one-by-one. This method would be good to use in exploring the program's features and capabilities, and for new users. However, once the program is well understood, use of config files is strongly recommended. Besides convenience, they provide self-documentation for what specifically was run.

Syntax

The format for the config files is given in the example file `config_example.m`. See the source code for this file in Appendix B: MATLAB Source Code.

Usage

Config files may have any filename that MATLAB approves of for a function. The filename of the config file (minus the ".m" is passed as an argument to `xrd_sim`. The program `xrd_sim` also allows one pair of Property and Value passed after the config file, and this, like the first argument of config file, is optional. Supported (Property, Value) pairs that can be passed to the function `xrd_sim` are `azimuth`, `TDDuniform`, `TDDinitial`, `TDDfinal`, and `NalphaARG`.

List of Configuration Items

For a list of the configuration items, see `config_choices.m`, the contents of which are given in Appendix B: MATLAB Source Code. It documents all the possible choices that may be specified in a config file for setting up a simulation. If using a config file and all of the relevant items listed here are *not* specified, `xrd_sim` will prompt the user at runtime. Prompts will interrupt a "multisim" as they require user intervention.

Appendix B: MATLAB Source Code

Following is the list of MATLAB ".M-files", referred to here as functions, which comprise the program. The main program is `xrd_sim.m` and `xrd_sim_PIDDM.m`.

Summary of Functions

The following list of functions and their descriptions was generated automatically by the script `MakeMfileList.m`. This script can be run again as needed to regenerate this list for documentation purposes.

M-FILE LISTING

1) FWHMdata.m
2) asf_calc.m
3) asf_calc2.m
4) asf_coefficients.m
5) cauchypdf.m
6) closewindows.m
7) config.m
8) config_choices.m
9) config_example.m
10) elasticconstant_calc.m
11) fwhm.m
12) instr_broaden.m
13) latticeparam_calc.m
14) makeMfileList.m
15) multisimD.m
16) myfwhm.m
17) nonlinspace.m
18) num2clip.m
19) parallel.m
20) plotscolor.m
21) rockingcurve_plot.m
22) sf_calc.m
23) str2clip.m
24) tripdf.m
25) xrd_sim.m
26) xrd_sim_PIDDM.m

FILE DESCRIPTIONS

1) FWHMdata
Calculate full width at half maximum (FWHM) in a subrange of a formatted dataset made by xrd_sim.
automatically removes header from the XRDdata cell and converts cell to a matrix for use by myfwhm

2) ASF_CALC
Calculate atomic scattering factor (ASF)
Function calculates the analytical approximation to the x-ray ASF with
anomalous dispersion correction. Anomalous dispersion correction can be
included by setting deltaf_real and deltaf_imag to appropriate values
(use zero to neglect anomalous dispersion).

Input Variables:
> coeffs = {a b c deltaf_real deltaf_imag}
> x = sin(theta) / lambda
> radiationChoice = 1,2,3,4,5 for CrKalpha, FeKalpha, CuKalpha, MoKalpha, AgKalpha
where:
> a = [a1 a2 a3 a4] is the 'a' coefficient vector
> b = [b1 b2 b3 b4] is the 'b' coefficient vector
> c = scattering factor coefficient
> deltaf_real = [deltaf'(CrKa) deltaf'(FeKa) deltaf'(CuKa) deltaf'(MoKa) deltaf'(AgKa)] is the real component of the anomalous dispersion correction.
> deltaf_imag = [deltaf''(CrKa) deltaf''(FeKa) deltaf''(CuKa) deltaf''(MoKa) deltaf''(AgKa)] is the imaginary component of the anomalous dispersion correction.

The expression for the uncorrected ASF is given by:

$$f0 = \sum_{i=1}^{4} \{ a(i) * \exp[-b(i) * x^2] \} + c$$

And the anomolous dispersion corrected ASF is:

$$f = f0 + deltaf\_real + i * deltaf\_imag$$

where 'i' is the imaginary unit.

- Values for a, b, c, and x can be obtained from the International
  Tables for X-Ray Crystallography, Volume IV, Table 2.2B (pp 99-101)
- Values for deltaf_real and deltaf_imag can be obtained from
  Table 2.3.1 (pp 149-150)

3) ASF_CALC2
Calculate atomic scattering factor (ASF) for ternary material

4) ASF_COEFF
Provides the atomic scattering factor (ASF) coefficients for
various elements, including the anomolous dispersion corrections.
See asf_calc.m for a detailed explanation of ASF calculations.

> a = [a1 a2 a3 a4], b = [b1 b2 b3 b4], and c can be obtained from the
International Tables for X-Ray Crystallography, Volume IV, Table 2.2B (pp 99-101)

> deltaf_real = [deltaf'(CrKa) deltaf'(FeKa) deltaf'(CuKa) deltaf'(MoKa) deltaf'(AgKa)]
and
> deltaf_imag = [deltaf"(CrKa) deltaf"(FeKa) deltaf"(CuKa) deltaf"(MoKa) deltaf"(AgKa)]
can be obtained from Table 2.3.1 (pp 149-150).

5) CAUCHYPDF
Cauchy probability density function (pdf), p= b/(pi*(b^2 + (x-a)^2)).

USAGE: p= cauchypdf(x, a, b)

ARGUMENTS:
x might be of any dimension.
a (default value: 0.0) must be scalars or size(x).
b (b>0, default value: 1.0) must be scalars or size(x).

EXAMPLE:
x= -3:0.01:3;
plot(x, cauchypdf(x));

SEE ALSO: cauchycdf, cauchyfit, cauchyinv, cauchyrnd.

6) CLOSEWINDOWS
Closes all currently open MATLAB Figure windows.

Useful for when your simulations create tons of open plots and it takes forever to close them all. Also useful when you abort a simulation or an error exception occurs, leaving a stuck progress bar. Especially handy to have as a Shortcut on the MATLAB toolbar.

7) CONFIG
Specify here the configuration of a rocking curve simulation for use with xrd_sim.m.

8) CONFIG_CHOICES
List of possible choices in the config file.

9) CONFIG_EXAMPLE
An example config file. Config files specify the configuration of a rocking curve simulation for use with xrd_sim.m.
Note: Items with three asterisks in a comment (***) indicate that the output for this particular item must be in the corresponding list in xrd_sim.m, and you must put the corresponding index for output.

10) ELASTICCONSTANT_CALC
Calculates the elastic constant using Vegard's Law.

11) FWHM
Full-Width at Half-Maximum (FWHM) of the waveform y(x) and its polarity.

The FWHM result in 'width' will be in units of 'x'

12) INSTR_BROADEN
Convolves 'intensity' with a broadening function of the user's choice.

13) LATTICEPARAM_CALC
Calculates lattice parameter for a ternary layer

a is an input vector of the form [a1 a2 ... an] for the calculation of:
a = a1*x^n + a2*x^(n-1) + ... + an

14) MAKEMFILELIST
Reads the comment header of all ".m" files in the directory and writes them in a formatted list Mfiles.txt.

15) MULTISIMD
Run a batch of simulations with various dislocation densities and save the results as .fig files. It's also possible to run the set specified set of of dislocation densities on another set of config files. The number of simulations run will be the product length(D) x length(configFNs).

16) MYFWHM
Calculate full width at half maximum (FWHM) in a subrange of a plot made by xrd_sim.m

usage: myfwhm(x_vec, y_vec, x_lower, x_upper)

17) NONLINSPACE
Like linspace, except gives a cubic vector

18) NUM2CLIP
copies a numerical-array to the clipboard

ARRAYSTRING = NUM2CLIP(ARRAY)

Copies the numerical array ARRAY to the clipboard as a tab-separated string. This format is suitable for direct pasting to Excel and other programs.

The tab-separated result is returned as ARRAYSTRING. This functionality has been included for completeness.

19) PARALLEL
Configure parallel computing xrd_sim job

Overloaded methods:
InputOutputModel/parallel

20) PLOTSCOLOR
Creates an overlaid plot of all the .FIG files in the invoked directory that match the search string.

21) ROCKINGCURVE_PLOT
Plots, in a new figure, a rocking curve versus x_scale which must be in arc seconds, adds axis labels and a title, and returns a handle to the figure.

22) SF_CALC
Calculate Structure Factor
Function calculates the structure factor for the hkl reflection of a zinc blende crystal structure made up of atoms with atomic scattering factors f_A and f_B.

23) NUM2CLIP copies a numerical-array to the clipboard

ARRAYSTRING = NUM2CLIP(ARRAY)

Copies the numerical array ARRAY to the clipboard as a tab-separated string. This format is suitable for direct pasting to Excel and other programs.

The tab-separated result is returned as ARRAYSTRING. This functionality has been included for completeness.

24) TRIPDF
Triangular Probability Distribution Function

25) XRD_SIM
Simulate x-ray diffraction rocking curves using mosaic crystal model

An ongoing program to simulate x-ray diffraction rocking curves for arbitrary heterostructures. Used for research with Dr. John Ayers.

Paul Rago
Department of Electrical & Computer Engineering
University of Connecticut
(c) 2008 - 2014

26) XRD_SIM_PIDDM
Simulate x-ray diffraction rocking curves using phase-invariant model

An ongoing program to simulate x-ray diffraction rocking curves for arbitraty heterostructures. Used for research with Dr. John Ayers.

Paul Rago
Department of Electrical & Computer Engineering
University of Connecticut
(c) 2008 - 2014

Source Code of Functions

1) FWHMdata.m

```
% FWHMdata
% Calculate full width at half maximum (FWHM) in a subrange of a formatted dataset made
by xrd_sim.
% automatically removes header from the XRDdata cell and converts cell to a matrix for
use by myfwhm

function width = FWHMdata(XRDdata, x_lower, x_upper )
width = myfwhm(cell2mat(XRDdata(2:end,1)),cell2mat(XRDdata(2:end,2)),x_lower,x_upper);
```

2) asf_calc.m

```
% ASF_CALC
% Calculate atomic scattering factor (ASF)
% Function calculates the analytical approximation to the x-ray ASF with
% anomalous dispersion correction. Anomalous dispersion correction can be
% included by setting deltaf_real and deltaf_imag to appropriate values
% (use zero to neglect anomalous dispersion).
function [ f ] = asf_calc(coeffs, x, radiationChoice)
%
%   Input Variables:
%   > coeffs = {a b c deltaf_real deltaf_imag}
%   > x = sin(theta) / lambda
%   > radiationChoice = 1,2,3,4,5 for CrKalpha, FeKalpha, CuKalpha, MoKalpha, AgKalpha
%   where:
%   > a = [a1 a2 a3 a4] is the 'a' coefficient vector
%   > b = [b1 b2 b3 b4] is the 'b' coefficient vector
%   > c = scattering factor coefficient
%   > deltaf_real = [deltaf'(CrKa) deltaf'(FeKa) deltaf'(CuKa) deltaf'(MoKa)
deltaf'(AgKa)] is the real component of the anomalous dispersion correction.
%   > deltaf_imag = [deltaf"(CrKa) deltaf"(FeKa) deltaf"(CuKa) deltaf"(MoKa)
deltaf"(AgKa)] is the imaginary component of the anomalous dispersion correction.
%
%   The expression for the uncorrected ASF is given by:
%
%            4
%     f0 = SUM { a(i) * exp[ -b(i) * x^2 ] } + c
%          i=1
%
%   And the anomolous dispersion corrected ASF is:
%
%     f = f0 + deltaf_real + i * deltaf_imag
%
%   where 'i' is the imaginary unit.
%
%   - Values for a, b, c, and x can be obtained from the International
%     Tables for X-Ray Crystallography, Volume IV, Table 2.2B (pp 99-101)
%   - Values for deltaf_real and deltaf_imag can be obtained from
%     Table 2.3.1 (pp 149-150)


%set coefficient variables from "coeffs" cell
a = coeffs{1};
b = coeffs{2};
c = coeffs{3};
deltaf_real = coeffs{4}(radiationChoice);
deltaf_imag = coeffs{5}(radiationChoice);

%perform calculation
output = c + deltaf_real + i * deltaf_imag;

for k = 1:4
    output = output + (a(k) * exp(-b(k) * x^2));
```

```
    end

    f = output;
```

3) asf_calc2.m

```
% ASF_CALC2
% Calculate atomic scattering factor (ASF) for ternary material

function [ f ] = asf_calc2(coeffs1, coeffs2, composition, x, radiationChoice)

f = composition * asf_calc(coeffs1, x, radiationChoice) + (1 - composition) * asf_calc(coeffs2,
x, radiationChoice);
```

4) asf_coefficients.m

```
    % ASF_COEFF
    % Provides the atomic scattering factor (ASF) coefficients for
    % various elements, including the anomolous dispersion corrections.
    % See asf_calc.m for a detailed explanation of ASF calculations.
    %
    %   > a = [a1 a2 a3 a4], b = [b1 b2 b3 b4], and c can be obtained from the
    %   International Tables for X-Ray Crystallography, Volume IV, Table 2.2B (pp 99-101)
    %
    %   > deltaf_real = [deltaf'(CrKa) deltaf'(FeKa) deltaf'(CuKa) deltaf'(MoKa)
    deltaf'(AgKa)]
    %   and
    %   > deltaf_imag = [deltaf"(CrKa) deltaf"(FeKa) deltaf"(CuKa) deltaf"(MoKa)
    deltaf"(AgKa)]
    %   can be obtained from Table 2.3.1 (pp 149-150).
    function [ coeffs ] = asf_coefficients(element)

    switch element
        case 'Ga'
            a = [15.2354 6.7006 4.3591 2.9623];
            b = [3.0669 0.2412 10.7805 61.4135];
            c = 1.7189;
            deltaf_real = [-0.57 -0.841 -1.354 0.163 0.249];
            deltaf_imag = [1.569 1.168 0.777 1.609 1.059];
        case 'As'
            a = [16.6723 6.0701 3.4313 4.2779];
            b = [2.6345 0.2647 12.9479 47.7972];
            c = 2.531;
            deltaf_real = [-0.365 -0.607 -1.011 -0.06 0.196];
            deltaf_imag = [2.022 1.508 1.006 2.007 1.332];
        case 'Si'
            a = [6.2915 3.0353 1.98910 1.541];
            b = [2.4386 32.3337 0.678500 81.6937];
            c = 1.14070;
            deltaf_real = [0.355 0.311 0.244 0.072 0.042];
            deltaf_imag = [0.693 0.509 0.33 0.071 0.043];
        case 'In'
            a = [19.1624 18.5596 4.2948 2.0396];
            b = [0.5476 6.3776 25.8499 92.8029];
            c = 4.9391;
            deltaf_real = [-1.788 -0.626 -0.126 -0.936 -1.493];
            deltaf_imag = [9.627 7.356 5.045 1.31 0.854];
        case 'P'
            a = [6.4345 4.1791 1.78 1.4908];
            b = [1.9067 27.157 0.526 68.1645];
            c = 1.1149;
            deltaf_real = [0.377 0.347 0.283 0.09 0.055];
            deltaf_imag = [0.9 0.664 0.434 0.095 0.058];
        case 'Zn'
            a = [14.0743 7.0318 5.1652 2.41];
```

```
            b = [3.2655 0.2333 10.3163 58.7097];
            c = 1.3041;
            deltaf_real = [-0.684 -0.978 -1.612 0.222 0.26];
            deltaf_imag = [1.373 1.021 0.678 1.431 0.938];
        case 'Se'
            a = [17.0006 5.8196 3.9731 4.3543];
            b = [2.4098 0.2726 15.2372 43.8163];
            c = 2.8409;
            deltaf_real = [-0.273 -0.503 -0.879 -0.178 0.152];
            deltaf_imag = [2.283 1.704 1.139 2.223 1.481];
        case 'Ge'
            a = [16.0816 6.3747 3.7068 3.683];
            b = [2.8509 0.2516 11.4468 54.7625];
            c = 2.1313;
            deltaf_real = [-0.462 -0.717 -1.163 0.081 0.228];
            deltaf_imag = [1.786 1.331 0.886 1.801 1.19];
        case 'Al'
            a = [6.4202 1.9002 1.5936 1.9646];
            b = [3.0387 0.7426 31.5472 85.0886];
            c = 1.1151;
            deltaf_real = [0.318 0.269 0.204 0.056 0.032];
            deltaf_imag = [0.522 0.381 0.246 0.052 0.031];
        case 'S'
            a = [6.9053 5.2034 1.43479 1.5863];
            b = [1.4679 22.2151 0.2536 56.172];
            c = 0.8669;
            deltaf_real = [0.374 0.37 0.319 0.11 0.068];
            deltaf_imag = [1.142 0.847 0.557 0.124 0.076];
        case 'Sb'
            a = [19.6418 19.0455 5.0371 2.6827];
            b = [5.3034 0.4607 27.9074 75.2825];
            c = 4.5909;
            deltaf_real = [-3.194 -1.214 -0.287 -0.816 -1.284];
            deltaf_imag = [11.166 8.557 5.894 1.546 1.010];
        case 'Cd'
            a = [19.2214 17.6444 4.461 1.6029];
            b = [0.5946 6.9089 24.7008 87.4825];
            c = 5.0694;
            deltaf_real = [-1.303 -0.416 -0.079 -1.005 -1.637];
            deltaf_imag = [8.912 6.8 4.653 1.202 0.783];
        case 'Te'
            a = [19.9644 19.0138 6.14487 2.5239];
            b = [4.81742 0.420885 28.5284 70.8403];
            c = 4.352;
            deltaf_real = [-4.267 -1.63 -0.418 -0.772 -1.212];
            deltaf_imag = [11.995 9.203 6.352 1.675 1.096];
        case 'Hg'
            a = [20.6809 19.0417 21.6575 5.9676];
            b = [0.545 8.4484 1.5729 38.3246];
            c = 12.6089;
            deltaf_real = [-4.803 -4.523 -4.99 -3.084 -1.576];
            deltaf_imag = [14.143 10.963 7.686 9.223 6.299];
    end

    coeffs = {a b c deltaf_real deltaf_imag};
```

5) cauchypdf.m

```
% CAUCHYPDF
% Cauchy probability density function (pdf), p= b/(pi*(b^2 + (x-a)^2)).
%
%   USAGE:       p= cauchypdf(x, a, b)
%
%   ARGUMENTS:
%   x might be of any dimension.
%   a (default value: 0.0) must be scalars or size(x).
%   b (b>0, default value: 1.0) must be scalars or size(x).
%
```

```
%   EXAMPLE:
%   x= -3:0.01:3;
%   plot(x, cauchypdf(x));
%
function p= cauchypdf(x, varargin)

    % Default values
    a=  0.0;
    b=  1.0;



    % Check the arguments
    if(nargin >= 2)
        a=  varargin{1};
        if(nargin == 3)
            b=          varargin{2};
            b(b <= 0)=  NaN;    % Make NaN of out of range values.
        end
    end
    if((nargin < 1) || (nargin > 3))
        error('At least one argument, at most three!');
    end



    % Calculate
    p=  b./(pi*(b.^2 + (x-a).^2));
end
```

6) closewindows.m

```
% CLOSEWINDOWS
% Closes all currently open MATLAB Figure windows.
%
%   Useful for when your simulations create tons of open plots and it takes forever to
close them
%   all. Also useful when you abort a simulation or an error exception occurs, leaving a
stuck
%   progress bar. Especially handy to have as a Shortcut on the MATLAB toolbar.

choice = questdlg('Are you sure you want to close all windows?', 'Close All Windows?',
'Yes','No','Yes');
if strcmp(choice,'Yes')
    set(0,'ShowHiddenHandles','on');
    delete(get(0,'Children'));
end
```

7) config.m

```
% CONFIG
% Specify here the configuration of a rocking curve simulation for use with xrd_sim.m.

function [ output ] = config( param )

switch param

%Structure Description
    case 'substrate' % *** (1)GaAs, (2)Si, (3)InP, (4)InSb, (5)CdTe, (6)GaSb
        output = 1;
%
    case 'layer_1_type' % (1) uniform, (2) linearly graded, (3) superlattice, (4) (no
more layers)
        output = 1;
```

```
%
    case 'layer_1_material'  %UNIFORM:  (1) 'ZnSe',  (2) 'ZnS',  (3)'ZnS(x)Se(1-x)',  (4)
'In(x)Ga(1-x)As',  (5) 'Si(1-x)Ge(x)',  (6) 'Hg(x)Cd(1-x)Te',  (7) 'In(x)Ga(1-x)Sb',  (8)
'Al(x)Ga(1-x)As'
        output = 3;
    case 'layer_1_composition'
        output = 0.2;
    case 'layer_1_thickness'
        output = 1; %µm
    case 'layer_1_tilt'
        output = 0; % arc seconds


    case 'TDD_mode' %(1) symmetric--must also specify "layer_n_TDD"; (2) asymmetric--must
also specify "layer_n_TDD_A", "layer_n_TDD_B" and "azimuth"; (3) none
        output = 1;
    case 'layer_1_TDD'
        output = 0.1;   %in 10^8 cm^-2 (output=2 <-> TDD=2*10^8/cm^2)
%     case 'layer_1_TDD_A'
%         output = 0.5;              %in 10^8 cm^-2 (output=2 <-> TDD=2*10^8/cm^2)
%     case 'layer_1_TDD_B'
%         output = 0.1;                %in 10^8 cm^-2 (output=2 <-> TDD=2*10^8/cm^2)
    case 'azimuth'
        output = 0;                %in degrees



%     case 'layer_1_material'  %LG:  (1) 'ZnS(x)Se(1-x)', (2) 'In(x)Ga(1-x)As', (3)
'Si(1-x)Ge(x)', (4) 'Hg(x)Cd(1-x)Te', (5) 'In(x)Ga(1-x)Sb', (6) 'Al(x)Ga(1-x)As'
%         output = 1;
%     case 'layer_1_numberOfLamina'
%         output = 21;
%
%     case 'layer_1_startingComposition'
%         output = 0.17;
%     case 'layer_1_endingComposition'
%         output = 0.20;
%
%
%     case 'TDDinitial' %for linear dislocation density profile
%         output = 0;
%     case 'TDDfinal' %for linear dislocation density profile
%         output = 0;




%commonly changed things
    case 'dislocation_distribution_type' %(1) Gaussian, (2) Lorentzian, (3) Triangular,
(4) Rectangular
        output = 1;
    case 'theta_Start'          %in arcseconds
        output = -1200;
    case 'theta_End'            %in arcseconds
        output = 2200;
    case 'reflection'           %put any '(hkl)' you like here
        output = '(004)';
    case 'strain' % (1) Completely Relaxed (none), (2) Pseudomorphic / Coherently
Strained, (3) Custom: Constant Strain (4) Custom: Constant % residual strain
        output = 2;
    case 'filename'
        output = inf; % no XLS file, and no question for XLS file




%seldom changed things
    case 'simulation'
        output = 1; % 'Dynamical (Takagi-Taupin), Kinematical (Bartels-Hornstra-
Lobeek/Speriosu-Vreeland), Both
    case 'fwhm'
```

```
        output = 1; % (1) yes, (2) no
    case 'display_layersummary' % (1) yes, (2) no
        output = 1;
    case 'display_latticeprofile' %  (1)relaxed lattice constant, (2) in-plane, (3) out-
of-plane, (4) in- and out-of-plane in same plot, (5) in- and out-of-plane in different
plots, (6) none
        output = 6;
    case 'display_strainprofile' %  (1) in-plane strain, (2) out-of-plane, (3) in- and
out-of-plane in same plot, (4) in- and out-of-plane in different plots, (5) none
        output = 5;
    case 'radiation' % *** (1) Cr kalpha1, (2) Fe kalpha1, (3) Cu kalpha1, (4) Mo
kalpha1, (5) Ag kalpha1
        output = 3;
    case 'broadening_choice' % *** (1) Gaussian, (2) Rectangular, (3) (none)
        output = 3;
    case 'broadening_width' %%%
        output = 12; %arcseconds
    case 'view_broadening_fn' %(1) yes, (2) no
        output = 2;
    case 'numPtsPerArcSec' %enter any postive integer here, but 5 seems to work well.
        output = 2;
    case 'substrateOrientation' %put any '(uvw)' you like here
        output = '(001)';



    otherwise
        output = -inf;
end
```

8) config_choices.m

```
% CONFIG_CHOICES
% List of possible choices in the config file.

%% KEY
% (1) Dashes (-) indicate disjunction. You must choose only one in the group.
% (2) Indenting indicates that only the indented option is available if its parent is
chosen.



%% LIST OF CONFIG FILE OPTIONS

'substrate'; %  (1)GaAs, (2)Si, (3)InP, (4)InSb, (5)CdTe, (6)GaSb

'layer_N_type'; % (1) uniform, (2) linearly graded, (3) superlattice, (4) (no more
layers)
    - 'layer_N_material';  %UNIFORM:  (1) 'ZnSe',  (2) 'ZnS',  (3)'ZnS(x)Se(1-x)',  (4)
'In(x)Ga(1-x)As',  (5) 'Si(1-x)Ge(x)',  (6) 'Hg(x)Cd(1-x)Te',  (7) 'In(x)Ga(1-x)Sb',  (8)
'Al(x)Ga(1-x)As'
        'layer_N_composition'; %0 to 1, only for ternery layers
        'layer_N_thickness'; % in microns (µm)
    - 'layer_N_material';  %LG:  (1) 'ZnS(x)Se(1-x)', (2) 'In(x)Ga(1-x)As', (3) 'Si(1-
x)Ge(x)', (4) 'Hg(x)Cd(1-x)Te', (5) 'In(x)Ga(1-x)Sb', (6) 'Al(x)Ga(1-x)As'
        'layer_N_composition'; %0 to 1, only for ternery layers
        'layer_N_startingComposition' ; 'layer_N_endingComposition'; %0 to 1, only for
linearly graded ternary layers
        'layer_N_numberOfLamina';
        'layer_N_thickness'; % in microns (µm)
    - 'layer_N_material_A'; 'layer_N_material_B';  %SL:  (1) 'ZnS(x)Se(1-x)', (2)
'In(x)Ga(1-x)As', (3) 'Si(1-x)Ge(x)', (4) 'Hg(x)Cd(1-x)Te', (5) 'In(x)Ga(1-x)Sb', (6)
'Al(x)Ga(1-x)As'
        'layer_N_composition_A'; 'layer_N_composition_B'; %0 to 1 the composition for
material A and B
        'layer_N_thickness_A'; 'layer_N_thickness_B'; %in nanometers
        'layer_N_numberOfPeriods';
```

```
%dislcoations
- 'TDD_mode'; %(1) symmetric--must also specify "layer_n_TDD"; (2) asymmetric--must also
specify "layer_n_TDD_A", "layer_n_TDD_B" and "azimuth"; (3) none
- 'layer_N_TDD'; %for uniform D in layer N; in 10^8 cm^-2 (output=2 <-> TDD=2*10^8/cm^2)
- 'layer_N_TDD_A'; %in 10^8 cm^-2 (output=2 <-> TDD=2*10^8/cm^2)
- 'layer_N_TDD_B'; %in 10^8 cm^-2 (output=2 <-> TDD=2*10^8/cm^2)
- 'TDD_uniform';   %in 10^8 cm^-2 (output=2 <-> TDD=2*10^8/cm^2)
- 'TDDinitial' ; 'TDDfinal'; %for linear dislocation density profile; in 10^8 cm^-2
(output=2 <-> TDD=2*10^8/cm^2)
'dislocation_distribution_type'; %(1) Gaussian, (2) Lorentzian, (3) Triangular, (4)
Rectangular


- 'layer_N_tilt'; %arc seconds
- 'azimuth';        %in degrees


'theta_Start';          %angular range for theta-thetaBsub in arcseconds
'theta_End';            %angular range for theta-thetaBsub in arcseconds
'reflection';           %put any '(hkl)' you like here, e.g. '(004)'
'strain'; % (1) Completely Relaxed (none), (2) Pseudomorphic / Coherently Strained, (3)
Custom: Constant Strain (4) Custom: Constant % residual strain
    'layer_N_inPlaneStrain'; %for custom strain only
'filename'; % outfile.XLS, or inf for no XLS file and no question
'PutDataOnClipboard'; %puts XRD data on clipboard after running simulation
'simulation'; % 'Dynamical (Takagi-Taupin), Kinematical (Bartels-Hornstra-
Lobeek/Speriosu-Vreeland), Both
'fwhm'; % (1) yes, (2) no
'display_layersummary'; % (1) yes, (2) no
'display_latticeprofile'; %  (1)relaxed lattice constant, (2) in-plane, (3) out-of-plane,
(4) in- and out-of-plane in same plot, (5) in- and out-of-plane in different plots, (6)
none
'display_strainprofile'; %  (1) in-plane strain, (2) out-of-plane, (3) in- and out-of-
plane in same plot, (4) in- and out-of-plane in different plots, (5) none
'radiation'; % *** (1) Cr kalpha1, (2) Fe kalpha1, (3) Cu kalpha1, (4) Mo kalpha1, (5) Ag
kalpha1
'broadening_choice'; % *** (1) Gaussian, (2) Rectangular, (3) (none)
'broadening_width'; %width of broadening function if broadening_choice = 1 or 2 in
arcseconds
'view_broadening_fn'; %(1) yes, (2) no
'numPtsPerArcSec'; %enter any postive integer here
'substrateOrientation'; %put any '(uvw)' you like here, e.g. '(001)'
```

9) config_example.m

```
% CONFIG_EXAMPLE
% An example config file. Config files specify the configuration of a rocking curve
simulation for
% use with xrd_sim.m.
%   Note: Items with three asterisks in a comment ( *** ) indicate that the
%   output for this particular item must be in the corresponding list
%   in xrd_sim.m, and you must put the corresponding index for output.

function [ output ] = config_example( param )



switch param


%Structure Description
    case 'substrate' % *** (1)GaAs, (2)Si, (3)InP, (4)InSb, (5)CdTe, (6)GaSb
        output = 6;


    case 'layer_1_type' % (1) uniform, (2) linearly graded, (3) superlattice, (4) (no
more layers)
```

```
        output = 1; % 1 = uniform
    case 'layer_1_material' % (1)ZnSe, (2)ZnS, (3)ZnS(x)Se(1-x), (4)In(x)Ga(1-x)As,
(5)Si(1-x)Ge(x), (6)Hg(x)Cd(1-x)Te, (7)In(x)Ga(1-x)Sb
        output = 7; %7 = (7)In(x)Ga(1-x)Sb
    case 'layer_1_composition' % composition x = any value in the range 0 to 1).
        output = 1;
    case 'layer_1_thickness' % in microns (µm)
        output = 2;

%commonly changed things
    case 'reflection'           %put any '(hkl)' you like here
        output = '(004)';
    case 'theta_Start'          %in arcseconds
        output = -10000;
    case 'theta_End'            %in arcseconds
        output = 1000;
    case 'strain' % (1) Completely Relaxed (none), (2) Pseudomorphic / Coherently
Strained, (3) Custom Strain
        output = 1;
    case 'filename'
        output = inf; % inf means no XLS file, and no question for XLS file
    case 'fwhm'
        output = 1; % (1) yes, (2) no



%seldom changed things
    case 'display_layersummary' % (1) yes, (2) no
        output = 2;
    case 'display_latticeprofile' % (1) yes, (2) no
        output = 2;
    case 'broadening_choice' % *** (1) Gaussian, (2) Rectangular, (3) (none)
        output = 3;
    case 'radiation' % *** (1) Cr kalpha1, (2) Fe kalpha1, (3) Cu kalpha1, (4) Mo
kalpha1, (5) Ag kalpha1
        output = 3;
    case 'numPtsPerArcSec' %enter any postive integer here, but 5 seems to work well.
        output = 5;
    case 'substrateOrientation' %put any '(uvw)' you like here
        output = '(001)';



    otherwise
        output = -inf;
end
```

10) elasticconstant_calc.m

```
% ELASTICCONSTANT_CALC
% Calculates the elastic constant using Vegard's Law.

function [ elasticconstant ] = elasticconstant_calc(C_constants, composition )
elasticconstant = composition * C_constants(1) + (1 - composition) * C_constants(2);
```

11) fwhm.m

```
% FWHM
% Full-Width at Half-Maximum (FWHM) of the waveform y(x) and its polarity.
%
%   The FWHM result in 'width' will be in units of 'x'
```

```
function width = fwhm(x,y)

y = y / max(y);
N = length(y);
lev50 = 0.5;
if y(1) < lev50                  % find index of center (max or min) of pulse
    [garbage,centerindex]=max(y);
    Pol = +1;
    %disp('Pulse Polarity = Positive')
else
    [garbage,centerindex]=min(y);
    Pol = -1;
    %disp('Pulse Polarity = Negative')
end
i = 2;
while sign(y(i)-lev50) == sign(y(i-1)-lev50)
    i = i+1;
end                                  %first crossing is between v(i-1) & v(i)
interp = (lev50-y(i-1)) / (y(i)-y(i-1));
tlead = x(i-1) + interp*(x(i)-x(i-1));
i = centerindex+1;                   %start search for next crossing at center
while ((sign(y(i)-lev50) == sign(y(i-1)-lev50)) & (i <= N-1))
    i = i+1;
end
if i ~= N
    Ptype = 1;
    %disp('Pulse is Impulse or Rectangular with 2 edges')
    interp = (lev50-y(i-1)) / (y(i)-y(i-1));
    ttrail = x(i-1) + interp*(x(i)-x(i-1));
    width = ttrail - tlead;
else
    Ptype = 2;
    %disp('Step-Like Pulse, no second edge')
    ttrail = NaN;
    width = NaN;
end
```

12) instr_broaden.m

```
% INSTR_BROADEN
% Convolves 'intensity' with a broadening function of the user's choice.

function [ intensityBroadened ] = instr_broaden( intensity, numPtsPerArcSec, config )
broadening_choice = config('broadening_choice');
if broadening_choice == -inf
    broadening_choice = menu('With what type of instrumental broadening function would
you like to convolve the scattering amplitude?', 'Gaussian', 'Rectangular', '(none)');
end
if broadening_choice == 1 || broadening_choice == 2 %user wants broadening
    broadening_width = numPtsPerArcSec * config('broadening_width');
    if broadening_width == -inf
        broadening_width = numPtsPerArcSec * str2double(inputdlg('Enter the full width at
half maximum (FWHM) of the broadening function (in arc seconds):'));
    end

    switch broadening_choice
        case 1 %guassian
            sigma = broadening_width / (2 * sqrt(2 * log(2)));
            numpts = 10*sigma;
            if mod(numpts,2) ~= 0
                numpts = floor(numpts)+1;
            end
            x_values = linspace(-5*sigma,5*sigma,numpts);
            broadening_fn = 1/(sigma*sqrt(2*pi)) .* exp(-(x_values).^2 / (2*sigma^2) );
        case 2 %rectangle
            broadening_fn = ones(1, broadening_width); %construct the rectangle
```

```
            broadening_fn = padarray(broadening_fn, [0 broadening_width]); %put some
zeros on either side (for better viewing of fn.)
    end

    %normalize area of broadening function to be 1
    broadening_fn = broadening_fn / sum(broadening_fn);

    view_fn = config('view_broadening_fn');
    if view_fn == -inf
        view_fn = menu('Would you like to view a plot of the broadening function?',
'yes','no');
    end
    if view_fn == 1
        xs = length(broadening_fn)/numPtsPerArcSec;
        figure;
        plot(linspace(-xs/2,xs/2,length(broadening_fn)), broadening_fn);
        xlabel('(arc seconds)');
        title('Instrumental Broadening Function');
    end

    output = conv(intensity, broadening_fn);
    trimsides = length(broadening_fn) / 2;
    output = output(trimsides:length(output)-trimsides); %trim to correct length
    intensityBroadened = output;
else
    intensityBroadened = -1;
end
```

13) latticeparam_calc.m

```
% LATTICEPARAM_CALC
% Calculates lattice parameter for a ternary layer
%
%   a is an input vector of the form [a1 a2 ... an] for the calculation of:
%   a = a1*x^n + a2*x^(n-1) + ... + an

function [ lattice_parameter ] = latticeparam_calc( a, x )

out = 0;
n = numel(a)-1;

for k = 0:n
    out = out + a(k+1) * x^(n-k);
end

lattice_parameter = out;
```

14) makeMfileList.m

```
% MAKEMFILELIST
% Reads the comment header of all ".m" files in the directory and writes them in a
formatted list
% Mfiles.txt.

%% get the list of m files in the directory
mFiles = dir('*.m');

%% step thru and grab the helptext for each m file; write to outfile
fid = fopen('Mfiles.txt', 'w');  %w for write permission
```

```
fprintf(fid, '%s\n\n', 'XRD_SIM M-FILE LISTING');
for idx = 1:length(mFiles)
    fprintf(fid, '%2.0f%s%s\n', idx, ') ', mFiles(idx).name);
end

fprintf(fid, '\n\n\nFILE DESCRIPTIONS\n\n');

for idx = 1:length(mFiles)
    helptext = help(mFiles(idx).name);
    fprintf(fid, '%2.0f%s%s\n\n', idx, ') ', helptext);
end

fclose(fid);
```

15) multisimD.m

```
% MULTISIMD
% Run a batch of simulations with various dislocation densities and save the results as
.fig files.
% It's also possible to run the set specified set of of dislocation densities on another
set of
% config files. The number of simulations run will be the product length(D) x
length(configFNs).

clear;

D=[0 .01 .05 .1 .5];
%D=fliplr(D);
configFNs = {'LinearlyGradedInGaAs'};

for kdx = 1:length(configFNs)

    %start making output filename
    cwd = pwd;
    posbackslashes = strfind(cwd,'\');
    outfile = '';%cwd(posbackslashes(end)+1:end);
    outdir='';%['tilt=' num2str(tilt) 'arcsec; D=' num2str(D) 'e8cm-2\'];
    %mkdir(outdir);

    %step thru the config files one at a time and run the sim on em
    for idx = 1:length(D);
        clear XRDdata;

        %run the simulation
        fprintf(['\nNow doing D=' num2str(D(idx)) ' x10^8 cm-2...\n']);
        [XRDdata, RockingCurve, RockingCurveBroadened] = xrd_sim(configFNs{kdx}',
'TDDuniform', D(idx));

    %     %determine and log FWHM
    %     x_vec=cell2mat(XRDdata(2:end,1));
    %     y_vec=cell2mat(XRDdata(2:end,3));
    %     beta(idx) = myfwhm(x_vec,y_vec,-1200,-200);
    %

        %add D to title
        rc = [RockingCurve RockingCurveBroadened];
        for k=1:2
            figure(rc(k));
            h=get(gca,'Title');
            origtitle = get(h,'String');
            set(h,'String',[origtitle '; D = ' num2str(D(idx)) ' \times 1\times10^8 cm^{-
2}']);
```

```
        end


        %save figures
        outfile = [configFNs{kdx} '_' num2str(idx) '_D=' num2str(D(idx)) 'x1e8 cm^{-2}'];
        saveas(RockingCurve,strcat(outdir,outfile,'_unbroadened.fig'), 'fig');
        saveas(RockingCurveBroadened,strcat(outdir,outfile,'_broadened.fig'), 'fig');
        close(RockingCurve);
        %close(RockingCurveBroadened);

    %     %write the data to the outfile
    %     success = xlswrite(strcat(outfile,'_',configfilelist{idx},'.xls'),XRDdata);
    %     if success
    %        disp(['The worksheet ' configfilelist{idx} ' was written to '
strcat(outfile,'_',configfilelist{idx},'.xls')]);
    %     else
    %        warning(['There was a problem writing the worksheet ' configfilelist{idx} '
to ' strcat(outfile,'_',configfilelist{idx},'.xls')]);
    %     end
    %
    end

    % b2 = beta.^2; %square the fwhms

    % h_betapsi = figure;
    % plot(psi,b2);
    % xlabel('\psi (degrees)');
    % ylabel('\beta^2 (arcsec)^2');

    % saveas(h_betapsi,strcat(outdir,'beta(psi).fig'));

    %[a,b,h_betapsi] = createFit(az,b2);
    %saveas(h_betapsi,strcat(outdir,'beta(psi).fig'));
end
```

16) myfwhm.m

```
% MYFWHM
% Calculate full width at half maximum (FWHM) in a subrange of a plot made by xrd_sim.m
%
%   usage: myfwhm(x_vec, y_vec, x_lower, x_upper)

function [ width ] = myfwhm (x_vec, y_vec, x_lower, x_upper )

for k = 1:length(x_vec)

    if x_vec(k) >= x_lower %found start point
        k_lower = k;
        break;
    end
end
for k = k_lower:length(x_vec)
    if x_vec(k) >= x_upper %found end point
        k_upper = k;
        break;
    end
end

width = fwhm(x_vec(k_lower:k_upper), y_vec(k_lower:k_upper));
```

17) nonlinspace.m

```
% NONLINSPACE
% Like linspace, except gives a cubic vector

function [ output ] = nonlinspace( limit, numPts )

output = zeros(1,numPts);

for idx = 1:((numPts-1)/2)

    output((numPts-1)/2 - idx+1) = - limit * (2*idx/(numPts-1))^3;
    output((numPts-1)/2 + idx+1) = - output((numPts-1)/2 - idx+1);

end
```

18) num2clip.m

```
% NUM2CLIP
% copies a numerical-array to the clipboard
%
%   ARRAYSTRING = NUM2CLIP(ARRAY)
%
%   Copies the numerical array ARRAY to the clipboard as a tab-separated
%   string.  This format is suitable for direct pasting to Excel and other
%   programs.
%
%   The tab-separated result is returned as ARRAYSTRING.  This
%   functionality has been included for completeness.
%


function arraystring = num2clip(array)
%convert the numerical array to a string array
%note that num2str pads the output array with space characters to account
%for differing numbers of digits in each index entry
arraystring = num2str(array);
arraystring(:,end+1) = char(10); %add a carrige return to the end of each row
%reshape the array to a single line
%note that the reshape function reshape is column based so to reshape by
%rows one must use the inverse of the matrix
arraystring = reshape(arraystring',1,prod(size(arraystring))); %reshape the array to a
single line

arraystringshift = [' ',arraystring]; %create a copy of arraystring shifted right by one
space character
arraystring = [arraystring,' ']; %add a space to the end of arraystring to make it the
same length as arraystringshift

%now remove the additional space charaters - keeping a single space
%charater after each 'numerical' entry
arraystring = arraystring((double(arraystring)~=32 | double(arraystringshift)~=32) &
~(double(arraystringshift==10) & double(arraystring)==32) );

arraystring(double(arraystring)==32) = char(9); %convert the space characters to tab
characters

clipboard('copy',arraystring); %copy the result to the clipboard ready for pasting
```

19) parallel.m

```
% PARALLEL
% Configure parallel computing xrd_sim job

%% Configure the server for job manager and create job
sched1 = findResource('scheduler', 'type', 'hpcserver');
set(sched1, 'SchedulerHostname', 'ecs-hpc1-head1.ad.engr.uconn.edu');
set(sched1, 'ClusterVersion', 'HPCServer2008');
set(sched1, 'UseSOAJobSubmission', false);
set(sched1, 'ClusterMatlabRoot', 'C:\Program Files\MATLAB\R2010b\');
%set(sched1, 'DataLocation',
'\\nas.engr.uconn.edu\home\public_html\XRDsim_current\output');
set(sched1, 'DataLocation', 'J:\public_html\XRDsim_current');
%set(sched1, 'DataLocation', '\\ecs-hpc1-head1.ad.engr.uconn.edu\Work\Output');
get(sched1);
j1 = createParallelJob(sched1);
createTask(j1, 'xrd_sim_parallel.m', 1, {});
set(j1, 'MaximumNumberOfWorkers', 8);
set(j1, 'MinimumNumberOfWorkers', 8);
submit(j1);
waitForState(j1, 'finished');
results = getAllOutputArguments(j1);
```

20) plotscolor.m

```
% PLOTSCOLOR
% Creates an overlaid plot of all the .FIG files in the invoked directory that match the
search
% string.

function [h_plot, FWHMs] = plotscolor(searchstr,paramstr,DoBroadened,colors)

% searchstr = 'LinearlyGraded'; %the string from which all plots will be gathered
% paramstr = 'D='; %the parameter to get from the plots' filenames (begins after this)
% DoBroadened = 'yes';

if ~exist('colors','var')
    colors = 'rgbmkcy';
end;
if isequal(DoBroadened, 'yes')
    broadstr='_broadened';
else
    broadstr='_unbroadened';
end
figFNs = dir(strcat('*',searchstr,'*',broadstr,'.fig'));

n=0;
h_multiplot = figure;
xmin=0;
xmax=0;
for file_idx = 1:length(figFNs)
    if figFNs(file_idx).name(1) ~= 'z'

        %increment the counter
        n=n+1;

        %open the fig
        open(figFNs(file_idx).name)

        %Get a handle to the current figure:
        h = gcf;
```

```
        %Find the line objects in the figure. This will return all the lines in the
current figure.
        line = findall(h, 'Type', 'Line');

        %Extract X and Y data values from the appropriate line
        for line_idx = 1:length(line)
            if numel(get(line(line_idx),'xdata')) > 100
                break;
            end
        end
        x_scale   = get(line(line_idx), 'xdata');
        intensity = get(line(line_idx), 'ydata');

        xmin = min(xmin,x_scale(1));
        xmax = max(xmax,x_scale(end));

        %close the figure
        close(h);

        %keep track of the plots titles, for the legend later
        startidx = strfind(figFNs(file_idx).name,paramstr) + length(paramstr);
        stopidx = strfind(figFNs(file_idx).name(startidx:end),'_');
        titles{n} =
strrep(strrep(strrep(figFNs(file_idx).name(startidx:startidx+stopidx(1)-2),'--
','\rightarrow'),'_broadened',''),'_unbroadened','');

        %measure and record FWHMs
        FWHMs(n) = myfwhm(x_scale, intensity, x_scale(1), x_scale(end));

        %add to the overlayed plot
        semilogy(x_scale,intensity,'Color',colors(n),'LineWidth',2);
        if n == 1; hold on; end;

    end
end %for file_idx...

legend(titles);
%add legend and labels
h_legend = legend(titles);
h_axes = get(h_multiplot,'CurrentAxes');
set(h_axes,'LineWidth',2)
set(h_legend, 'EdgeColor', [1 1 1]);
set(h_legend, 'FontWeight','Bold')
set(h_legend, 'FontSize',8)
xlabel('\theta - \theta_{B_S}  (arc sec)','FontSize',12);
ylabel('intensity  (a.u.)','FontSize',12);
ylim([5e-8 1e0]);
xlim([xmin xmax]);

%%allow user config of legend
%pause;
%legend('boxoff');

%fwhmstr = [{'D'} {'FWHM(sec)'}];
%fwhmstr = [fwhmstr; titles' num2cell(FWHMs)'];

%disp(fwhmstr);

%title('MCDDM (new) model');
title(searchstr);
%W_MCDDM = FWHMs;
h_plot = gcf;

pause;
```

```
%save figure
outdir='';
outfile = [searchstr ' varying ' paramstr];
saveas(h_multiplot,strcat(outdir,outfile,'.fig'), 'fig');
```

21) rockingcurve_plot.m

```
% ROCKINGCURVE_PLOT
% Plots, in a new figure, a rocking curve versus x_scale which must be in arc seconds,
% adds axis labels and a title, and returns a handle to the figure.

function [ fighandle ] = rockingcurve_plot( x_scale, intensity_dyn, title_string)

fighandle = figure; %open a new figure window

maxval = 1; % max(max(intensity_dyn),max(intensity_kin)); %determine maximum value of
both arrays, use this as the normalizing factor

semilogy(x_scale, (intensity_dyn/maxval)) %,'bx-');
%title(strcat(title_string, {' (dynamical)'}));
title(title_string);

xlabel('\theta - \theta_B (arc seconds)');
ylabel('diffracted x-ray intensity (a.u.)');
```

22) sf_calc.m

```
% SF_CALC
% Calculate Structure Factor
%   Function calculates the structure factor for the hkl reflection of a
%   zinc blende crystal structure made up of atoms with atomic scattering
%   factors f_A and f_B.

function [ F_hkl ] = sf_calc( f_A, f_B, h, k, l )

F_hkl = 4 * (f_A + f_B*exp((i*pi/2)*(h+k+l)));
```

23) str2clip.m

```
%NUM2CLIP copies a numerical-array to the clipboard
%
%   ARRAYSTRING = NUM2CLIP(ARRAY)
%
%   Copies the numerical array ARRAY to the clipboard as a tab-separated
%   string.  This format is suitable for direct pasting to Excel and other
%   programs.
%
%   The tab-separated result is returned as ARRAYSTRING.  This
%   functionality has been included for completeness.
%convert the numerical array to a string array
%note that num2str pads the output array with space characters to account
%for differing numbers of digits in each index entry
function arraystring = num2clip(array)
for idx = 1:numel(array)
    array{idx} = num2str(array{idx});
end
arraystring = num2str(array);
arraystring(:,end+1) = char(10); %add a carrige return to the end of each row
```

```
%reshape the array to a single line
%note that the reshape function reshape is column based so to reshape by
%rows one must use the inverse of the matrix
arraystring = reshape(arraystring',1,prod(size(arraystring))); %reshape the array to a
single line


arraystringshift = [' ',arraystring]; %create a copy of arraystring shifted right by one
space character
arraystring = [arraystring,' ']; %add a space to the end of arraystring to make it the
same length as arraystringshift


%now remove the additional space charaters - keeping a single space
%charater after each 'numerical' entry
arraystring = arraystring((double(arraystring)~=32 | double(arraystringshift)~=32) &
~(double(arraystringshift==10) & double(arraystring)==32) );


arraystring(double(arraystring)==32) = char(9); %convert the space characters to tab
characters


clipboard('copy',arraystring); %copy the result to the clipboard ready for pasting
```

24) tripdf.m

```
% TRIPDF
% Triangular Probability Distribution Function
function [ Y ] = tripdf( N )



if mod(N,2) %odd N
    Y1 = linspace(0,1,floor(N/2)+1);
    Y2 = fliplr(Y1(1:(numel(Y1)-1)));
else        %even N
    Y1 = linspace(0,1,N/2);
    Y2 = fliplr(Y1);
end


Y = [Y1 Y2];


end
```

25) xrd_sim.m

```
%% XRD_SIM
% Simulate x-ray diffraction rocking curves using mosaic crystal model
%
%   An ongoing program to simulate x-ray diffraction rocking curves for
%   arbitrary heterostructures. Used for research with Dr. John Ayers.
%
%   Paul Rago
%   Department of Electrical & Computer Engineering
%   University of Connecticut
%   (c) 2008 - 2014
%


%% (0.1) Changelog
%
%       2014-09-11  - Mod Nmin = 15; Dcrossover = 1e9. (See experiment 9/8/14
demonstrating Na,Nb higher than necessary.)
%       2013-10-24  - Modified mosaic crystal model to add all X_n(i,j)s *after*
calculating the scattering amplitude to the top layer
```

```
%       2013-10-21  - Added InAlAs
%       2013-10-01  - Modified to employ the new "mosaic crystal model" instead of the
"phase invariant model"
%       2012-06-22  - Fixed bug in eta: sign of delta_phi_tet and delta_phi_tilt wrong;
delta_phi_tet was not present in D>0 eta.
%       2012-06-18  - Added asymmetric dislocation densities, azimuthal variation, and
epilayer tilt
%                   - Added vargin, property/value pair feature. So far only azimuth
implemented.
%       2011-05-05  - Fixed diagnostic alpha and beta limit lines
%       2011-04-27  - Enhancements of when to display ETC (if LG layer, don't show,
unless TDDinitial=TDDfinal)
%       2011-04-19  - Changed clear to only clear select variables
%                   - added multisim.m, which simulates numerous config files
programatically
%                   - changed xrd_sim to be a function instead of a script, with argin of
config file name
%       2011-04-18  - Set a minimum of 21 for NBin_alpha, NBin_beta.
%                   - Updated makeXvsLayerPlot.m to ask for rocking angle at which to
show plots.
%                   - Added feature to copy XRD data to clipboard. In config.m it's
called 'PutDataOnClipboard'.
%       2011-04-11  - Added makeEtaPlots.m
%                   - Added makeXvsLayerPlot.m
%                   - Modified rockingcurve_plot.m to plot logarithmically.
%                   - Made fixed numPtsPerArcSec/numBinsPerArcSec.
%       2011-03-30  - Finished adding Lorentzian distribution for P_alpha and P_epsilon;
added linear profile for TDD; added fancy Nbin
%                     calculation (instead of fixed); added estimated time till
completion (ETC); cleaned up command window output.
%       2011-03-23  - Added rectangular and triangular distributions for P_alpha and
P_epsilon
%                   - Added 'dislocation_distribution_type' choice to the config file.
%                   - Started implementing Lorentzian distribution.
%       2011-03-16  - Fixed bug in instr_broaden.m where the broadening function's width
was off by a factor of 1/numPtsPerArcsec
%       2011-03-02  - Modified the TDD calculation to use reciprocol of sum of
reciprocols.
%       2011-02-25  - Finished adding the threading dislocation density (TDD) calculation
using Gaussians for alpha and beta.
%       2011-02-24  - Finished adding Al(x)Ga(1-x)As to uniform, LG, and SL layers
(elastic constants)
%       2011-02-23  - Began adding threading dislocation density stuff.
%       2011-02-18  - Added Al(x)Ga(1-x)As to uniform, LG, and SL layers
%                   - Added reflection (P_H) calculation.
%       2011-02-10  - Fixed an error with T(n), the polarization factor C(n) is now a
factor in T(n).
%                   - Removed normalization in the rocking curve plot
%       2011-02-09  - Added kinematical calculation.
%                     ==> STILL NEED TO CLEAN UP THE XLS OUTFILE PART -- currently non-
operational
%       2010-04-28  - Added "percent residual strain" custom strain option
%                     for linearly graded layers
%       2010-03-21  - Changed plotting routine to produce intensity
%                     normalized to the maximum value in the produced
%                     plot.
%       2010-03-02  - Added a configuration file, config.m, which allows a
%                     user to pre-specify the answer to nearly every
%                     question about how the simulation should be run.
%       2010-02-28  - Added AgKalpha1 x-ray source wavelength
%       2010-02-17  - Enhanced the "write to XLS feature". Added title and layer
%                     summary, put XRD data in separate worksheet, and output
%                     instrumentally broadened XRD data too.
%                   - Added Hg(x)Cd(1-x)Te and In(x)Ga(1-x)Sb to superlattice layers.
%                   - Added Cr,Fe,Mo,Ag radiations back in, now including all the real
%                     and imaginary anomalous dispersion corrections for
%                     ASFs. Still need AgKa wavelength!
%       2010-02-15  - Finished adding Hg(x)Cd(1-x)Te and In(x)Ga(1-x)Sb to uniform
%                     and LG layers (with elastic constants).
%       2010-02-13  - Added InSb, CdTe, GaSb substrates.
%                   - Added several reflections and streamlined the
```

```
%                     process of adding reflections
%                   - Removed all source radiation other than Cu kalpha1,
%                     because noticed that in Int'l. Table for X-Ray
%                     Crystallography, I had only used dispersion
%                     corrections for Cu kalpha.
%                   - Added "write to XLS" feature
%
%  ............ /Changelog
.................................................................

%% (0.2) declarations, argument handling
function [XRDdata, RockingCurve, RockingCurveBroadened] = xrd_sim(configfilename,
Property, Value)
% NOTES RE: INPUT VARIABLES
% (1) It is not necessary to specify any of them. If you don't, config.m will be assumed.
% (2) Property is a string that overrides that specified (or not) in the config file.
% (3) Value is the value corresponding to Property, and together these override that
specified (or not specified)
%     in the config file.

if ~exist('Property','var') %if nothing specified for a parameter..
    Property = ''; %..at least make the variable, so as not to throw errors later
else
    eval([Property ' = ' num2str(Value) ';']);
end

fprintf('\nBeginning XRD Simulation...\n');

% if user specified a configfilename argument to xrd_sim, redefine config to
% be a function handle to the function specified by the string configfilename
if nargin > 0
   config = str2func(configfilename);
else
    config = str2func('config');
end

%% (0.3) conversion factors
deg2rad = pi / 180; %multiply by this to convert degrees to radians.
rad2deg = 1 / deg2rad; %multiply by this to convert radians to degrees.
deg2arcsec = 3600; %multiply by this to convert deg. to arc seconds.
arcsec2deg = 1 / deg2arcsec; %multiply by this to convert arcseconds to degrees.
rad2arcsec = rad2deg * deg2arcsec; %multiply by this to convert radians to arc seconds.
arcsec2rad = 1 / rad2arcsec; %multiply by this to convert arcseconds to radians.
r_e = 2.8179E-5; %classical electron radius (in angstroms)

%Note: to add to the x-ray source wavelengths list, first add to sourceWavelengths_str in
the same syntax, then add anomolous dispersion corrections associated with that radiation
in asf_coefficients.m.
sourceWavelengths_str = {'Cr kalpha1 (2.28970 A)', 'Fe kalpha1 (1.936042 A)', 'Cu kalpha1
(1.540594 A)', 'Mo kalpha1 (0.70930 A)', 'Ag kalpha1 (0.5594075 A)'};

%Note: to add reflections, just add to reflection_str using the same syntax.
reflection_str = {'(002)', '(111)', '(113)', '(004)', '(224)', '(115)', '(044)', '(135)',
'(006)', '(026)', '(335)', '(444)', '(117)'};

%Note: to add substrate orientations, just add to substrateOrientation_str using the same
syntax.
substrateOrientation_str = {'(001)', '(111)'};




%% (1) MATERIALS AND LATTICE PARAMETER DECLARATIONS

layerTypes_str = {'uniform', 'linearly graded', 'superlattice', '(no more layers)'};
```

```
%substrate
%           Note:   to add a substrate, you must add (a) the name and (b) lattice
constant here,
%           but also add (c) the asf_coefficients assignment in section (2) below, and
(d) add
%           the ASFs to asf_coefficients.m if any of the atoms are not already there.

substrates_str =           { 'GaAs',  'Si',    'InP',   'InSb',  'CdTe',  'GaSb' };
substrates_latticeParams = { 5.65340, 5.43108, 5.86900, 6.47940, 6.48100, 6.0960 }; %(in
angstroms)

%           Note: to add uniform, LG, or SL layers, add the lattice constants here, and
make an entry with the elastic constants in section (2) below

%uniform layers
uniformLayers_str =           {'ZnSe', 'ZnS', 'ZnS(x)Se(1-x)',  'In(x)Ga(1-x)As', 'Si(1-
x)Ge(x)',   'Hg(x)Cd(1-x)Te', 'In(x)Ga(1-x)Sb', 'Al(x)Ga(1-x)As', 'In(x)Al(1-x)As'};
uniformLayers_latticeParams = {5.6687, 5.4105 [-0.2582 5.6687], [0.405 5.6534],   [0.2265
5.43108], [-0.02 6.481],    [0.3834 6.096],   [0.0066 5.6534],  [0.3984 5.66]};  % [a b
c]  <==>  ax^2 + bx + c    ;    [b c] <==> bx+c

%linearly graded layers
LGlayers_str =           { 'ZnS(x)Se(1-x)',  'In(x)Ga(1-x)As', 'Si(1-x)Ge(x)',
'Hg(x)Cd(1-x)Te', 'In(x)Ga(1-x)Sb', 'Al(x)Ga(1-x)As', 'In(x)Al(1-x)As'  };
LGlayers_latticeParams = { [-0.2582 5.6687], [0.405 5.6534],   [0.2265 5.43108], [-0.02
6.481],    [0.3834 6.096],   [0.0066 5.6534]   [0.3984 5.66]     }; % [a b c]  <==>  ax^2
+ bx + c

%superlattice layers
SLlayers_str =           { 'ZnS(x)Se(1-x)',  'In(x)Ga(1-x)As', 'Si(1-x)Ge(x)',
'Hg(x)Cd(1-x)Te', 'In(x)Ga(1-x)Sb', 'Al(x)Ga(1-x)As', 'In(x)Al(1-x)As'  };
SLlayers_latticeParams = { [-0.2582 5.6687], [0.405 5.6534],   [0.2265 5.43108], [-0.02
6.481],    [0.3834 6.096],   [0.0066 5.6534]   [0.3984 5.66]     }; % [a b c]  <==>  ax^2
+ bx + c

%--end (1) materials and lattice param. declarations-----------------------



%% (2) USER INPUT

%obtain strain information
strainchoice = config('strain');
if strainchoice == -inf
    strainchoice = menu('Enter the type of strain you would like to use in the
structure:', 'Completely Relaxed (none)', 'Pseudomorphic / Coherently Strained', 'Custom:
constant strain', 'Custom: constant % residual strain (LG layer only)');
end

%obtain choice of substrate
substrateLayer_choice = config('substrate');
if substrateLayer_choice == -inf
    substrateLayer_choice = menu('Choose a substrate (layer 0):', substrates_str);
end %substrateLayer_choice == -inf
switch substrateLayer_choice
    case 1 % GaAs substrate selected
        coeffs_A{1} = asf_coefficients('Ga'); %atom A is Ga
        coeffs_B{1} = asf_coefficients('As'); %atom B is As
    case 2 % Si substrate selected
        coeffs_A{1} = asf_coefficients('Si'); %atom A is Si
        coeffs_B{1} = coeffs_A{1};            %atom B is also Si
    case 3 % InP substrate selected
        coeffs_A{1} = asf_coefficients('In'); %atom A is In
        coeffs_B{1} = asf_coefficients('P');  %atom B is P
    case 4 % InSb substrate selected
        coeffs_A{1} = asf_coefficients('In'); %atom A is In
```

```
        coeffs_B{1} = asf_coefficients('Sb'); %atom B is Sb
    case 5 % CdTe substrate selected
        coeffs_A{1} = asf_coefficients('Cd'); %atom A is Cd
        coeffs_B{1} = asf_coefficients('Te'); %atom B is Te
    case 6 % GaSb substrate selected
        coeffs_A{1} = asf_coefficients('Ga'); %atom A is Ga
        coeffs_B{1} = asf_coefficients('Sb'); %atom B is Sb
    otherwise
        msgbox('You must choose a substrate. Please try again.','Invalid Input',
'error');
        error('Error: you must choose a substrate. Please try again.');
end %switch substrateLayer_choice

layer_type{1} = 'Substrate';
layer_material{1} = substrates_str{substrateLayer_choice};
layer_thickness(1) = Inf;
a0{1} = substrates_latticeParams{substrateLayer_choice}; %lattice parameter of the
substrate -- index {1} corresponds to layer zero
a(1) = a0{1};
c(1) = a0{1};
D(1) = 0;
n_layers = 1;
N_laminae = 1;

%Substrate Orientation
substrateOrientation = config('substrateOrientation');
if substrateOrientation == -inf
    substrateOrientation_choice = menu(['Choose the crystal orientation of the '
layer_material{1} ' substrate:'],substrateOrientation_str);
    substrateOrientation = char(substrateOrientation_str(substrateOrientation_choice));
end
u1 = str2double(substrateOrientation(2)); v1 = str2double(substrateOrientation(3)); w1 =
str2double(substrateOrientation(4));
% /Substrate Orientation



while 1 == 1 %loop until user doesn't want any more layers

    layerType_choice = config(strcat('layer_',num2str(n_layers),'_type'));
    if layerType_choice == -inf
        if exist('configSpecifiedStructure','var') == 0
            layerType_choice = menu(['Choose the type of layer ' num2str(n_layers) ':'],
layerTypes_str);
        else
            uniformLayer_choice = 4; %no more layers!
        end
    else
        configSpecifiedStructure = true;
    end

    switch layerType_choice
        case 1 %Uniform layer selected



            uniformLayer_choice = config(strcat('layer_',num2str(n_layers),'_material'));
            if uniformLayer_choice == -inf
                uniformLayer_choice = menu(['Choose material for layer '
num2str(n_layers) ':'], uniformLayers_str);
            end

            %question user about thickness of layer
            layer_thickness(N_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_thickness'))*10000;
            if layer_thickness(N_laminae + 1) == -inf
                layer_thickness(N_laminae + 1) = str2double(inputdlg(['Enter thickness
for the uniform ' uniformLayers_str{uniformLayer_choice} ' layer (layer '
num2str(n_layers) ') in microns:']))*10000;
```

```
            end

            if strainchoice == 3 %custom strain
                epsilon_par(N_laminae +
1)=config(strcat('layer_',num2str(n_layers),'_inplanestrain'));
                if epsilon_par(N_laminae + 1) == -inf
                    epsilon_par(N_laminae + 1) = str2double(inputdlg(['Enter the in-plane
strain for layer ' num2str(n_layers)]));
                end
            end

            %layer tilt
            epilayer_tilt(N_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_tilt'));
            if epilayer_tilt(N_laminae + 1) == -inf
                epilayer_tilt(N_laminae + 1) = str2double(inputdlg(['Enter the the tilt
(in arc sec) for layer ' num2str(n_layers)]));
            end

            switch uniformLayer_choice
                %TERNARY LAYER NOTE: when coeffs_C exists, coeffs_A has no subscript,
coeffs_B has 'x' subscript, coeffs_C has '1-x' subscript.
                case 1 % ZnSe
                    coeffs_A{N_laminae + 1} = asf_coefficients('Zn');
                    coeffs_B{N_laminae + 1} = asf_coefficients('Se');
                    C11_constants{N_laminae + 1} = [87.2 87.2]; %elastic constants
                    C12_constants{N_laminae + 1} = [52.4 52.4]; %elastic constants
                    x(N_laminae + 1) = 1;
                case 2 % ZnS
                    coeffs_A{N_laminae + 1} = asf_coefficients('Zn');
                    coeffs_B{N_laminae + 1} = asf_coefficients('S');
                    C11_constants{N_laminae + 1} = [104.62 104.62]; %elastic constants
                    C12_constants{N_laminae + 1} = [65.33 65.33]; %elastic constants
                    x(N_laminae + 1) = 1;
                case 3 % ZnS(x)Se(1-x)
                    coeffs_A{N_laminae + 1} = asf_coefficients('Zn');
                    coeffs_B{N_laminae + 1} = asf_coefficients('S');
                    coeffs_C{N_laminae + 1} = asf_coefficients('Se');
                    C11_constants{N_laminae + 1} = [104.62 87.2]; %elastic constants
                    C12_constants{N_laminae + 1} = [65.33 52.4]; %elastic constants
                case 4 %In(x)Ga(1-x)As
                    coeffs_A{N_laminae + 1} = asf_coefficients('As');
                    coeffs_B{N_laminae + 1} = asf_coefficients('In');
                    coeffs_C{N_laminae + 1} = asf_coefficients('Ga');
                    C11_constants{N_laminae + 1} = [83.29 118.4]; %elastic constants
                    C12_constants{N_laminae + 1} = [45.26 53.7]; %elastic constants
                case 5 % Si(1-x)Ge(x)
                    coeffs_A{N_laminae + 1} = [];
                    coeffs_B{N_laminae + 1} = asf_coefficients('Ge');
                    coeffs_C{N_laminae + 1} = asf_coefficients('Si');
                    C11_constants{N_laminae + 1} = [124.0 160.1]; %elastic constants
                    C12_constants{N_laminae + 1} = [41.3 57.8]; %elastic constants
                case 6 % Hg(x)Cd(1-x)Te selected
                    coeffs_A{N_laminae + 1} = asf_coefficients('Te');
                    coeffs_B{N_laminae + 1} = asf_coefficients('Hg');
                    coeffs_C{N_laminae + 1} = asf_coefficients('Cd');
                    C11_constants{N_laminae + 1} = [53.61 53.3]; % [C11(HgTe) C11(CdTe)]
                    C12_constants{N_laminae + 1} = [36.60 36.5]; % [C12(HgTe) C12(CdTe)]
                case 7 % In(x)Ga(1-x)Sb selected
                    coeffs_A{N_laminae + 1} = asf_coefficients('Sb');
                    coeffs_B{N_laminae + 1} = asf_coefficients('In');
                    coeffs_C{N_laminae + 1} = asf_coefficients('Ga');
                    C11_constants{N_laminae + 1} = [65.92 88.50]; % [C11(InSb) C11(GaSb)]
                    C12_constants{N_laminae + 1} = [35.63 40.40]; % [C12(InSb) C12(GaSb)]
                case 8 % Al(x)Ga(1-x)As selected
                    coeffs_A{N_laminae + 1} = asf_coefficients('As');
                    coeffs_B{N_laminae + 1} = asf_coefficients('Al');
                    coeffs_C{N_laminae + 1} = asf_coefficients('Ga');
                    C11_constants{N_laminae + 1} = [125 118.4]; % [C11(AlAs) C11(GaAs)]
                    C12_constants{N_laminae + 1} = [53.4 53.7]; % [C12(AlAs) C12(GaAs)]
```

```
                case 9 % In(x)Al(1-x)As selected
                    coeffs_A{N_laminae + 1} = asf_coefficients('As');
                    coeffs_B{N_laminae + 1} = asf_coefficients('In');
                    coeffs_C{N_laminae + 1} = asf_coefficients('Al');
                    C11_constants{N_laminae + 1} = [83.29 125];  % [C11(InAs) C11(AlAs)]
                    C12_constants{N_laminae + 1} = [45.26 53.4]; % [C12(InAs) C12(AlAs)]
                otherwise
                    msgbox('You failed to make a selection. Please try again.','Invalid
Input', 'error');
                    n_layers = n_layers - 1;
                    N_laminae = N_laminae - 1;
            end %switch uniformLayer_choice

            layer_material{N_laminae + 1} = uniformLayers_str{uniformLayer_choice};

            if numel(findstr(layer_material{N_laminae + 1}, '(x)')) == 1 %check if the
layer is a ternary
                x(N_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_composition'));
                if x(N_laminae + 1) == -inf
                    x(N_laminae + 1) = str2double(inputdlg(['Enter composition of layer '
num2str(n_layers) ':']));
                end
                %x(N_laminae + 1) = obtainComposition(n_layers); %old way of getting
composition, doesn't work properly since relays on config.m
            end

            a0{N_laminae + 1} = uniformLayers_latticeParams{uniformLayer_choice};

        case 2 %Linearly graded layer selected

            LGlayer_choice = config(strcat('layer_',num2str(n_layers),'_material'));
            if LGlayer_choice == -inf
                LGlayer_choice = menu(['Choose material for the '
layerTypes_str{layerType_choice} ' layer (layer ' num2str(n_layers) '):'], LGlayers_str);
            end

            %question user about thickness of layer
            layer_thickness(N_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_thickness'))*10000;
            if layer_thickness(N_laminae + 1) == -inf
                layer_thickness(N_laminae + 1) = str2double(inputdlg(['Enter thickness
for the linearly graded ' LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers)
') in microns:']))*10000;
            end

            n_lam = config(strcat('layer_',num2str(n_layers),'_numberOfLamina'));
            if n_lam == -inf
                n_lam = str2double(inputdlg(['Enter the desired number of laminae for the
' LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers) '):']));
            end

            x_start = config(strcat('layer_',num2str(n_layers),'_startingComposition'));
            if x_start == -inf
                x_start = str2double(inputdlg(['Enter starting composition for the '
LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers) '):']));
            end
            x_end = config(strcat('layer_',num2str(n_layers),'_endingComposition'));
            if x_end == -inf
                x_end = str2double(inputdlg(['Enter ending composition for the '
LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers) '):']));
            end

            if strainchoice == 3 %custom strain
                inplanestrain = str2double(inputdlg('Enter the in-plane strain for all
the laminae in the graded layer:'));
```

```
                epsilon_par((N_laminae + 1):(N_laminae + n_lam)) = inplanestrain; %set in
plane strain for all laminae in the GL
            end
            if strainchoice == 4 %custom percent residual strain
                percentResidualStrain = str2double(inputdlg('Enter the percent residual
strain for the layer:'));
                fractionalStrain((N_laminae + 1):(N_laminae + n_lam)) =
(percentResidualStrain / 100);
            end

            layer_thickness((N_laminae + 1):(N_laminae + n_lam)) =
layer_thickness(N_laminae+1) / n_lam; %set thickness of all laminae in the GL equal to
(total thickness)/(number of laminae)

            for lam_counter = N_laminae:(N_laminae + n_lam - 1)
                switch LGlayer_choice
                    %TERNARY LAYER NOTE: when coeffs_C exists, coeffs_A has no subscript,
coeffs_B has 'x' subscript, coeffs_C has '1-x' subscript.
                    case 1 % ZnS(x)Se(1-x)
                        coeffs_A{lam_counter + 1} = asf_coefficients('Zn');
                        coeffs_B{lam_counter + 1} = asf_coefficients('S');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Se');
                        %C11 constants is the two-element vector: [ C11(ZnS)  C11(ZnSe) ]
for later calculation of x*C11(ZnS) + (1-x)*C11(ZnSe).
                        C11_constants{lam_counter + 1} = [104.62 87.2];%[ C11(ZnS)
C11(ZnSe) ]
                        C12_constants{lam_counter + 1} = [65.33 52.4]; %[ C12(ZnS)
C12(ZnSe) ]
                    case 2 % In(x)Ga(1-x)As
                        coeffs_A{lam_counter + 1} = asf_coefficients('As');
                        coeffs_B{lam_counter + 1} = asf_coefficients('In');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Ga');
                        C11_constants{lam_counter + 1} = [83.29 118.4]; %[ C11(InAs)
C11(GaAs) ]
                        C12_constants{lam_counter + 1} = [45.26 53.7];  %[ C12(InAs)
C12(GaAs) ]
                    case 3 % Si(1-x)Ge(x)
                        coeffs_A{lam_counter + 1} = [];
                        coeffs_B{lam_counter + 1} = asf_coefficients('Ge');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Si');
                        C11_constants{lam_counter + 1} = [124.0 160.1]; %[ C11(Ge)
C11(Si) ]
                        C12_constants{lam_counter + 1} = [41.3 57.8];   %[ C12(Ge)
C12(Si) ]
                    case 4 % Hg(x)Cd(1-x)Te
                        coeffs_A{lam_counter + 1} = asf_coefficients('Te');
                        coeffs_B{lam_counter + 1} = asf_coefficients('Hg');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Cd');
                        C11_constants{lam_counter + 1} = [53.61 53.3]; % [C11(HgTe)
C11(CdTe)]
                        C12_constants{lam_counter + 1} = [36.60 36.5]; % [C12(HgTe)
C12(CdTe)]
                    case 5 % In(x)Ga(1-x)Sb
                        coeffs_A{lam_counter + 1} = asf_coefficients('Sb');
                        coeffs_B{lam_counter + 1} = asf_coefficients('In');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Ga');
                        C11_constants{lam_counter + 1} = [65.92 88.50]; % [C11(InSb)
C11(GaSb)]
                        C12_constants{lam_counter + 1} = [35.63 40.40]; % [C12(InSb)
C12(GaSb)]
                    case 6 % Al(x)Ga(1-x)As selected
                        coeffs_A{lam_counter + 1} = asf_coefficients('As');
                        coeffs_B{lam_counter + 1} = asf_coefficients('Al');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Ga');
                        C11_constants{lam_counter + 1} = [125 118.4]; % [C11(AlAs)
C11(GaAs)]
                        C12_constants{lam_counter + 1} = [53.4 53.7]; % [C12(AlAs)
C12(GaAs)]
                    case 7 % In(x)Al(1-x)As selected
                        coeffs_A{lam_counter + 1} = asf_coefficients('As');
```

```
                        coeffs_B{lam_counter + 1} = asf_coefficients('In');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Al');
                        C11_constants{lam_counter + 1} = [83.29 125];  % [C11(InAs)
C11(AlAs)]
                        C12_constants{lam_counter + 1} = [45.26 53.4]; % [C12(InAs)
C12(AlAs)]
                end

                %Grade the composition x from x_start to x_end
                x(lam_counter + 1) = x_start + (lam_counter - N_laminae) * (x_end -
x_start) / (n_lam-1);

                %set strings describing the layer
                layer_type{lam_counter + 1} = layerTypes_str{layerType_choice};
                layer_material{lam_counter + 1} = LGlayers_str{LGlayer_choice};

                a0{lam_counter + 1} = LGlayers_latticeParams{LGlayer_choice}; %set
lattice parameter

                %layer tilt
                epilayer_tilt(lam_counter + 1) =
config(strcat('layer_',num2str(n_layers),'_tilt'));
                if epilayer_tilt(lam_counter + 1) == -inf
                    epilayer_tilt(lam_counter + 1) = str2double(inputdlg(['Enter the the
tilt (in arc sec) for layer ' num2str(n_layers)]));
                end

            end %for lam_counter = N_laminae:(N_laminae + n_lam - 1)

            N_laminae = N_laminae + n_lam - 1;

        case 3 %Superlattice layer selected

            %the materials, thicknesses, and compositions are all indexed
            %as (1) for atom A and (2) for atom B.
            SLlayer_choice(1) = config(strcat('layer_',num2str(n_layers),'_material_A'));
            if SLlayer_choice(1) == -inf
                SLlayer_choice(1) = menu(['Choose material for atom A in the superlattice
layer (layer ' num2str(n_layers) '):'], SLlayers_str);
            end
            SLlayer_choice(2) = config(strcat('layer_',num2str(n_layers),'_material_B'));
            if SLlayer_choice(2) == -inf
                SLlayer_choice(2) = menu(['Choose material for atom B in the superlattice
layer (layer ' num2str(n_layers) '):'], SLlayers_str);
            end

            SLthickness(1) = config(strcat('layer_',num2str(n_layers),'_thickness_A')) *
10;
            if SLthickness(1) == -inf
                SLthickness(1) = 10 * str2double(inputdlg(['Enter the thickness for
superlattice layer A in one period of the ' SLlayers_str{SLlayer_choice} ' layer (layer '
num2str(n_layers) ') in nanometers:']));
            end
            SLthickness(2) = config(strcat('layer_',num2str(n_layers),'_thickness_B')) *
10;
            if SLthickness(2) == -inf
                SLthickness(2) = 10 * str2double(inputdlg(['Enter the thickness for
superlattice layer B in one period of the ' SLlayers_str{SLlayer_choice} ' layer (layer '
num2str(n_layers) ') in nanometers:']));
            end

            SLx(1) = config(strcat('layer_',num2str(n_layers),'_composition_A'));
            if SLx(1) == -inf
                SLx(1) = str2double(inputdlg(['Enter the composition for atom A of the '
SLlayers_str{SLlayer_choice} ' superlattice layer (layer ' num2str(n_layers) '):']));
            end
```

```
            SLx(2) = config(strcat('layer_',num2str(n_layers),'_composition_B'));
            if SLx(2) == -inf
                SLx(2) = str2double(inputdlg(['Enter the composition for atom B of the '
SLlayers_str{SLlayer_choice} ' superlattice layer (layer ' num2str(n_layers) '):']));
            end

            num_periods = config(strcat('layer_',num2str(n_layers),'_numberOfPeriods'));
            if num_periods == -inf
                num_periods = str2double(inputdlg(['Enter the number of periods for the '
SLlayers_str{SLlayer_choice} ' superlattice layer (layer ' num2str(n_layers) '):']));
            end

            if strainchoice == 3 %custom strain
                inplanestrain =
config(strcat('layer_',num2str(n_layers),'_inplanestrain'));
                if inplanestrain == -inf
                    inplanestrain = str2double(inputdlg('Enter the in-plane strain for
the superlattice layer:'));
                end
            end

            for period_counter = N_laminae:1:(N_laminae + 2 * num_periods - 1)

                atomSelector = mod(period_counter-N_laminae,2) + 1; %this shall alternate
between 1 and 2, respectively corresponding to atoms A and B.

                switch SLlayer_choice(atomSelector)
                    %TERNARY LAYER NOTE: when coeffs_C exists, coeffs_A has no subscript,
coeffs_B has 'x' subscript, coeffs_C has '1-x' subscript.
                    case 1 %ZnS(x)Se(1-x) selected
                        coeffs_A{period_counter + 1} = asf_coefficients('Zn');
                        coeffs_B{period_counter + 1} = asf_coefficients('S');
                        coeffs_C{period_counter + 1} = asf_coefficients('Se');
                        C11_constants{period_counter + 1} = [104.62 87.2];%[ C11(ZnS)
C11(ZnSe) ]
                        C12_constants{period_counter + 1} = [65.33 52.4]; %[ C12(ZnS)
C12(ZnSe) ]
                    case 2 %In(x)Ga(1-x)As selected
                        coeffs_A{period_counter + 1} = asf_coefficients('As');
                        coeffs_B{period_counter + 1} = asf_coefficients('In');
                        coeffs_C{period_counter + 1} = asf_coefficients('Ga');
                        C11_constants{period_counter + 1} = [83.29 118.4]; %[ C11(InAs)
C11(GaAs) ]
                        C12_constants{period_counter + 1} = [45.26 53.7]; %[ C12(InAs)
C12(GaAs) ]
                    case 3 %Si(1-x)Ge(x) selected
                        coeffs_A{period_counter + 1} = [];
                        coeffs_B{period_counter + 1} = asf_coefficients('Ge');
                        coeffs_C{period_counter + 1} = asf_coefficients('Si');
                        C11_constants{period_counter + 1} = [124.0 160.1]; %[ C11(Ge)
C11(Si) ]
                        C12_constants{period_counter + 1} = [41.3 57.8]; %[ C12(Ge)
C12(Si) ]
                    case 4 % Hg(x)Cd(1-x)Te
                        coeffs_A{period_counter + 1} = asf_coefficients('Te');
                        coeffs_B{period_counter + 1} = asf_coefficients('Hg');
                        coeffs_C{period_counter + 1} = asf_coefficients('Cd');
                        C11_constants{period_counter + 1} = [53.61 53.3]; % [C11(HgTe)
C11(CdTe)]
                        C12_constants{period_counter + 1} = [36.60 36.5]; % [C12(HgTe)
C12(CdTe)]
                    case 5 % In(x)Ga(1-x)Sb
                        coeffs_A{period_counter + 1} = asf_coefficients('Sb');
                        coeffs_B{period_counter + 1} = asf_coefficients('In');
                        coeffs_C{period_counter + 1} = asf_coefficients('Ga');
                        C11_constants{period_counter + 1} = [65.92 88.50]; % [C11(InSb)
C11(GaSb)]
                        C12_constants{period_counter + 1} = [35.63 40.40]; % [C12(InSb)
C12(GaSb)]
```

```
                    case 6 % Al(x)Ga(1-x)As selected
                        coeffs_A{period_counter + 1} = asf_coefficients('As');
                        coeffs_B{period_counter + 1} = asf_coefficients('Al');
                        coeffs_C{period_counter + 1} = asf_coefficients('Ga');
                        C11_constants{period_counter + 1} = [125 118.4]; % [C11(AlAs)
C11(GaAs)]
                        C12_constants{period_counter + 1} = [53.4 53.7]; % [C12(AlAs)
C12(GaAs)]
                    case 7 % In(x)Al(1-x)As selected
                        coeffs_A{period_counter + 1} = asf_coefficients('As');
                        coeffs_B{period_counter + 1} = asf_coefficients('In');
                        coeffs_C{period_counter + 1} = asf_coefficients('Al');
                        C11_constants{period_counter + 1} = [83.29 125];  % [C11(InAs)
C11(AlAs)]
                        C12_constants{period_counter + 1} = [45.26 53.4]; % [C12(InAs)
C12(AlAs)]
                end

                layer_type{period_counter + 1} = layerTypes_str{layerType_choice}; %layer
type (superlattice)

                layer_material{period_counter + 1} =
SLlayers_str{SLlayer_choice(atomSelector)}; %material

                a0{period_counter + 1} =
SLlayers_latticeParams{SLlayer_choice(atomSelector)}; %lattice parameter

                x(period_counter + 1) = SLx(atomSelector); %composition

                if strainchoice == 3 %custom strain selected, so calculate in-plane
strain for superlattice layer

                    a0A = latticeparam_calc(a0{period_counter + 1}, SLx(1));
                    a0B = latticeparam_calc(a0{period_counter + 2}, SLx(2));
                    a0_ave = (a0A*SLthickness(1) + a0B*SLthickness(2)) / (SLthickness(1)
+ SLthickness(2));
                    a_SL = a0_ave * (1 + inplanestrain);
                    if atomSelector == 1
                        epsilon_par(period_counter + 1) = (a_SL - a0A) / a0A;
                    else
                        epsilon_par(period_counter + 2) = (a_SL - a0B) / a0B;
                    end
                end

                layer_thickness(period_counter + 1) = SLthickness(atomSelector);



                %layer tilt
                epilayer_tilt(period_counter + 1) =
config(strcat('layer_',num2str(n_layers),'_tilt'));
                if epilayer_tilt(period_counter + 1) == -inf
                    epilayer_tilt(period_counter + 1) = str2double(inputdlg(['Enter the
the tilt (in arc sec) for layer ' num2str(n_layers)]));
                end


            end

            N_laminae = N_laminae + 2 * num_periods - 1;

        otherwise % no more layers selected
            break;
    end %switch layerType_choice

    layer_type{N_laminae + 1} = layerTypes_str{layerType_choice}; %set layer type string
```

```
    n_layers = n_layers + 1; %increment because more layers are desired
    N_laminae = N_laminae + 1;
end % while 1 == 1

% Reflection (hkl)
reflection = config('reflection');
if reflection == -inf
    reflection_choice = menu('Choose the desired reflection:',reflection_str);
    reflection = char(reflection_str(reflection_choice));
end
h = str2double(reflection(2)); k = str2double(reflection(3)); l =
str2double(reflection(4));
% /Reflections



% Angle
theta_Start = config('theta_Start') * arcsec2rad;
if theta_Start == -inf
    theta_Start = str2double(inputdlg('Enter the starting angle, (theta-theta_B, in arc
seconds):')) * arcsec2rad;
end
theta_End = config('theta_End') * arcsec2rad;
if theta_End == -inf
    theta_End = str2double(inputdlg('Enter the ending angle (theta-theta_B, in arc
seconds):')) * arcsec2rad;
end

broadening_width = config('broadening_width');
if broadening_width > 0
    theta_Start = theta_Start - 2 * (broadening_width * arcsec2rad);
    theta_End   = theta_End   + 2 * (broadening_width * arcsec2rad);
end



if strcmp(Property,'azimuth')
    azimuth = Value;
else
    azimuth = config('azimuth');
    if azimuth==-inf; azimuth = str2double(inputdlg('Enter the azimuth in degrees'));
end;
end

psi = azimuth * deg2rad;

% /Angle



%Number of Points Per Arcsecond
numPtsPerArcSec = config('numPtsPerArcSec');
if numPtsPerArcSec == -inf
    str2double(inputdlg('Enter the number of rocking curve intensity points to calculate
per arcsecond of theta-theta_B:'));
end

% Radiation
radiationChoice = config('radiation');
if radiationChoice == -inf
    radiationChoice = menu('Choose a radiation source wavelength:',
sourceWavelengths_str);
end
idxLambdaStart = strfind(sourceWavelengths_str{radiationChoice}, '(') + 1;
idxLambdaEnd = strfind(sourceWavelengths_str{radiationChoice}, 'A)') - 1;
lambda = str2double(sourceWavelengths_str{radiationChoice}(idxLambdaStart:idxLambdaEnd)
);
```

```
% /Radiation

% Simulation Type
simulationChoice = config('simulation');
if simulationChoice == -inf
    simulationChoice = menu('Choose the simulation type:', {'Dynamical (Takagi-Taupin)',
'Kinematical (Bartels-Hornstra-Lobeek/Speriosu-Vreeland)', 'Both'});
end



%% (3) COMPUTE SCATTERING AMPLITUDE OF THE SUBSTRATE USING DARWIN-PRINS FORMULA
w = waitbar(0,'Calculating rocking curve for substrate','CreateCancelBtn',
'setappdata(gcbf,''canceling'',1)');
setappdata(w,'canceling',0);
%define angles to calculate intensity at; calculate Bragg Angle theta_B for the substrate
theta_B(1) = asin(lambda*sqrt(h^2+k^2+l^2)/(2*a0{1})); %Bragg angle for substrate (index
(1) corresponds to layer zero, the substrate)
theta_Spread = theta_End - theta_Start;
numPts = round(numPtsPerArcSec * (theta_Spread * rad2arcsec));
theta_Step = theta_Spread / (numPts - 1); %(in radians)
theta = (theta_B(1)+theta_Start:theta_Step:theta_B(1)+theta_End); %vector of angles (in
radians)

%calculate substrate ASFs, structure factors
f_A(1) = asf_calc(coeffs_A{1}, sin(theta_B(1))/lambda, radiationChoice);
f_B(1) = asf_calc(coeffs_B{1}, sin(theta_B(1))/lambda, radiationChoice);
F_o(1) = sf_calc(f_A(1), f_B(1), 0,0,0);
F_H(1) = sf_calc(f_A(1), f_B(1), h,k,l);
F_HBar(1) = sf_calc(f_A(1), f_B(1), -h,-k,-l);



%calculate substrate deviation parameter eta_s
phi = acos((u1*h+v1*k+w1*l)/sqrt((u1^2+v1^2+w1^2)*(h^2+k^2+l^2)));
if (phi > theta_B)
    msgbox('Sorry, this reflection is invalid because phi is greater than the Bragg angle
(theta_B).');
    return;
end
gamma_o(1) = sin(theta_B(1) + phi);
gamma_H(1) = -sin(theta_B(1) - phi);
b(1) = gamma_o(1) / gamma_H(1);  % asymmetry factor

V(1) = a0{1}^3; %volume of cubic crystal (in angstroms^3)
Gamma(1) = (r_e * lambda^2) / (pi * V(1));
C(1) = 0.5 * (1 + cos(2*theta_B(1))^2);  % polarization factor for random polarization

eta_num = -b * (theta-theta_B(1)) * sin(2 * theta_B(1)) - 0.5 * (1-b(1)) * Gamma(1) *
F_o(1);
eta_den = sqrt(abs(b(1))) * C(1) * Gamma(1) * sqrt(F_H(1) * F_HBar(1));
eta(1,:) = eta_num / eta_den;



%calculate substrate scattering amplitude
X0 = eta(1,:) - sign(real(eta(1,:))) .* sqrt(eta(1,:) .^ 2 - 1);

%preallocate dynamical and kinematical arrays and populate the first layer with the
substrate amplitude.
X_n = zeros(N_laminae,numPts); %preallocate
X_n(1,:) = X0;                 %set the first layer with substrate scattering amplitude



%% (4) PERFORM DYNAMICAL CALCULATIONS RECURSIVELY
```

```
if N_laminae > 1 %check to make sure more laminae than just substrate exist

    %% first loop thru and find dislocation densities, so know max beforehand
    for n = 2:N_laminae % n=2 corresponds to layer 1, the first layer above substrate
        TDD_mode = config('TDD_mode');
        if TDD_mode == -inf; TDD_mode = str2double(inputdlg('Enter the threading
dislocation density mode. (1) symmetric, (2) asymmetric')); end;

        if TDD_mode == 1 % symmetric

            if n==2 %the first iteration of the layer loop
                if ~exist('TDDuniform','var'); TDDuniform = config('TDDuniform'); end;
                if (TDDuniform ~= -inf)
                    TDDinitial = TDDuniform;
                    TDDfinal = TDDuniform;
                else
                    if ~exist('TDDinitial','var'); TDDinitial = config('TDDinitial');
end;
                    if ~exist('TDDfinal','var'); TDDfinal   = config('TDDfinal'  ); end;
                end
            end
            if ((TDDinitial ~= -inf) && (TDDfinal ~= -inf)) %then user specified a linear
profile
                if N_laminae > 2
                    TDD = TDDinitial + (n-2) * (TDDfinal - TDDinitial) / (N_laminae-2);
                else
                    TDD=TDDinitial;
                end
            else %no profile specified, so ask for TDD for each layer
                TDD = config(strcat('layer_',num2str(n-1),'_TDD'));
                if TDD == -inf
                    TDD = str2double(inputdlg('Enter the threading dislocation density in
10^8/cm^2:'));
                    if ( isempty(TDD) || isnan(TDD) ); TDD = 0; end
                end
            end

            D(n) = TDD * 10^(-8); %convert input which is in 10^8/cm^2 to A^-2.  1 cm^-2
= 10^-16 A^-2

        elseif TDD_mode == 2 % asymmetric

            TDD_A = config(strcat('layer_',num2str(n-1),'_TDD_A'));
            TDD_B = config(strcat('layer_',num2str(n-1),'_TDD_B'));

            D_A(n) = TDD_A * 10^(-8); %convert input which is in 10^8/cm^2 to A^-2.  1
cm^-2 = 10^-16 A^-2
            D_B(n) = TDD_B * 10^(-8); %convert input which is in 10^8/cm^2 to A^-2.  1
cm^-2 = 10^-16 A^-2

            D(n) = D_A(n) + D_B(n); %convert input which is in 10^8/cm^2 to A^-2.  1 cm^-
2 = 10^-16 A^-2

        else
            TDD = 0;
            D(n)= 0;
            warning('TDD not specified, using zero (perfect crystal).');

        end %if TDD_mode == 1 % symmetric

    end % for n = 2:N_laminae
    Dmax = max(D) * 1e+16; %convert to cm^-2


%     %disp D profile
```

```
%     figure;
%     stem(0:(length(D)-1),D*1e8);
%     xlabel 'layer no.';
%     ylabel 'D (10^8 cm-2)';
%


    %% TT-eqn. recursion
    tStart=tic; %start stopwatch


    for n = 2:N_laminae % n=2 corresponds to layer 1, the first layer above substrate


        waitbar((n-2)/(N_laminae-1),w,['Calculating rocking curve for lamina ' num2str(n-
1)],'CreateCancelBtn', 'setappdata(gcbf,''canceling'',1)');
        if getappdata(w,'canceling')
            delete(w);
            error('User canceled operation.');
        end


        if (numel(a0{n}) > 1) %a0n used for strain calculations
            a0n = latticeparam_calc(a0{n}, x(n));
        else
            a0n = a0{n};
        end


        %determine in-plane and out-of-plane lattice constants
        switch strainchoice
            case 1 %relaxed
                a(n) = a0n;
                c(n) = a0n;
            case 2 %coherent strain
                a(n) = a0{1};
                epsilon_par(n) = (a(n) - a0n) / a0n;
                C11(n) = elasticconstant_calc(C11_constants{n}, x(n));
                C12(n) = elasticconstant_calc(C12_constants{n}, x(n));
                epsilon_perp(n) = -2 * epsilon_par(n) * (C12(n) / C11(n));
                c(n) = a0n * (1 + epsilon_perp(n));
            case 3 %custom strain
                a(n) = a0n * (1 + epsilon_par(n));
                C11(n) = elasticconstant_calc(C11_constants{n}, x(n));
                C12(n) = elasticconstant_calc(C12_constants{n}, x(n));
                epsilon_perp(n) = -2 * epsilon_par(n) * (C12(n) / C11(n));
                c(n) = a0n * (1 + epsilon_perp(n));
            case 4 %custom strain
                latticeMismatch(n) = (a0{1} - a0n) / a0n;
                epsilon_par(n) = fractionalStrain(n) * latticeMismatch(n);
                a(n) = a0n * (1 + epsilon_par(n));
                C11(n) = elasticconstant_calc(C11_constants{n}, x(n));
                C12(n) = elasticconstant_calc(C12_constants{n}, x(n));
                epsilon_perp(n) = -2 * epsilon_par(n) * (C12(n) / C11(n));
                c(n) = a0n * (1 + epsilon_perp(n));
        end


        if numel(a0{n}) > 1 %ternary layer (if > 1, a nonscaler value exists in a0{n},
indicating polynomial form of relaxed lattice parameter)
            a0{n} = latticeparam_calc(a0{n}, x(n)); %calculate lattice parameter based on
current composition, x(n)


            theta_B(n) = asin((lambda/2) * sqrt( (h/a(n))^2 + (k/a(n))^2 + (l/c(n))^2) );
%Bragg angle for nth laminate


            %calculate ASFs for a ternary layer
            if numel(coeffs_A{n}) == 0 %two-atom ternary layer, such as Si(1-x)Ge(x)
                f_A(n) = asf_calc2(coeffs_B{n}, coeffs_C{n}, x(n),
sin(theta_B(n))/lambda, radiationChoice);
                f_B(n) = f_A(n);
            else %three atom ternary layer, such as ZnS(x)Se(1-x)
                f_A(n) = asf_calc(coeffs_A{n}, sin(theta_B(n))/lambda, radiationChoice);
```

```
                f_B(n) = asf_calc2(coeffs_B{n}, coeffs_C{n}, x(n),
sin(theta_B(n))/lambda, radiationChoice);
            end
        else
            theta_B(n) = asin((lambda/2) * sqrt( (h/a(n))^2 + (k/a(n))^2 + (l/c(n))^2) );
%Bragg angle for nth laminate

            %calculate ASFs
            f_A(n) = asf_calc(coeffs_A{n}, sin(theta_B(n))/lambda, radiationChoice); %ASF
for atom A
            f_B(n) = asf_calc(coeffs_B{n}, sin(theta_B(n))/lambda, radiationChoice); %ASF
for atom B
        end

        delta_phi_tet(n) = acos( (l/c(n)) / sqrt( (h/a(n))^2 + (k/a(n))^2 + (l/c(n))^2 )
) - acos(l/sqrt( h^2 + k^2 + l^2));

        delta_phi_tilt(n) = epilayer_tilt(n) * arcsec2rad;

        Gamma(n) = (r_e * lambda^2) / (pi * a0{n}^3); % (capital Gamma)

        %direction cosines, asymmetry factor
        psi_0 = 0;
        gamma_o(n) = sin(theta_B(n) + (phi + delta_phi_tet(n) +
delta_phi_tilt(n)*cos(psi-psi_0)));
        gamma_H(n) = -sin(theta_B(n) - (phi + delta_phi_tet(n) +
delta_phi_tilt(n)*cos(psi-psi_0)));
        b(n) = gamma_o(n) / gamma_H(n);  % asymmetry factor

        %structure factors
        F_H(n) = sf_calc(f_A(n),f_B(n), h,k,l); %SF
        F_HBar(n) = sf_calc(f_A(n),f_B(n), -h,-k,-l);
        F_o(n) = sf_calc(f_A(n),f_B(n), 0,0,0);



        %% (4.1) Deviation Parameter
        C(n) = 0.5 * (1 + cos(2*theta_B(n))^2);  % polarization factor, for random
polarization
        T(n) = layer_thickness(n) * C(n) * (pi * Gamma(n) * sqrt(F_H(n) * F_HBar(n))) /
(lambda * sqrt(abs(gamma_o(n) * gamma_H(n)))); %thickness parameter

        % Threading Dislocation Density
        if (D(n)==0) %user entered zero dislocation density or didn't specify

            eta_num = -b(n) * (theta - (theta_B(n) + delta_phi_tet(n) +
delta_phi_tilt(n)*cos(psi-psi_0))) * sin(2*theta_B(n)) - 0.5*(1-b(n))*Gamma(n)*F_o(n);
            eta_den = sqrt(abs(b(n))) * C(n) * Gamma(n) * sqrt(F_H(n) * F_HBar(n));
            eta(n,:) = eta_num ./ eta_den;

        else % use mosaic crystal model for dislocated crystal

            %alpha-beta mesh parameters
            Nmin = 21;
            Dcrossover = 1e9; %cm^-2
            %Nalpha=ceil(Nmin*sqrt(Dmax/Dcrossover));
            Nalpha = Nmin;
            if exist('NalphaARG','var'); Nalpha = NalphaARG; end;
            if (mod(Nalpha,2) == 0); Nalpha = (Nalpha + 1); end %also make sure it's odd



            Nbeta=Nalpha;

            NSig = 3;
```

```
            fprintf('Computing effective deviation parameter for layer %i. \n', (n-1));
            fprintf('Nsig=%f; Nalpha=%i; Nbeta=%i\n', NSig, Nalpha, Nbeta);

            dislocation_distribution_type = config('dislocation_distribution_type');

            if TDD_mode == 1 % symmetric
                sigma_alpha = ( a0n*sqrt(pi*D(n)) ) / (2*sqrt(2));
                sigma_epsilon  = ( 0.127*a0n*sqrt(D(n)*abs(log(2E-7*sqrt(D(n)*10^16)))) *
tan(theta_B(n)) ) / sqrt(2);
            elseif TDD_mode == 2 % asymmetric
                if ~strcmp(reflection,'(004)'); error('Asymmetric dislocation profile not
yet implemented for reflections other than 004!'); end;
                gamma_A = 45 * deg2rad;
                gamma_B = 45 * deg2rad;
                sigma_alpha = (a0n/sqrt(2))*sqrt(pi)*sqrt(
D_A(n)*cos(psi)^2*cos(gamma_A)^2 + D_B(n)*sin(psi)^2*cos(gamma_B)^2 );
                sigma_epsilon  = ( 0.127*a0n*sqrt(D(n)*abs(log(2E-7*sqrt(D(n)*10^16)))) *
tan(theta_B(n)) ) / sqrt(2);
            end

            alpha = NSig * sigma_alpha * linspace(-1,1,Nalpha);
            beta  = NSig * sigma_epsilon * linspace(-1,1,Nbeta);
            alpha_step = alpha(2)-alpha(1);
            beta_step = beta(2)-beta(1);

            for idx_i = 1:length(alpha)
                for idx_j = 1:length(beta)
                    eta_num = -b(n) * (theta - (theta_B(n) + delta_phi_tet(n) +
delta_phi_tilt(n)*cos(psi-psi_0) + alpha(idx_i) - beta(idx_j))) * sin(2*theta_B(n)) -
0.5*(1-b(n))*Gamma(n)*F_o(n);
                    eta_den = sqrt(abs(b(n))) * C(n) * Gamma(n) * sqrt(F_H(n) *
F_HBar(n));
                    eta_nij(n,idx_i,idx_j,:) = eta_num ./ eta_den;
                end
            end

        end %if (D(n)==0)


        %% (4.2) Scattering Amplitude
        if ((simulationChoice == 1) || (simulationChoice == 3)) %Dynamical: Takagi-Taupin

            if (D(n)==0) %user entered zero dislocation density or didn't specify

                S_1n = (X_n(n-1,:) - eta(n,:) + sqrt(eta(n,:).^2-1)) .* exp(-1i * T(n) *
sqrt(eta(n,:).^2-1));
                S_2n = (X_n(n-1,:) - eta(n,:) - sqrt(eta(n,:).^2-1)) .* exp(1i * T(n)  *
sqrt(eta(n,:).^2-1));
                X_n(n,:) = eta(n,:) + sqrt(eta(n,:).^2-1) .* ((S_1n + S_2n) ./ (S_1n -
S_2n));

            else
                %% calc. scattering amplitude for each alpha, beta
                %h_super = figure('Renderer','zbuffer'); x_scale = (theta - theta_B(1)) *
rad2arcsec; zz=0;
                if ~exist('X_nij','var'); X_nij =
zeros(N_laminae,length(alpha),length(beta),numPts); end
                for idx_i = 1:length(alpha)
                    for idx_j = 1:length(beta)
                        enij = squeeze(eta_nij(n,idx_i,idx_j,:)).' ; %remove singleton
dimensions so can do math with it as a vector. Note: .' is the NON-conjugate transpose
                        if n==2 %get it started with the substrate scattering amplitude
                            S_1nij = (X_n(n-1,:) - enij + sqrt(enij.^2-1)) .* exp(-1i *
T(n) * sqrt(enij.^2-1));
```

```
                            S_2nij = (X_n(n-1,:) - enij - sqrt(enij.^2-1)) .* exp(1i  *
T(n) * sqrt(enij.^2-1));
                        else
                            S_1nij = (squeeze(X_nij(n-1,idx_i,idx_j,:)).' - enij +
sqrt(enij.^2-1)) .* exp(-1i * T(n) * sqrt(enij.^2-1));
                            S_2nij = (squeeze(X_nij(n-1,idx_i,idx_j,:)).' - enij -
sqrt(enij.^2-1)) .* exp(1i  * T(n) * sqrt(enij.^2-1));
                        end
                        X_nij(n,idx_i,idx_j,:) = enij + sqrt(enij.^2-1) .* ((S_1nij +
S_2nij) ./ (S_1nij - S_2nij));
                        %zz=zz+1;
semilogy(x_scale,abs(squeeze(X_Nij(idx_i,idx_j,:)).').^2,getNextColorAndLinestyle(zz));
hold on; axis tight manual; F(zz) = getframe(gcf);
                    end
                end

                %figure;
                %movie(gcf,F,10,5);
                %movie2avi([F],'summing_of_XN.avi','fps',10);
            end
        end

        %
        %figure; stem(alpha,W_alpha); xlabel('\alpha'); ylabel('W_\alpha');
title(['Angular weighting function W_\alpha.        \sigma_\alpha = '
num2str(sigma_alpha)]);
        %figure; stem(beta, W_epsilon); xlabel('\beta'); ylabel('W_\epsilon');
title(['Angular weighting function W_\epsilon.        \sigma_\epsilon = '
num2str(sigma_epsilon)]);



        if ((simulationChoice == 2) || (simulationChoice == 3)) %Kinematical: Bartels-
Hornstra-Lobeek/Speriosu-Vreeland
            %Xkin(n,:) = Xkin(n-1,:) .* exp(-1i .*2 .* eta(n,:) .* T(n)) + (1 - exp(1i .*
2 .* eta(n,:) .* T(n))) ./ (2*eta(n,:));
            error('Kinematical simulation is depreciated. If needed, revert to previous
revision.');
        end

    end % for n = 2:N_laminae % n=2 corresponds to layer 1, the first layer above
substrate



%     if (D(n)~=0) %user entered zero dislocation density or didn't specify
%
%         %% correct phase in scattering amplitude for each alpha, beta
%         referenceangle=3; %arc sec -- shooting for the substrate peak
%         x_scale = (theta - theta_B(1)) * rad2arcsec;
%         indices=find(x_scale>referenceangle);
%         idx_referenceangle = indices(1);
%         fprintf('\nReference angle used: %f arc sec\n', x_scale(idx_referenceangle));
%
%         for idx_i = 1:length(alpha)
%             for idx_j = 1:length(beta)
%                 phasecorrection = -1 *
phase(X_nij(N_laminae,idx_i,idx_j,idx_referenceangle));
%                 X_nij(N_laminae,idx_i,idx_j,:) = X_nij(N_laminae,idx_i,idx_j,:) *
exp(1i * phasecorrection);
%             end
%         end
%         %% perform double summation of scattering amplitude over all alpha, beta
%         %(1) compute the two distributions
%         W_alpha   = exp( -alpha.^2 / ( 2*sigma_alpha^2  )) / ( sqrt(2*pi)*sigma_alpha
);
%         W_epsilon = exp( -beta.^2  / ( 2*sigma_epsilon^2)) / ( sqrt(2*pi)*sigma_epsilon
);
%         %(2) start the summing variable off with zeros
%         doublesum = zeros(1,numPts);
```

```
%         %(3) perform the summing
%         for idx_i = 1:length(alpha) %sum over all i,j
%             for idx_j = 1:length(beta)
%                 xNij=squeeze(X_nij(N_laminae,idx_i,idx_j,:)).'; % Note: .' is the NON-
conjugate transpose
%                 doublesum = doublesum  +  (xNij * W_alpha(idx_i)*alpha_step *
W_epsilon(idx_j)*beta_step);
%                 %zz=zz+1; semilogy(x_scale,abs(X_n(n,:).').^2); title(['adding:
\alpha=' num2str(alpha(idx_i)/sigma_alpha) '\sigma_\alpha, \beta='
num2str(beta(idx_j)/sigma_epsilon) '\sigma_\epsilon']); ylim([1e-16 1e0]); F(zz) =
getframe(gcf);
%             end
%         end
%         X_n(N_laminae,:) = doublesum; %set the output of the scattering amplitude at
the top of the stack
%     end



    comp_time = toc(tStart); %stop stopwatch

    switch simulationChoice
        case 1 %dynamical
            tictoc_str = 'Takagi-Taupin';
        case 2 %kinematical
            tictoc_str = 'Bartels-Hornstra-Lobeek';
        case 3 %both
            tictoc_str = 'Takagi-Taupin and Bartels-Hornstra-Lobeek';
    end
    if comp_time < 60
        fprintf('The %.0f recursion(s) of the %s equation(s) took %.0f
seconds.\n',(N_laminae-1),tictoc_str,comp_time);
    else
        fprintf('The %.0f recursion(s) of the %s equation(s) took %.1f
minutes.\n',(N_laminae-1),tictoc_str,(comp_time/60));
    end
    %disp([' -- The ' num2str(N_laminae-1) ' recursion(s) of the ' tictoc_str '
equation(s) took ' num2str(comp_time) ' seconds.']);
end % if N_laminae > 1

delete(w);

%% (4.3) Intensity and Reflectivity

% %calculate the reflectivity P_H
% P_H_dyn = abs(F_H(N_laminae)/F_HBar(N_laminae)) * abs(X_n(N_laminae, :)) .^ 2;

%calculate the intensity as the magnitudes squared of the scattering amplitudes



%% perform double summation of intensity over all alpha

if (D(n)~=0)
    %(1) compute the two distributions
    W_alpha   = exp( -alpha.^2 / ( 2*sigma_alpha^2  )) / ( sqrt(2*pi)*sigma_alpha   );
    W_epsilon = exp( -beta.^2  / ( 2*sigma_epsilon^2)) / ( sqrt(2*pi)*sigma_epsilon );
    %(2) start the summing variable off with zeros
    doublesum = zeros(1,numPts);
    %(3) perform the summing
    w = waitbar(0,'Combining weighting rocking curve intensities...','CreateCancelBtn',
'setappdata(gcbf,''canceling'',1)');
    for idx_i = 1:length(alpha) %sum over all i,j
        for idx_j = 1:length(beta)
            xNij=squeeze(X_nij(N_laminae,idx_i,idx_j,:)).'; % Note: .' is the NON-
conjugate transpose
```

```
            doublesum = doublesum  +  (abs(xNij).^2 * W_alpha(idx_i)*alpha_step *
W_epsilon(idx_j)*beta_step);
            %zz=zz+1; semilogy(x_scale,abs(X_n(n,:).').^2); title(['adding: \alpha='
num2str(alpha(idx_i)/sigma_alpha) '\sigma_\alpha, \beta='
num2str(beta(idx_j)/sigma_epsilon) '\sigma_\epsilon']); ylim([1e-16 1e0]); F(zz) =
getframe(gcf);
        end
        waitbar((idx_i-1)/length(alpha),w,'Combining weighting rocking curve
intensities...','CreateCancelBtn', 'setappdata(gcbf,''canceling'',1)');
    end
    intensity_dyn = doublesum; %set the output of the scattering amplitude at the top of
the stack
    delete(w);
else
    intensity_dyn = abs(X_n(N_laminae, :)) .^ 2;
end



%% (5) ADD INSTRUMENTAL BROADENING
intensityBroadened_dyn = instr_broaden(intensity_dyn, numPtsPerArcSec, config);
% P_H_Broadened_dyn = instr_broaden(P_H_dyn, numPtsPerArcSec, config);
%-------end (5)-----------------------------------------------------



%% (6) PLOT RESULTS
x_scale = (theta - theta_B(1)) * rad2arcsec; %plot in arc sec

%if nargin == 0 %only produce plots if not run as a function

    %6a plot the scattering intensity
    titlestr = strrep([reflection(2:end-1) ' '
sourceWavelengths_str{radiationChoice}(1:findstr(sourceWavelengths_str{radiationChoice},'
(')-2)],'alpha','\alpha');
    titlestr = [titlestr(1:end-1) '_' titlestr(end)]; %insert the underscore for
subscript 1
    RockingCurve = rockingcurve_plot(x_scale, intensity_dyn, titlestr);
    xlim( [(x_scale(1)+2*broadening_width) (x_scale(end)-2*broadening_width)] );

    %add alpha and beta limit lines
    if exist('alpha_start','var')
        alpha_start_AS = (min(theta_B(2:length(theta_B)) + alpha1) - theta_B(1)) *
rad2arcsec;
        alpha_end_AS =   (max(theta_B(2:length(theta_B)) - alpha1) - theta_B(1)) *
rad2arcsec;
        beta_start_AS = (min(theta_B(2:length(theta_B)) + beta1) - theta_B(1)) *
rad2arcsec;
        beta_end_AS =   (max(theta_B(2:length(theta_B)) - beta1) - theta_B(1)) *
rad2arcsec;

%         alpha_start_AS = (min(alpha_start)-theta_B(1)) * rad2arcsec;
%         beta_start_AS  = (max(beta_start)-theta_B(1))  * rad2arcsec;
%         alpha_end_AS = (min(alpha_end)-theta_B(1)) * rad2arcsec;
%         beta_end_AS  = (max(beta_end)-theta_B(1))  * rad2arcsec;
        line( [alpha_start_AS alpha_start_AS], [1e-8 1e-5], 'Color','r' );
        line( [beta_start_AS  beta_start_AS],  [1e-8 1e-5], 'Color','g' );
        line( [alpha_end_AS alpha_end_AS], [1e-8 1e-5], 'Color','r' );
        line( [beta_end_AS  beta_end_AS],  [1e-8 1e-5], 'Color','g' );
    end

    %rockingcurve_plot(x_scale, intensity_dyn, intensity_kin, 'XRD Rocking Curve -
Scattering Amplitude. ');
    if (length(intensityBroadened_dyn) ~= 1) %user did select a broadening fn, so plot it
        %RockingCurveBroadened = rockingcurve_plot(x_scale, intensityBroadened_dyn, 'XRD
Rocking Curve - Scattering Amplitude (w/ broadening function). ');
        RockingCurveBroadened = rockingcurve_plot(x_scale, intensityBroadened_dyn,
[titlestr ' (broadened)']);
        if exist('alpha_start','var')
```

```
            line( [alpha_start_AS alpha_start_AS], [1e-8 1e-5], 'Color','r' );
            line( [beta_start_AS  beta_start_AS],  [1e-8 1e-5], 'Color','g' );
            line( [alpha_end_AS alpha_end_AS], [1e-8 1e-5], 'Color','r' );
            line( [beta_end_AS  beta_end_AS],  [1e-8 1e-5], 'Color','g' );
        end

        xlim( [(x_scale(1)+2*broadening_width) (x_scale(end)-2*broadening_width)] );

    end



    %6b plot the reflectivity
    %rockingcurve_plot(x_scale, P_H_dyn, P_H_kin, 'XRD Rocking Curve - Reflectivity (no
broadening function)');
    %if ((length(P_H_Broadened_dyn) ~= 1) || (length(P_H_Broadened_kin) == -1)) %user did
select a broadening fn, so plot it
    %    rockingcurve_plot(x_scale, P_H_Broadened_dyn, P_H_Broadened_kin, 'XRD Rocking
Curve - Reflectivity (w/ broadening function)');
    %end
%end
%----end (6)-------------------------------------------------



%% (7) DATA OUTPUT AND LAYER SUMMARIES
%% (7.1) layer summary
display_layersummary = config('display_layersummary');
if display_layersummary == -inf
    display_layersummary = menu('Display layer summary in the command window?', 'yes',
'no');
end

summary_matrix = {'Lamina ' 'Layer Type' 'Material' 'Thickness(μm)' 'Relaxed lattice
constant a0 (Angstroms)' 'Structure Factor F_H'}; %build the header, then append this
with rows of data in the loop below.

for n = 1:N_laminae
    if numel(findstr(layer_material{n}, '(x)')) == 1 %check if the layer is a ternary, in
which case need to report composition too
        material_str = strrep(layer_material{n},'(x)',strcat('(',num2str(x(n)),')'));
        material_str = strrep(material_str,'(1-x)',strcat('(',num2str(1-x(n)),')'));
    else
        material_str = layer_material{n};
    end
    summary_str{n} = ['LAMINA ' num2str(n-1) ': Type: ' layer_type{n} '. Material: '
material_str '. Thickness: ' num2str(layer_thickness(n)/10000) ' microns. Relaxed lattice
constant a0: ' num2str(a0{n}) 'A. Structure Factor F_H: ' num2str(F_H(n)) '.'];
    layer_data_str = {num2str(n-1) layer_type{n} layer_material{n}
num2str(layer_thickness(n)/10000) num2str(a0{n}) num2str(F_H(n))};
    summary_matrix = [summary_matrix; layer_data_str];

    if display_layersummary == 1
        disp(summary_str{n});
    end
end

%% (7.2) lattice profile
display_latticeprofile = config('display_latticeprofile');
if display_latticeprofile == -inf
    if strainchoice == 1 %user selected completely relaxed structure, so only ask if they
want to see profile or not
        display_latticeprofile = menu('Display lattice constant profile?', 'yes', 'no');
        if display_latticeprofile == 2  %if they selected no...
            display_latticeprofile = 6; %...then set the variable to 6, the value used
herein for 'no'.
        end
    else %user selected some sort of strain so give all profile options
```

```
        display_latticeprofile = menu('Which lattice constant profile(s) would you like
to view?', 'relaxed lattice constant (a_0)', 'in-plane (a)', 'out-of-plane (c)', 'both
(a) and (c) in same plot', 'both (a) and (c) in different plots', '(none)');
    end
end

if display_latticeprofile ~= 6 %user wants lattice constant profile(s)

    %create vectors for lattice constants a0, a, c, and distance from interface
    dist_vec(1) = -1.5*10^4;
    a0_vec(1) = a0{1};
    a_vec(1) = a(1);
    c_vec(1) = c(1);
    dist_vec(2) = 0;
    a0_vec(2) = a0{1};
    a_vec(2) = a(1);
    c_vec(2) = c(1);

    idx = 3;
    for k = 2:length(a0)
        dist_vec(idx) = dist_vec(idx-1) + 0.00001;
        a0_vec(idx) = a0{k};
        a_vec(idx) = a(k);
        c_vec(idx) = c(k);
        idx = idx + 1;
        dist_vec(idx) = dist_vec(idx-2) + layer_thickness(k);
        a0_vec(idx) = a0{k};
        a_vec(idx) = a(k);
        c_vec(idx) = c(k);
        idx = idx + 1;
    end

    if display_latticeprofile == 1 %user wants relaxed lattice constant profile
        figure; %open a new figure window
        plot(dist_vec/10^4, a0_vec);
        xlabel('distance from interface (\mum)');
        ylabel('relaxed lattice constant a_0 (in angstroms)');
        title('Relaxed Lattice Constant Profile');
    end
    if (display_latticeprofile == 2 || display_latticeprofile == 5) %user wants in-plane
lattice constant profile
        figure; %open a new figure window
        plot(dist_vec/10^4, a_vec);
        xlabel('distance from interface (\mum)');
        ylabel('in-plane lattice constant a (in angstroms)');
        title('In-Plane Lattice Constant Profile');
    end
    if (display_latticeprofile == 3 || display_latticeprofile == 5) %user wants out-of-
plane lattice constant profile
        figure; %open a new figure window
        plot(dist_vec/10^4, c_vec);
        xlabel('distance from interface (\mum)');
        ylabel('out-of-plane lattice constant c (in angstroms)');
        title('Out-of-Plane Lattice Constant Profile');
    end
    if display_latticeprofile == 4 %user wants both in- and out- of-plane lattice
constant profiles in same plot
        figure; %open a new figure window
        plot(dist_vec/10^4, a_vec, dist_vec/10^4, c_vec)
        xlabel('distance from interface (\mum)');
        ylabel(strcat('lattice constant (', char(197), ')'));
        title('In- and Out- of-Plane Lattice Constant Profile');
        legend('in-plane', 'out-of-plane', 'Location', 'Best');
    end
end

%% (7.3) strain profile
if strainchoice ~= 1 %check if user selected completely relaxed structure; if not,
proceed by asking which type of strain profile they want to view.
```

```
    disp_strainprofile = config('display_strainprofile');
    if disp_strainprofile == -inf
        disp_strainprofile = menu('Which strain profile(s) would you like to view?', 'in-
plane strain', 'out-of-plane strain', 'in- and out- of-plane strains in same plot', 'in-
and out- of-plane strains in different plots', '(none)');
    end

    if disp_strainprofile ~= 5 %user wants strain profile(s)

        %create vectors for elastic constants and distance from interface
        dist_vec(1) = -1.5*10^4;
        epsilon_par_vec(1) = epsilon_par(1);
        epsilon_perp_vec(1) = epsilon_perp(1);
        dist_vec(2) = 0;
        epsilon_par_vec(2) = epsilon_par(1);
        epsilon_perp_vec(2) = epsilon_perp(1);

        idx = 3;
        for k = 2:length(epsilon_par)
            dist_vec(idx) = dist_vec(idx-1) + 0.00001;
            epsilon_par_vec(idx) = epsilon_par(k);
            epsilon_perp_vec(idx) = epsilon_perp(k);
            idx = idx + 1;
            dist_vec(idx) = dist_vec(idx-2) + layer_thickness(k);
            epsilon_par_vec(idx) = epsilon_par(k);
            epsilon_perp_vec(idx) = epsilon_perp(k);
            idx = idx + 1;
        end

        if (disp_strainprofile == 1 || disp_strainprofile == 4) %user wants in-plane
strain profile
            figure; %open a new figure window
            plot(dist_vec/10^4, epsilon_par_vec);
            xlabel('distance from interface (\mum)');
            ylabel('in-plane strain \epsilon_|_| (dimensionless)');
            title('In-Plane Strain Profile');
        end
        if (disp_strainprofile == 2 || disp_strainprofile == 4) %user wants out-of-plane
strain profile
            figure; %open a new figure window
            plot(dist_vec/10^4, epsilon_perp_vec);
            xlabel('distance from interface (\mum)');
            ylabel('out-of-plane strain \epsilon_\perp (dimensionless)');
            title('Out-of-Plane Strain Profile');
        end
        if disp_strainprofile == 3 %user wants both in- and out- of-plane strain profiles
in same plot
            figure; %open a new figure window
            plot(dist_vec/10^4, epsilon_par_vec, dist_vec/10^4, epsilon_perp_vec)
            xlabel('distance from interface (\mum)');
            ylabel('strain (dimensionless)');
            title('In- and Out- of-Plane Strain Profile');
            legend('in-plane', 'out-of-plane', 'Location', 'Best');
        end
    end
end

%% (7.4) put data on clipboard
if config('PutDataOnClipboard')

    %stuff = {'x scale' x_scale; 'intensity_dyn' intensity_dyn}'; %tried to add header
4/18
    if length(intensityBroadened_dyn) == 1
        stuff = [x_scale;intensity_dyn]';
    else
        stuff = [x_scale;intensity_dyn;intensityBroadened_dyn]';
    end
```

```
    num2clip(stuff);
    fprintf('XRD data copied to clipboard.\n');
end

%% (7.5) write data to Excel
filename = config('filename');
if (filename == -inf)
    filename = char(inputdlg(['Write Data to Excel: Enter filename.xls or nothing if no
XLS desired.']));
elseif (filename == inf)
    filename = []; % make it empty if config file has inf, this means no XLS file, and no
question for XLS file
end
if (isempty(filename) == 0) %user wants to write an XLS
    success = 1;

    layerSummaryTitle = cell(5,6);
    layerSummaryTitle(1) = {'X-RAY DIFFRACTION SIMULATION SUMMARY'};
    layerSummaryTitle(2) = {strcat('Reflection:',reflection)};
    layerSummaryTitle(3) = {strcat('Radiation:',sourceWavelengths_str{radiationChoice})};

    layerSummary = [layerSummaryTitle;summary_matrix]; % add a header to the cell array
    success = xlswrite(filename, layerSummary, 'Layer Summary') * success;

    dataArray_str = num2cell([x_scale;intensity_dyn]');



    dataArrayWithHeader_str = {layerSummary;{['Theta-ThetaB(arcsec)']
['Intensity(a.u.)']};dataArray_str};
    success = xlswrite(filename, dataArrayWithHeader_str, 'XRD Data') * success;

    if intensityBroadened ~= -1 %user did select a broadening fn, so stick this in XLS
file too
        dataArray_Broadened_str = num2cell([x_scale;intensityBroadened]'); % format the
table properly for excel
        dataArrayWithHeader_Broadened_str = [{['Theta-ThetaB(arcsec)'] ['Broadened
Intensity (a.u.)']};dataArray_str];
        success = xlswrite(filename, dataArrayWithHeader_Broadened_str, 'Broadened XRD
Data') * success;
    end

    if success == 1
        msgbox('Excel file successfully created.')
    else
        msgbox('ERROR: Excel file creation failed.')
    end
end

%% (7.6) report the rocking curve width across the whole thing.
rockingCurveWidth_Dynamical =
myfwhm(x_scale,intensity_dyn,x_scale(1),x_scale(numel(x_scale)));
fprintf('rockingCurveWidth_Dynamical = %.2f"\n',rockingCurveWidth_Dynamical);
if length(intensityBroadened_dyn) > 1
    rockingCurveWidth_Dynamical_Broadened =
myfwhm(x_scale,intensityBroadened_dyn,x_scale(1),x_scale(numel(x_scale)));
    fprintf('rockingCurveWidth_Dynamical_Broadened =
%.2f"\n',rockingCurveWidth_Dynamical_Broadened);
end

fprintf('XRD Simulation complete.\n');

%% (7.7) return the XRD data (function)
if nargin > 0 %only bother to do this if function was called with input config file
```

```
    if length(intensityBroadened_dyn) == 1
        header = [{'Theta-ThetaB(")'} {'IntensityDyn'}];
        XRDdata = [header; num2cell(x_scale') num2cell(intensity_dyn')];
    else
        header = [{'Theta-ThetaB(")'} {'IntensityDyn'} {'IntensityDynBroadened'}];
        XRDdata = [header; num2cell(x_scale') num2cell(intensity_dyn')
num2cell(intensityBroadened_dyn')];
    end

end
if ~exist('RockingCurveBroadened', 'var')
    RockingCurveBroadened = [];
end
```

26) xrd_sim_PIDDM.m

```
%% XRD_SIM_PIDDM
% Simulate x-ray diffraction rocking curves using phase-invariant model
%
%   An ongoing program to simulate x-ray diffraction rocking curves for
%   arbitraty heterostructures. Used for research with Dr. John Ayers.
%
%   Paul Rago
%   Department of Electrical & Computer Engineering
%   University of Connecticut
%   (c) 2008 - 2014
%


%% Changelog
%
%       2012-06-22  - Fixed bug in eta: sign of delta_phi_tet and delta_phi_tilt wrong;
delta_phi_tet was not present in D>0 eta.
%       2012-06-18  - Added asymmetric dislocation densities, azimuthal variation, and
epilayer tilt
%                   - Added vargin, property/value pair feature. So far only azimuth
implemented.
%       2011-05-05  - Fixed diagnostic alpha and beta limit lines
%       2011-04-27  - Enhancements of when to display ETC (if LG layer, don't show,
unless TDDinitial=TDDfinal)
%       2011-04-19  - Changed clear to only clear select variables
%                   - added multisim.m, which simulates numerous config files
programatically
%                   - changed xrd_sim to be a function instead of a script, with argin of
config file name
%       2011-04-18  - Set a minimum of 21 for NBin_alpha, NBin_beta.
%                   - Updated makeXvsLayerPlot.m to ask for rocking angle at which to
show plots.
%                   - Added feature to copy XRD data to clipboard. In config.m it's
called 'PutDataOnClipboard'.
%       2011-04-11  - Added makeEtaPlots.m
%                   - Added makeXvsLayerPlot.m
%                   - Modified rockingcurve_plot.m to plot logarithmically.
%                   - Made fixed numPtsPerArcSec/numBinsPerArcSec.
%       2011-03-30  - Finished adding Lorentzian distribution for P_alpha and P_epsilon;
added linear profile for TDD; added fancy Nbin
%                     calculation (instead of fixed); added estimated time till
completion (ETC); cleaned up command window output.
%       2011-03-23  - Added rectangular and triangular distributions for P_alpha and
P_epsilon
%                   - Added 'dislocation_distribution_type' choice to the config file.
%                   - Started implementing Lorentzian distribution.
%       2011-03-16  - Fixed bug in instr_broaden.m where the broadening function's width
was off by a factor of 1/numPtsPerArcsec
%       2011-03-02  - Modified the TDD calculation to use reciprocol of sum of
reciprocols.
```

```
%       2011-02-25  - Finished adding the threading dislocation density (TDD) calculation
using Gaussians for alpha and beta.
%       2011-02-24  - Finished adding Al(x)Ga(1-x)As to uniform, LG, and SL layers
(elastic constants)
%       2011-02-23  - Began adding threading dislocation density stuff.
%       2011-02-18  - Added Al(x)Ga(1-x)As to uniform, LG, and SL layers
%                   - Added reflection (P_H) calculation.
%       2011-02-10  - Fixed an error with T(n), the polarization factor C(n) is now a
factor in T(n).
%                   - Removed normalization in the rocking curve plot
%       2011-02-09  - Added kinematical calculation.
%                     ==> STILL NEED TO CLEAN UP THE XLS OUTFILE PART -- currently non-
operational
%       2010-04-28  - Added "percent residual strain" custom strain option
%                     for linearly graded layers
%       2010-03-21  - Changed plotting routine to produce intensity
%                     normalized to the maximum value in the produced
%                     plot.
%       2010-03-02  - Added a configuration file, config.m, which allows a
%                     user to pre-specify the answer to nearly every
%                     question about how the simulation should be run.
%       2010-02-28  - Added AgKalpha1 x-ray source wavelength
%       2010-02-17  - Enhanced the "write to XLS feature". Added title and layer
%                     summary, put XRD data in separate worksheet, and output
%                     instrumentally broadened XRD data too.
%                   - Added Hg(x)Cd(1-x)Te and In(x)Ga(1-x)Sb to superlattice layers.
%                   - Added Cr,Fe,Mo,Ag radiations back in, now including all the real
%                     and imaginary anomalous dispersion corrections for
%                     ASFs. Still need AgKa wavelength!
%       2010-02-15  - Finished adding Hg(x)Cd(1-x)Te and In(x)Ga(1-x)Sb to uniform
%                     and LG layers (with elastic constants).
%       2010-02-13  - Added InSb, CdTe, GaSb substrates.
%                   - Added several reflections and streamlined the
%                     process of adding reflections
%                   - Removed all source radiation other than Cu kalpha1,
%                     because noticed that in Int'l. Table for X-Ray
%                     Crystallography, I had only used dispersion
%                     corrections for Cu kalpha.
%                   - Added "write to XLS" feature
%
%  ............ /Changelog
.................................................................


function [XRDdata, RockingCurve, RockingCurveBroadened] = xrd_sim(configfilename,
Property, Value)
% NOTES RE: INPUT VARIABLES
% (1) It is not necessary to specify any of them. If you don't, config.m will be assumed.
% (2) Property is a string that overrides that specified (or not) in the config file.
% (3) Value is the value corresponding to Property, and together these override that
specified (or not specified)
%     in the config file.

if ~exist('Property','var') %if nothing specified for a parameter..
    Property = ''; %..at least make the variable, so as not to throw errors later
else
    eval([Property ' = ' num2str(Value) ';']);
end



fprintf('\nBeginning XRD Simulation...\n');

% if user specified a configfilename argument to xrd_sim, redefine config to
% be a function handle to the function specified by the string configfilename
if nargin > 0
   config = str2func(configfilename);
else
    config = str2func('config');
```

```
end

%% conversion factors
deg2rad = pi / 180; %multiply by this to convert degrees to radians.
rad2deg = 1 / deg2rad; %multiply by this to convert radians to degrees.
deg2arcsec = 3600; %multiply by this to convert deg. to arc seconds.
arcsec2deg = 1 / deg2arcsec; %multiply by this to convert arcseconds to degrees.
rad2arcsec = rad2deg * deg2arcsec; %multiply by this to convert radians to arc seconds.
arcsec2rad = 1 / rad2arcsec; %multiply by this to convert arcseconds to radians.



%% constants
r_e = 2.8179E-5; %classical electron radius (in angstroms)

%Note: to add to the x-ray source wavelengths list, first add to sourceWavelengths_str in
the same syntax, then add anomolous dispersion corrections associated with that radiation
in asf_coefficients.m.
sourceWavelengths_str = {'Cr kalpha1 (2.28970 A)', 'Fe kalpha1 (1.936042 A)', 'Cu kalpha1
(1.540594 A)', 'Mo kalpha1 (0.70930 A)', 'Ag kalpha1 (0.5594075 A)'};

%Note: to add reflections, just add to reflection_str using the same syntax.
reflection_str = {'(002)', '(111)', '(113)', '(004)', '(224)', '(115)', '(044)', '(135)',
'(006)', '(026)', '(335)', '(444)', '(117)'};

%Note: to add substrate orientations, just add to substrateOrientation_str using the same
syntax.
substrateOrientation_str = {'(001)', '(111)'};




%% (1) MATERIALS AND LATTICE PARAMETER DECLARATIONS

layerTypes_str = {'uniform', 'linearly graded', 'superlattice', '(no more layers)'};

%substrate
%           Note:   to add a substrate, you must add (a) the name and (b) lattice
constant here,
%           but also add (c) the asf_coefficients assignment in section (2) below, and
(d) add
%           the ASFs to asf_coefficients.m if any of the atoms are not already there.

substrates_str =           { 'GaAs',  'Si',    'InP',   'InSb',  'CdTe',  'GaSb' };
substrates_latticeParams = { 5.65340, 5.43108, 5.86900, 6.47940, 6.48100, 6.0960 }; %(in
angstroms)

%           Note: to add uniform, LG, or SL layers, add the lattice constants here, and
make an entry with the elastic constants in section (2) below

%uniform layers
uniformLayers_str =           {'ZnSe', 'ZnS', 'ZnS(x)Se(1-x)',  'In(x)Ga(1-x)As', 'Si(1-
x)Ge(x)',   'Hg(x)Cd(1-x)Te', 'In(x)Ga(1-x)Sb', 'Al(x)Ga(1-x)As' };
uniformLayers_latticeParams = {5.6687, 5.4105 [-0.2582 5.6687], [0.405 5.6534],   [0.2265
5.43108], [-0.02 6.481],    [0.3834 6.096],   [0.0066 5.6534]  };  % [a b c]  <==>  ax^2
+ bx + c    ;    [b c] <==> bx+c

%linearly graded layers
LGlayers_str =           { 'ZnS(x)Se(1-x)',  'In(x)Ga(1-x)As', 'Si(1-x)Ge(x)',
'Hg(x)Cd(1-x)Te', 'In(x)Ga(1-x)Sb', 'Al(x)Ga(1-x)As'  };
LGlayers_latticeParams = { [-0.2582 5.6687], [0.405 5.6534],   [0.2265 5.43108], [-0.02
6.481],    [0.3834 6.096],   [0.0066 5.6534]   }; % [a b c]  <==>  ax^2 + bx + c

%superlattice layers
```

```
SLlayers_str =           { 'ZnS(x)Se(1-x)',  'In(x)Ga(1-x)As', 'Si(1-x)Ge(x)',
'Hg(x)Cd(1-x)Te', 'In(x)Ga(1-x)Sb', 'Al(x)Ga(1-x)As'  };
SLlayers_latticeParams = { [-0.2582 5.6687], [0.405 5.6534],   [0.2265 5.43108], [-0.02
6.481],    [0.3834 6.096],   [0.0066 5.6534]   }; % [a b c]  <==>  ax^2 + bx + c

%--end (1) materials and lattice param. declarations----------------------



%% (2) USER INPUT

%obtain strain information
strainchoice = config('strain');
if strainchoice == -inf
    strainchoice = menu('Enter the type of strain you would like to use in the
structure:', 'Completely Relaxed (none)', 'Pseudomorphic / Coherently Strained', 'Custom:
constant strain', 'Custom: constant % residual strain (LG layer only)');
end

%obtain choice of substrate
substrateLayer_choice = config('substrate');
if substrateLayer_choice == -inf
    substrateLayer_choice = menu('Choose a substrate (layer 0):', substrates_str);
end %substrateLayer_choice == -inf
switch substrateLayer_choice
    case 1 % GaAs substrate selected
        coeffs_A{1} = asf_coefficients('Ga'); %atom A is Ga
        coeffs_B{1} = asf_coefficients('As'); %atom B is As
    case 2 % Si substrate selected
        coeffs_A{1} = asf_coefficients('Si'); %atom A is Si
        coeffs_B{1} = coeffs_A{1};            %atom B is also Si
    case 3 % InP substrate selected
        coeffs_A{1} = asf_coefficients('In'); %atom A is In
        coeffs_B{1} = asf_coefficients('P');  %atom B is P
    case 4 % InSb substrate selected
        coeffs_A{1} = asf_coefficients('In'); %atom A is In
        coeffs_B{1} = asf_coefficients('Sb'); %atom B is Sb
    case 5 % CdTe substrate selected
        coeffs_A{1} = asf_coefficients('Cd'); %atom A is Cd
        coeffs_B{1} = asf_coefficients('Te'); %atom B is Te
    case 6 % GaSb substrate selected
        coeffs_A{1} = asf_coefficients('Ga'); %atom A is Ga
        coeffs_B{1} = asf_coefficients('Sb'); %atom B is Sb
    otherwise
        msgbox('You must choose a substrate. Please try again.','Invalid Input',
'error');
        error('Error: you must choose a substrate. Please try again.');
end %switch substrateLayer_choice


layer_type{1} = 'Substrate';
layer_material{1} = substrates_str{substrateLayer_choice};
layer_thickness(1) = Inf;
a0{1} = substrates_latticeParams{substrateLayer_choice}; %lattice parameter of the
substrate -- index {1} corresponds to layer zero
a(1) = a0{1};
c(1) = a0{1};
D(1) = 0;
n_layers = 1;
n_laminae = 1;


%Substrate Orientation
substrateOrientation = config('substrateOrientation');
if substrateOrientation == -inf
    substrateOrientation_choice = menu(['Choose the crystal orientation of the '
layer_material{1} ' substrate:'],substrateOrientation_str);
    substrateOrientation = char(substrateOrientation_str(substrateOrientation_choice));
end
u1 = str2double(substrateOrientation(2)); v1 = str2double(substrateOrientation(3)); w1 =
str2double(substrateOrientation(4));
```

```
% /Substrate Orientation

while 1 == 1 %loop until user doesn't want any more layers

    layerType_choice = config(strcat('layer_',num2str(n_layers),'_type'));
    if layerType_choice == -inf
        if exist('configSpecifiedStructure','var') == 0
            layerType_choice = menu(['Choose the type of layer ' num2str(n_layers) ':'],
layerTypes_str);
        else
            uniformLayer_choice = 4; %no more layers!
        end
    else
        configSpecifiedStructure = true;
    end

    switch layerType_choice
        case 1 %Uniform layer selected



            uniformLayer_choice = config(strcat('layer_',num2str(n_layers),'_material'));
            if uniformLayer_choice == -inf
                uniformLayer_choice = menu(['Choose material for layer '
num2str(n_layers) ':'], uniformLayers_str);
            end

            %question user about thickness of layer
            layer_thickness(n_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_thickness'))*10000;
            if layer_thickness(n_laminae + 1) == -inf
                layer_thickness(n_laminae + 1) = str2double(inputdlg(['Enter thickness
for the uniform ' uniformLayers_str{uniformLayer_choice} ' layer (layer '
num2str(n_layers) ') in microns:']))*10000;
            end

            if strainchoice == 3 %custom strain
                epsilon_par(n_laminae +
1)=config(strcat('layer_',num2str(n_layers),'_inplanestrain'));
                if epsilon_par(n_laminae + 1) == -inf
                    epsilon_par(n_laminae + 1) = str2double(inputdlg(['Enter the in-plane
strain for layer ' num2str(n_layers)]));
                end
            end

            %layer tilt
            epilayer_tilt(n_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_tilt'));
            if epilayer_tilt(n_laminae + 1) == -inf
                epilayer_tilt(n_laminae + 1) = str2double(inputdlg(['Enter the the tilt
(in arc sec) for layer ' num2str(n_layers)]));
            end

            switch uniformLayer_choice
                %TERNARY LAYER NOTE: when coeffs_C exists, coeffs_A has no subscript,
coeffs_B has 'x' subscript, coeffs_C has '1-x' subscript.
                case 1 % ZnSe
                    coeffs_A{n_laminae + 1} = asf_coefficients('Zn');
                    coeffs_B{n_laminae + 1} = asf_coefficients('Se');
                    C11_constants{n_laminae + 1} = [87.2 87.2]; %elastic constants
                    C12_constants{n_laminae + 1} = [52.4 52.4]; %elastic constants
                    x(n_laminae + 1) = 1;
                case 2 % ZnS
                    coeffs_A{n_laminae + 1} = asf_coefficients('Zn');
                    coeffs_B{n_laminae + 1} = asf_coefficients('S');
                    C11_constants{n_laminae + 1} = [104.62 104.62]; %elastic constants
                    C12_constants{n_laminae + 1} = [65.33 65.33]; %elastic constants
                    x(n_laminae + 1) = 1;
```

```
                case 3 % ZnS(x)Se(1-x)
                    coeffs_A{n_laminae + 1} = asf_coefficients('Zn');
                    coeffs_B{n_laminae + 1} = asf_coefficients('S');
                    coeffs_C{n_laminae + 1} = asf_coefficients('Se');
                    C11_constants{n_laminae + 1} = [104.62 87.2]; %elastic constants
                    C12_constants{n_laminae + 1} = [65.33 52.4]; %elastic constants
                case 4 %In(x)Ga(1-x)As
                    coeffs_A{n_laminae + 1} = asf_coefficients('As');
                    coeffs_B{n_laminae + 1} = asf_coefficients('In');
                    coeffs_C{n_laminae + 1} = asf_coefficients('Ga');
                    C11_constants{n_laminae + 1} = [83.29 118.4]; %elastic constants
                    C12_constants{n_laminae + 1} = [45.26 53.7]; %elastic constants
                case 5 % Si(1-x)Ge(x)
                    coeffs_A{n_laminae + 1} = [];
                    coeffs_B{n_laminae + 1} = asf_coefficients('Ge');
                    coeffs_C{n_laminae + 1} = asf_coefficients('Si');
                    C11_constants{n_laminae + 1} = [124.0 160.1]; %elastic constants
                    C12_constants{n_laminae + 1} = [41.3 57.8]; %elastic constants
                case 6 % Hg(x)Cd(1-x)Te selected
                    coeffs_A{n_laminae + 1} = asf_coefficients('Te');
                    coeffs_B{n_laminae + 1} = asf_coefficients('Hg');
                    coeffs_C{n_laminae + 1} = asf_coefficients('Cd');
                    C11_constants{n_laminae + 1} = [53.61 53.3]; % [C11(HgTe) C11(CdTe)]
                    C12_constants{n_laminae + 1} = [36.60 36.5]; % [C12(HgTe) C12(CdTe)]
                case 7 % In(x)Ga(1-x)Sb selected
                    coeffs_A{n_laminae + 1} = asf_coefficients('Sb');
                    coeffs_B{n_laminae + 1} = asf_coefficients('In');
                    coeffs_C{n_laminae + 1} = asf_coefficients('Ga');
                    C11_constants{n_laminae + 1} = [65.92 88.50]; % [C11(InSb) C11(GaSb)]
                    C12_constants{n_laminae + 1} = [35.63 40.40]; % [C12(InSb) C12(GaSb)]
                case 8 % Al(x)Ga(1-x)As selected
                    coeffs_A{n_laminae + 1} = asf_coefficients('As');
                    coeffs_B{n_laminae + 1} = asf_coefficients('Al');
                    coeffs_C{n_laminae + 1} = asf_coefficients('Ga');
                    C11_constants{n_laminae + 1} = [125 118.4]; % [C11(AlAs) C11(GaAs)]
                    C12_constants{n_laminae + 1} = [53.4 53.7]; % [C12(AlAs) C12(GaAs)]
                otherwise
                    msgbox('You failed to make a selection. Please try again.','Invalid
Input', 'error');
                    n_layers = n_layers - 1;
                    n_laminae = n_laminae - 1;
            end %switch uniformLayer_choice

            layer_material{n_laminae + 1} = uniformLayers_str{uniformLayer_choice};

            if numel(findstr(layer_material{n_laminae + 1}, '(x)')) == 1 %check if the
layer is a ternary
                x(n_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_composition'));
                if x(n_laminae + 1) == -inf
                    x(n_laminae + 1) = str2double(inputdlg(['Enter composition of layer '
num2str(n_layers) ':']));
                end
                %x(n_laminae + 1) = obtainComposition(n_layers); %old way of getting
composition, doesn't work properly since relays on config.m
            end

            a0{n_laminae + 1} = uniformLayers_latticeParams{uniformLayer_choice};

        case 2 %Linearly graded layer selected

            LGlayer_choice = config(strcat('layer_',num2str(n_layers),'_material'));
            if LGlayer_choice == -inf
                LGlayer_choice = menu(['Choose material for the '
layerTypes_str{layerType_choice} ' layer (layer ' num2str(n_layers) '):'], LGlayers_str);
            end

            %question user about thickness of layer
```

```
            layer_thickness(n_laminae + 1) =
config(strcat('layer_',num2str(n_layers),'_thickness'))*10000;
            if layer_thickness(n_laminae + 1) == -inf
                layer_thickness(n_laminae + 1) = str2double(inputdlg(['Enter thickness
for the linearly graded ' LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers)
') in microns:']))*10000;
            end


            n_lam = config(strcat('layer_',num2str(n_layers),'_numberOfLamina'));
            if n_lam == -inf
                n_lam = str2double(inputdlg(['Enter the desired number of laminae for the
' LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers) '):']));
            end


            x_start = config(strcat('layer_',num2str(n_layers),'_startingComposition'));
            if x_start == -inf
                x_start = str2double(inputdlg(['Enter starting composition for the '
LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers) '):']));
            end
            x_end = config(strcat('layer_',num2str(n_layers),'_endingComposition'));
            if x_end == -inf
                x_end = str2double(inputdlg(['Enter ending composition for the '
LGlayers_str{LGlayer_choice} ' layer (layer ' num2str(n_layers) '):']));
            end


            if strainchoice == 3 %custom strain
                inplanestrain = str2double(inputdlg('Enter the in-plane strain for all
the laminae in the graded layer:'));
                epsilon_par((n_laminae + 1):(n_laminae + n_lam)) = inplanestrain; %set in
plane strain for all laminae in the GL
            end
            if strainchoice == 4 %custom percent residual strain
                percentResidualStrain = str2double(inputdlg('Enter the percent residual
strain for the layer:'));
                fractionalStrain((n_laminae + 1):(n_laminae + n_lam)) =
(percentResidualStrain / 100);
            end


            layer_thickness((n_laminae + 1):(n_laminae + n_lam)) =
layer_thickness(n_laminae+1) / n_lam; %set thickness of all laminae in the GL equal to
(total thickness)/(number of laminae)


            for lam_counter = n_laminae:(n_laminae + n_lam - 1)
                switch LGlayer_choice
                    %TERNARY LAYER NOTE: when coeffs_C exists, coeffs_A has no subscript,
coeffs_B has 'x' subscript, coeffs_C has '1-x' subscript.
                    case 1 % ZnS(x)Se(1-x)
                        coeffs_A{lam_counter + 1} = asf_coefficients('Zn');
                        coeffs_B{lam_counter + 1} = asf_coefficients('S');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Se');
                        %C11 constants is the two-element vector: [ C11(ZnS)  C11(ZnSe) ]
for later calculation of x*C11(ZnS) + (1-x)*C11(ZnSe).
                        C11_constants{lam_counter + 1} = [104.62 87.2];%[ C11(ZnS)
C11(ZnSe) ]
                        C12_constants{lam_counter + 1} = [65.33 52.4]; %[ C12(ZnS)
C12(ZnSe) ]
                    case 2 % In(x)Ga(1-x)As
                        coeffs_A{lam_counter + 1} = asf_coefficients('As');
                        coeffs_B{lam_counter + 1} = asf_coefficients('In');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Ga');
                        C11_constants{lam_counter + 1} = [83.29 118.4]; %[ C11(InAs)
C11(GaAs) ]
                        C12_constants{lam_counter + 1} = [45.26 53.7];  %[ C12(InAs)
C12(GaAs) ]
                    case 3 % Si(1-x)Ge(x)
                        coeffs_A{lam_counter + 1} = [];
                        coeffs_B{lam_counter + 1} = asf_coefficients('Ge');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Si');
```

```
                        C11_constants{lam_counter + 1} = [124.0 160.1]; %[ C11(Ge)
C11(Si) ]
                        C12_constants{lam_counter + 1} = [41.3 57.8];   %[ C12(Ge)
C12(Si) ]
                    case 4 % Hg(x)Cd(1-x)Te
                        coeffs_A{lam_counter + 1} = asf_coefficients('Te');
                        coeffs_B{lam_counter + 1} = asf_coefficients('Hg');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Cd');
                        C11_constants{lam_counter + 1} = [53.61 53.3]; % [C11(HgTe)
C11(CdTe)]
                        C12_constants{lam_counter + 1} = [36.60 36.5]; % [C12(HgTe)
C12(CdTe)]
                    case 5 % In(x)Ga(1-x)Sb
                        coeffs_A{lam_counter + 1} = asf_coefficients('Sb');
                        coeffs_B{lam_counter + 1} = asf_coefficients('In');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Ga');
                        C11_constants{lam_counter + 1} = [65.92 88.50]; % [C11(InSb)
C11(GaSb)]
                        C12_constants{lam_counter + 1} = [35.63 40.40]; % [C12(InSb)
C12(GaSb)]
                    case 6 % Al(x)Ga(1-x)As selected
                        coeffs_A{lam_counter + 1} = asf_coefficients('As');
                        coeffs_B{lam_counter + 1} = asf_coefficients('Al');
                        coeffs_C{lam_counter + 1} = asf_coefficients('Ga');
                        C11_constants{lam_counter + 1} = [125 118.4]; % [C11(AlAs)
C11(GaAs)]
                        C12_constants{lam_counter + 1} = [53.4 53.7]; % [C12(AlAs)
C12(GaAs)]
                end

                %Grade the composition x from x_start to x_end
                x(lam_counter + 1) = x_start + (lam_counter - n_laminae) * (x_end -
x_start) / (n_lam-1);

                %set strings describing the layer
                layer_type{lam_counter + 1} = layerTypes_str{layerType_choice};
                layer_material{lam_counter + 1} = LGlayers_str{LGlayer_choice};

                a0{lam_counter + 1} = LGlayers_latticeParams{LGlayer_choice}; %set
lattice parameter

                %layer tilt
                epilayer_tilt(lam_counter + 1) =
config(strcat('layer_',num2str(n_layers),'_tilt'));
                if epilayer_tilt(lam_counter + 1) == -inf
                    epilayer_tilt(lam_counter + 1) = str2double(inputdlg(['Enter the the
tilt (in arc sec) for layer ' num2str(n_layers)]));
                end

            end %for lam_counter = n_laminae:(n_laminae + n_lam - 1)

            n_laminae = n_laminae + n_lam - 1;

        case 3 %Superlattice layer selected

            %the materials, thicknesses, and compositions are all indexed
            %as (1) for atom A and (2) for atom B.
            SLlayer_choice(1) = config(strcat('layer_',num2str(n_layers),'_material_A'));
            if SLlayer_choice(1) == -inf
                SLlayer_choice(1) = menu(['Choose material for atom A in the superlattice
layer (layer ' num2str(n_layers) '):'], SLlayers_str);
            end
            SLlayer_choice(2) = config(strcat('layer_',num2str(n_layers),'_material_B'));
            if SLlayer_choice(2) == -inf
                SLlayer_choice(2) = menu(['Choose material for atom B in the superlattice
layer (layer ' num2str(n_layers) '):'], SLlayers_str);
            end
```

```
            SLthickness(1) = config(strcat('layer_',num2str(n_layers),'_thickness_A')) *
10;
            if SLthickness(1) == -inf
                SLthickness(1) = 10 * str2double(inputdlg(['Enter the thickness for
superlattice layer A in one period of the ' SLlayers_str{SLlayer_choice} ' layer (layer '
num2str(n_layers) ') in nanometers:']));
            end
            SLthickness(2) = config(strcat('layer_',num2str(n_layers),'_thickness_B')) *
10;
            if SLthickness(2) == -inf
                SLthickness(2) = 10 * str2double(inputdlg(['Enter the thickness for
superlattice layer B in one period of the ' SLlayers_str{SLlayer_choice} ' layer (layer '
num2str(n_layers) ') in nanometers:']));
            end

            SLx(1) = config(strcat('layer_',num2str(n_layers),'_composition_A'));
            if SLx(1) == -inf
                SLx(1) = str2double(inputdlg(['Enter the composition for atom A of the '
SLlayers_str{SLlayer_choice} ' superlattice layer (layer ' num2str(n_layers) '):']));
            end
            SLx(2) = config(strcat('layer_',num2str(n_layers),'_composition_B'));
            if SLx(2) == -inf
                SLx(2) = str2double(inputdlg(['Enter the composition for atom B of the '
SLlayers_str{SLlayer_choice} ' superlattice layer (layer ' num2str(n_layers) '):']));
            end

            num_periods = config(strcat('layer_',num2str(n_layers),'_numberOfPeriods'));
            if num_periods == -inf
                num_periods = str2double(inputdlg(['Enter the number of periods for the '
SLlayers_str{SLlayer_choice} ' superlattice layer (layer ' num2str(n_layers) '):']));
            end

            if strainchoice == 3 %custom strain
                inplanestrain =
config(strcat('layer_',num2str(n_layers),'_inplanestrain'));
                if inplanestrain == -inf
                    inplanestrain = str2double(inputdlg('Enter the in-plane strain for
the superlattice layer:'));
                end
            end

            for period_counter = n_laminae:1:(n_laminae + 2 * num_periods - 1)

                atomSelector = mod(period_counter-n_laminae,2) + 1; %this shall alternate
between 1 and 2, respectively corresponding to atoms A and B.

                switch SLlayer_choice(atomSelector)
                    %TERNARY LAYER NOTE: when coeffs_C exists, coeffs_A has no subscript,
coeffs_B has 'x' subscript, coeffs_C has '1-x' subscript.
                    case 1 %ZnS(x)Se(1-x) selected
                        coeffs_A{period_counter + 1} = asf_coefficients('Zn');
                        coeffs_B{period_counter + 1} = asf_coefficients('S');
                        coeffs_C{period_counter + 1} = asf_coefficients('Se');
                        C11_constants{period_counter + 1} = [104.62 87.2];%[ C11(ZnS)
C11(ZnSe) ]
                        C12_constants{period_counter + 1} = [65.33 52.4]; %[ C12(ZnS)
C12(ZnSe) ]
                    case 2 %In(x)Ga(1-x)As selected
                        coeffs_A{period_counter + 1} = asf_coefficients('As');
                        coeffs_B{period_counter + 1} = asf_coefficients('In');
                        coeffs_C{period_counter + 1} = asf_coefficients('Ga');
                        C11_constants{period_counter + 1} = [83.29 118.4]; %[ C11(InAs)
C11(GaAs) ]
                        C12_constants{period_counter + 1} = [45.26 53.7]; %[ C12(InAs)
C12(GaAs) ]
                    case 3 %Si(1-x)Ge(x) selected
                        coeffs_A{period_counter + 1} = [];
```

```
                        coeffs_B{period_counter + 1} = asf_coefficients('Ge');
                        coeffs_C{period_counter + 1} = asf_coefficients('Si');
                        C11_constants{period_counter + 1} = [124.0 160.1]; %[ C11(Ge)
C11(Si) ]
                        C12_constants{period_counter + 1} = [41.3 57.8]; %[ C12(Ge)
C12(Si) ]
                    case 4 % Hg(x)Cd(1-x)Te
                        coeffs_A{period_counter + 1} = asf_coefficients('Te');
                        coeffs_B{period_counter + 1} = asf_coefficients('Hg');
                        coeffs_C{period_counter + 1} = asf_coefficients('Cd');
                        C11_constants{period_counter + 1} = [53.61 53.3]; % [C11(HgTe)
C11(CdTe)]
                        C12_constants{period_counter + 1} = [36.60 36.5]; % [C12(HgTe)
C12(CdTe)]
                    case 5 % In(x)Ga(1-x)Sb
                        coeffs_A{period_counter + 1} = asf_coefficients('Sb');
                        coeffs_B{period_counter + 1} = asf_coefficients('In');
                        coeffs_C{period_counter + 1} = asf_coefficients('Ga');
                        C11_constants{period_counter + 1} = [65.92 88.50]; % [C11(InSb)
C11(GaSb)]
                        C12_constants{period_counter + 1} = [35.63 40.40]; % [C12(InSb)
C12(GaSb)]
                    case 6 % Al(x)Ga(1-x)As selected
                        coeffs_A{period_counter + 1} = asf_coefficients('As');
                        coeffs_B{period_counter + 1} = asf_coefficients('Al');
                        coeffs_C{period_counter + 1} = asf_coefficients('Ga');
                        C11_constants{period_counter + 1} = [125 118.4]; % [C11(AlAs)
C11(GaAs)]
                        C12_constants{period_counter + 1} = [53.4 53.7]; % [C12(AlAs)
C12(GaAs)]
                end

                layer_type{period_counter + 1} = layerTypes_str{layerType_choice}; %layer
type (superlattice)

                layer_material{period_counter + 1} =
SLlayers_str{SLlayer_choice(atomSelector)}; %material

                a0{period_counter + 1} =
SLlayers_latticeParams{SLlayer_choice(atomSelector)}; %lattice parameter

                x(period_counter + 1) = SLx(atomSelector); %composition

                if strainchoice == 3 %custom strain selected, so calculate in-plane
strain for superlattice layer

                    a0A = latticeparam_calc(a0{period_counter + 1}, SLx(1));
                    a0B = latticeparam_calc(a0{period_counter + 2}, SLx(2));
                    a0_ave = (a0A*SLthickness(1) + a0B*SLthickness(2)) / (SLthickness(1)
+ SLthickness(2));
                    a_SL = a0_ave * (1 + inplanestrain);
                    if atomSelector == 1
                        epsilon_par(period_counter + 1) = (a_SL - a0A) / a0A;
                    else
                        epsilon_par(period_counter + 2) = (a_SL - a0B) / a0B;
                    end
                end
                epilayer_tilt(period_counter +1) = 0;
                layer_thickness(period_counter + 1) = SLthickness(atomSelector);

            end

            n_laminae = n_laminae + 2 * num_periods - 1;

        otherwise % no more layers selected
            break;
    end %switch layerType_choice
```

```
    layer_type{n_laminae + 1} = layerTypes_str{layerType_choice}; %set layer type string

    n_layers = n_layers + 1; %increment because more layers are desired
    n_laminae = n_laminae + 1;
end % while 1 == 1

% Reflection (hkl)
reflection = config('reflection');
if reflection == -inf
    reflection_choice = menu('Choose the desired reflection:',reflection_str);
    reflection = char(reflection_str(reflection_choice));
end
h = str2double(reflection(2)); k = str2double(reflection(3)); l =
str2double(reflection(4));
% /Reflections



% Angle
theta_Start = config('theta_Start') * arcsec2rad;
if theta_Start == -inf
    theta_Start = str2double(inputdlg('Enter the starting angle, (theta-theta_B, in arc
seconds):')) * arcsec2rad;
end
theta_End = config('theta_End') * arcsec2rad;
if theta_End == -inf
    theta_End = str2double(inputdlg('Enter the ending angle (theta-theta_B, in arc
seconds):')) * arcsec2rad;
end

broadening_width = config('broadening_width');
if broadening_width > 0
    theta_Start = theta_Start - 2 * (broadening_width * arcsec2rad);
    theta_End   = theta_End   + 2 * (broadening_width * arcsec2rad);
end



if strcmp(Property,'azimuth')
    azimuth = Value;
else
    azimuth = config('azimuth');
    if azimuth==-inf; azimuth = str2double(inputdlg('Enter the azimuth in degrees'));
end;
end

psi = azimuth * deg2rad;

% /Angle



%Number of Points Per Arcsecond
numPtsPerArcSec = config('numPtsPerArcSec');
if numPtsPerArcSec == -inf
    str2double(inputdlg('Enter the number of rocking curve intensity points to calculate
per arcsecond of theta-theta_B:'));
end

% Radiation
radiationChoice = config('radiation');
if radiationChoice == -inf
    radiationChoice = menu('Choose a radiation source wavelength:',
sourceWavelengths_str);
end
idxLambdaStart = strfind(sourceWavelengths_str{radiationChoice}, '(') + 1;
idxLambdaEnd = strfind(sourceWavelengths_str{radiationChoice}, 'A)') - 1;
```

```
lambda = str2double(sourceWavelengths_str{radiationChoice}(idxLambdaStart:idxLambdaEnd)
);
% /Radiation

% Simulation Type
simulationChoice = config('simulation');
if simulationChoice == -inf
    simulationChoice = menu('Choose the simulation type:', {'Dynamical (Takagi-Taupin)',
'Kinematical (Bartels-Hornstra-Lobeek/Speriosu-Vreeland)', 'Both'});
end




%% (3) COMPUTE SCATTERING AMPLITUDE OF THE SUBSTRATE USING DARWIN-PRINS FORMULA
w = waitbar(0,'Calculating rocking curve for substrate','CreateCancelBtn',
'setappdata(gcbf,''canceling'',1)');
setappdata(w,'canceling',0);
%define angles to calculate intensity at; calculate Bragg Angle theta_B for the substrate
theta_B(1) = asin(lambda*sqrt(h^2+k^2+l^2)/(2*a0{1})); %Bragg angle for substrate (index
(1) corresponds to layer zero, the substrate)
theta_Spread = theta_End - theta_Start;
numPts = round(numPtsPerArcSec * (theta_Spread * rad2arcsec));
theta_Step = theta_Spread / (numPts - 1); %(in radians)
theta = (theta_B(1)+theta_Start:theta_Step:theta_B(1)+theta_End); %vector of angles (in
radians)

%calculate substrate ASFs, structure factors
f_A(1) = asf_calc(coeffs_A{1}, sin(theta_B(1))/lambda, radiationChoice);
f_B(1) = asf_calc(coeffs_B{1}, sin(theta_B(1))/lambda, radiationChoice);
F_o(1) = sf_calc(f_A(1), f_B(1), 0,0,0);
F_H(1) = sf_calc(f_A(1), f_B(1), h,k,l);
F_HBar(1) = sf_calc(f_A(1), f_B(1), -h,-k,-l);



%calculate substrate deviation parameter eta_s
phi = acos((u1*h+v1*k+w1*l)/sqrt((u1^2+v1^2+w1^2)*(h^2+k^2+l^2)));
if (phi > theta_B)
    msgbox('Sorry, this reflection is invalid because phi is greater than the Bragg angle
(theta_B).');
    return;
end
gamma_o(1) = sin(theta_B(1) + phi);
gamma_H(1) = -sin(theta_B(1) - phi);
b(1) = gamma_o(1) / gamma_H(1);  % asymmetry factor

V(1) = a0{1}^3; %volume of cubic crystal (in angstroms^3)
Gamma(1) = (r_e * lambda^2) / (pi * V(1));
C(1) = 0.5 * (1 + cos(2*theta_B(1))^2);  % polarization factor for random polarization

eta_num = -b * (theta-theta_B(1)) * sin(2 * theta_B(1)) - 0.5 * (1-b(1)) * Gamma(1) *
F_o(1);
eta_den = sqrt(abs(b(1))) * C(1) * Gamma(1) * sqrt(F_H(1) * F_HBar(1));
eta(1,:) = eta_num / eta_den;



%calculate substrate scattering amplitude
X0 = eta(1,:) - sign(real(eta(1,:))) .* sqrt(eta(1,:) .^ 2 - 1);

%preallocate dynamical and kinematical arrays and populate the first layer with the
substrate amplitude.
Xdyn = zeros(n_laminae,numPts); %preallocate
Xkin = zeros(n_laminae,numPts);
Xdyn(1,:) = X0;                 %set the first layer with substrate scattering amplitude
Xkin(1,:) = X0;
```

```
%% (4) PERFORM DYNAMICAL / KINEMATICAL CALCULATIONS RECURSIVELY

if n_laminae > 1 %check to make sure more laminae than just substrate exist

    tStart=tic; %start stopwatch

    for n = 2:n_laminae % n=2 corresponds to layer 1, the first layer above substrate

        waitbar((n-2)/(n_laminae-1),w,['Calculating rocking curve for lamina ' num2str(n-
1)],'CreateCancelBtn', 'setappdata(gcbf,''canceling'',1)');
        if getappdata(w,'canceling')
            delete(w);
            error('User canceled operation.');
        end

        if (numel(a0{n}) > 1) %a0n used for strain calculations
            a0n = latticeparam_calc(a0{n}, x(n));
        else
            a0n = a0{n};
        end

        %determine in-plane and out-of-plane lattice constants
        switch strainchoice
            case 1 %relaxed
                a(n) = a0n;
                c(n) = a0n;
            case 2 %coherent strain
                a(n) = a0{1};
                epsilon_par(n) = (a(n) - a0n) / a0n;
                C11(n) = elasticconstant_calc(C11_constants{n}, x(n));
                C12(n) = elasticconstant_calc(C12_constants{n}, x(n));
                epsilon_perp(n) = -2 * epsilon_par(n) * (C12(n) / C11(n));
                c(n) = a0n * (1 + epsilon_perp(n));
            case 3 %custom strain
                a(n) = a0n * (1 + epsilon_par(n));
                C11(n) = elasticconstant_calc(C11_constants{n}, x(n));
                C12(n) = elasticconstant_calc(C12_constants{n}, x(n));
                epsilon_perp(n) = -2 * epsilon_par(n) * (C12(n) / C11(n));
                c(n) = a0n * (1 + epsilon_perp(n));
            case 4 %custom strain
                latticeMismatch(n) = (a0{1} - a0n) / a0n;
                epsilon_par(n) = fractionalStrain(n) * latticeMismatch(n);
                a(n) = a0n * (1 + epsilon_par(n));
                C11(n) = elasticconstant_calc(C11_constants{n}, x(n));
                C12(n) = elasticconstant_calc(C12_constants{n}, x(n));
                epsilon_perp(n) = -2 * epsilon_par(n) * (C12(n) / C11(n));
                c(n) = a0n * (1 + epsilon_perp(n));
        end

        if numel(a0{n}) > 1 %ternary layer (if > 1, a nonscaler value exists in a0{n},
indicating polynomial form of relaxed lattice parameter)
            a0{n} = latticeparam_calc(a0{n}, x(n)); %calculate lattice parameter based on
current composition, x(n)

            theta_B(n) = asin((lambda/2) * sqrt( (h/a(n))^2 + (k/a(n))^2 + (l/c(n))^2) );
%Bragg angle for nth laminate

            %calculate ASFs for a ternary layer
            if numel(coeffs_A{n}) == 0 %two-atom ternary layer, such as Si(1-x)Ge(x)
                f_A(n) = asf_calc2(coeffs_B{n}, coeffs_C{n}, x(n),
sin(theta_B(n))/lambda, radiationChoice);
                f_B(n) = f_A(n);
            else %three atom ternary layer, such as ZnS(x)Se(1-x)
                f_A(n) = asf_calc(coeffs_A{n}, sin(theta_B(n))/lambda, radiationChoice);
```

```
                f_B(n) = asf_calc2(coeffs_B{n}, coeffs_C{n}, x(n),
sin(theta_B(n))/lambda, radiationChoice);
            end
        else
            theta_B(n) = asin((lambda/2) * sqrt( (h/a(n))^2 + (k/a(n))^2 + (l/c(n))^2) );
%Bragg angle for nth laminate

            %calculate ASFs
            f_A(n) = asf_calc(coeffs_A{n}, sin(theta_B(n))/lambda, radiationChoice); %ASF
for atom A
            f_B(n) = asf_calc(coeffs_B{n}, sin(theta_B(n))/lambda, radiationChoice); %ASF
for atom B
        end


        delta_phi_tet(n) = acos( (l/c(n)) / sqrt( (h/a(n))^2 + (k/a(n))^2 + (l/c(n))^2 )
) - acos(l/sqrt( h^2 + k^2 + l^2));

        delta_phi_tilt(n) = epilayer_tilt(n) * arcsec2rad;


        Gamma(n) = (r_e * lambda^2) / (pi * a0{n}^3); % (capital Gamma)


        %direction cosines, asymmetry factor
        psi_0 = 0;
        gamma_o(n) = sin(theta_B(n) + (phi + delta_phi_tet(n) +
delta_phi_tilt(n)*cos(psi-psi_0)));
        gamma_H(n) = -sin(theta_B(n) - (phi + delta_phi_tet(n) +
delta_phi_tilt(n)*cos(psi-psi_0)));
        b(n) = gamma_o(n) / gamma_H(n);  % asymmetry factor


        %structure factors
        F_H(n) = sf_calc(f_A(n),f_B(n), h,k,l); %SF
        F_HBar(n) = sf_calc(f_A(n),f_B(n), -h,-k,-l);
        F_o(n) = sf_calc(f_A(n),f_B(n), 0,0,0);



        %% (4.1) Deviation Parameter
        C(n) = 0.5 * (1 + cos(2*theta_B(n))^2);  % polarization factor, for random
polarization
        T(n) = layer_thickness(n) * C(n) * (pi * Gamma(n) * sqrt(F_H(n) * F_HBar(n))) /
(lambda * sqrt(abs(gamma_o(n) * gamma_H(n)))); %thickness parameter


        % Threading Dislocation Density


        TDD_mode = config('TDD_mode');
        if TDD_mode == -inf; TDD_mode = str2double(inputdlg('Enter the threading
dislocation density mode. (1) symmetric, (2) asymmetric')); end;


        if TDD_mode == 1 % symmetric


            if n==2 %the first iteration of the layer loop
                if ~exist('TDDuniform','var'); TDDuniform = config('TDDuniform'); end;
                if (TDDuniform ~= -inf)
                    TDDinitial = TDDuniform;
                    TDDfinal = TDDuniform;
                else
                    if ~exist('TDDinitial','var'); TDDinitial = config('TDDinitial');
end;
                    if ~exist('TDDfinal','var'); TDDfinal   = config('TDDfinal'  ); end;
                end
            end
            if ((TDDinitial ~= -inf) && (TDDfinal ~= -inf)) %then user specified a linear
profile
                if n_laminae > 2
                    TDD = TDDinitial + (n-2) * (TDDfinal - TDDinitial) / (n_laminae-2);
                else
```

```
                    TDD=TDDinitial;
                end
            else %no profile specified, so ask for TDD for each layer
                TDD = config(strcat('layer_',num2str(n-1),'_TDD'));
                if TDD == -inf
                    TDD = str2double(inputdlg('Enter the threading dislocation density in
10^8/cm^2:'));
                    if ( isempty(TDD) || isnan(TDD) ); TDD = 0; end
                end
            end

        D(n) = TDD * 10^(-8); %convert input which is in 10^8/cm^2 to A^-2.  1 cm^-2 =
10^-16 A^-2

        elseif TDD_mode == 2 % asymmetric

            TDD_A = config(strcat('layer_',num2str(n-1),'_TDD_A'));
            TDD_B = config(strcat('layer_',num2str(n-1),'_TDD_B'));

            D_A(n) = TDD_A * 10^(-8); %convert input which is in 10^8/cm^2 to A^-2.  1
cm^-2 = 10^-16 A^-2
            D_B(n) = TDD_B * 10^(-8); %convert input which is in 10^8/cm^2 to A^-2.  1
cm^-2 = 10^-16 A^-2

            D(n) = D_A(n) + D_B(n); %convert input which is in 10^8/cm^2 to A^-2.  1 cm^-
2 = 10^-16 A^-2

        else
            TDD = 0;
            D(n)= 0;
            warning('TDD not specified, using zero (perfect crystal).');
        end

        if (D(n)==0) %user entered zero dislocation density or didn't specify

            eta_num = -b(n) * (theta - (theta_B(n) + delta_phi_tet(n) +
delta_phi_tilt(n)*cos(psi-psi_0))) * sin(2*theta_B(n)) - 0.5*(1-b(n))*Gamma(n)*F_o(n);
            eta_den = sqrt(abs(b(n))) * C(n) * Gamma(n) * sqrt(F_H(n) * F_HBar(n));
            eta(n,:) = eta_num ./ eta_den;

        else % calculate "effective eta"

            %number of standard deviations to calculate out to (+/- NSig)
            Nsig = config('Nsig');
            if Nsig == -Inf; Nsig = 3; end;
            disp(['Using Nsig = ' num2str(Nsig) '.']);
            BinsPerPoint = 2;
            dalpha = theta_Step / BinsPerPoint;
            dbeta = dalpha;

            fprintf('Computing effective deviation parameter for layer %i. ', (n-1));

            dislocation_distribution_type = config('dislocation_distribution_type');

            if TDD_mode == 1 % symmetric
                sigma_alpha = ( a0n*sqrt(pi*D(n)) ) / (2*sqrt(2));
                sigma_epsilon  = ( 0.127*a0n*sqrt(D(n)*abs(log(2E-7*sqrt(D(n)*10^16)))) *
tan(theta_B(n)) ) / sqrt(2);
            elseif TDD_mode == 2 % asymmetric
                if ~strcmp(reflection,'(004)'); error('Asymmetric dislocation profile not
yet implemented for reflections other than 004!'); end;
                gamma_A = 45 * deg2rad;
                gamma_B = 45 * deg2rad;
                sigma_alpha = (a0n/sqrt(2))*sqrt(pi)*sqrt(
D_A(n)*cos(psi)^2*cos(gamma_A)^2 + D_B(n)*sin(psi)^2*cos(gamma_B)^2 );
```

```
                % sigma_epsilon = 0; % TO BE IMPLEMENTED!!
                 sigma_epsilon  = ( 0.127*a0n*sqrt(D(n)*abs(log(2E-7*sqrt(D(n)*10^16)))) *
tan(theta_B(n)) ) / sqrt(2);
                 warning('sigma_epsilon not yet implemented!');
            end

            FWHM_alpha   = 2 * sqrt(2*log(2)) * sigma_alpha;
            FWHM_epsilon = 2 * sqrt(2*log(2)) * sigma_epsilon;

            NBin_alpha = ceil( 2 * Nsig * sigma_alpha   / dalpha );
            NBin_beta  = ceil( 2 * Nsig * sigma_epsilon / dbeta  );

            if (NBin_alpha < 21); NBin_alpha = 21; end %don't let NBins be less than 21.
            if (NBin_beta < 21);  NBin_beta = 21;  end
            if (mod(NBin_alpha,2) == 0); NBin_alpha = (NBin_alpha + 1); end %also make
sure they're odd
            if (mod(NBin_beta, 2) == 0); NBin_beta  = (NBin_beta  + 1); end

            switch dislocation_distribution_type

                case 1   % Gaussian Distribution
                    alpha = linspace( (-Nsig*sigma_alpha),   (Nsig*sigma_alpha),
NBin_alpha );
                    beta  = linspace( (-Nsig*sigma_epsilon), (Nsig*sigma_epsilon),
NBin_beta  );
                    P_alpha   = normpdf( alpha,0,sigma_alpha   );
                    P_epsilon = normpdf( beta, 0,sigma_epsilon );

                case 2   % Lorentzian (Cauchy) Distribution
                    dalphaGauss = (2*Nsig*sigma_alpha)   / (NBin_alpha-1);
                    dbetaGauss =  (2*Nsig*sigma_epsilon) / (NBin_beta-1);

                    NumFWHMs = 5;

                    alpha = -NumFWHMs*FWHM_alpha   : dalphaGauss : NumFWHMs*FWHM_alpha;
                    beta  = -NumFWHMs*FWHM_epsilon : dbetaGauss  : NumFWHMs*FWHM_epsilon;

                    P_alpha   = cauchypdf( alpha, 0, (FWHM_alpha/2) );
                    P_epsilon = cauchypdf( beta,  0, (FWHM_epsilon /2) );
                    P_alpha   = P_alpha   / trapz(alpha,P_alpha);
                    P_epsilon = P_epsilon / trapz(beta, P_epsilon);

                case 3   % Triangular Distribution
                    alpha = linspace(-FWHM_alpha,   FWHM_alpha,   NBin_alpha);
                    beta  = linspace(-FWHM_epsilon, FWHM_epsilon, NBin_beta);
                    P_alpha =   tripdf(NBin_alpha) / - alpha(1) ;
                    P_epsilon = tripdf(NBin_beta)  / - beta(1)  ;

                case 4   % Rectangular Distrubtion
                    alpha = linspace(-FWHM_alpha  /2, FWHM_alpha   /2, NBin_alpha);
                    beta  = linspace(-FWHM_epsilon/2, FWHM_epsilon /2, NBin_beta);
                    P_alpha   = ones(1,NBin_alpha) / FWHM_alpha;
                    P_epsilon = ones(1,NBin_beta) / FWHM_epsilon;
            end % switch dislocation_distribution_type

            fprintf('Nbin(alpha) = %.0f, Nbin(beta) = %.0f.\n',
length(alpha),length(beta));

            %determine the REAL increment in alpha and beta
            dalpha = alpha(2) - alpha(1);
            dbeta  = beta(2)  - beta(1);

            %diagnostic alpha and beta limit lines calculations
            alpha1(n-1) = alpha(1);
```

```
            beta1(n-1)  = beta(1);
            alpha_start(n-1) = theta_B(n) + alpha(1); %don't include the substrate
            alpha_end(n-1)   = theta_B(n) - alpha(1);
            beta_start(n-1)  = theta_B(n) + beta(1);
            beta_end(n-1)    = theta_B(n) - beta(1);



            for angle_idx = 1:numPts

                % Note: delta_phi_tet not present in this eta calculation because the TDD
feature should be used for symmetric reflections, in which
                % case delta_phi_tet=0 anyway.

                tStartOneAngle = tic;

                fracDone = (n-2+angle_idx/numPts)/(n_laminae-1);
                if (exist('ETC','var') && ( (n_laminae==2) || ((TDDinitial==TDDfinal) &&
(TDDinitial ~= -inf)) ) )
                    ETC = (1-fracDone) * tOneAngleMean * numPts * (n_laminae-1);
                    timeleftmins = round(10*ETC/60)/10;
                    timeleftthours = round(10*ETC/3600)/10;
                    if timeleftthours > 2
                        waitbar(((n-2+angle_idx/numPts)/(n_laminae-1)),w,['Calculating
rocking curve for lamina ' num2str(n-1) '.  Time left: ' num2str(timeleftthours) '
hours.']);
                    else
                        waitbar(((n-2+angle_idx/numPts)/(n_laminae-1)),w,['Calculating
rocking curve for lamina ' num2str(n-1) '.  Time left: ' num2str(timeleftmins) '
mins.']);
                    end
                else
                    waitbar(((n-2+angle_idx/numPts)/(n_laminae-1)),w,['Calculating
rocking curve for lamina ' num2str(n-1)]);
                end
                if getappdata(w,'canceling')
                    delete(w);
                    error('User canceled operation.');
                end

                doubleintegral = 0;

                for alpha_idx = 1:length(alpha)

                    for beta_idx = 1:length(beta)

                        dblintnum = P_alpha(alpha_idx) * P_epsilon(beta_idx) * dalpha *
dbeta;
                        dblintden = -b(n) * (theta(angle_idx) + alpha(alpha_idx) -
beta(beta_idx) - (theta_B(n) + delta_phi_tet(n) + delta_phi_tilt(n)*cos(psi-psi_0))) *
sin(2*theta_B(n))  -  0.5*Gamma(n)*F_o(n)*(1-b(n));

                        doubleintegral = doubleintegral +  (dblintnum / dblintden);

                    end %beta_idx

                end %alpha_idx

                dblintfactor = (sqrt(abs(b(n))) * C(n) * Gamma(n) * sqrt(F_H(n) *
F_HBar(n))) ^ -1;

                eta(n,angle_idx) = dblintfactor / doubleintegral;

                %Estimated time till completion (ETC)
```

```
                tOneAngle(angle_idx) = toc(tStartOneAngle); %time for angle_idx-th angle
to calc
                Nsec = 3; %number of seconds to wait, over which to average the
tOneAngles
                if ((toc(tStart) >= Nsec) && ~exist('ETC','var'))
                    tOneAngleMean = mean(tOneAngle);
                    ETC = tOneAngleMean * numPts * (n_laminae-1); %seconds
                    if ETC < 60
                        fprintf('Estimated time till completion = %.0f
seconds...\n',ETC);
                    elseif (ETC/3600) > 2 %greater than two hours
                        fprintf('Estimated time till completion = %.1f
hours...\n',(ETC/3600));
                    else
                        fprintf('Estimated time till completion = %.1f
minutes...\n',(ETC/60));
                    end
                end

            end %angle_idx

        end %if D(n)==0

        %plot(1:numPts,tOneAngle);

        %% (4.2) Scattering Amplitude

        if ((simulationChoice == 1) || (simulationChoice == 3)) %Dynamical: Takagi-Taupin
            S_1n = (Xdyn(n-1,:) - eta(n,:) + sqrt(eta(n,:).^2-1)) .* exp(-1i * T(n) *
sqrt(eta(n,:).^2-1));
            S_2n = (Xdyn(n-1,:) - eta(n,:) - sqrt(eta(n,:).^2-1)) .* exp(1i * T(n) *
sqrt(eta(n,:).^2-1));
            Xdyn(n,:) = eta(n,:) + sqrt(eta(n,:).^2-1) .* ((S_1n + S_2n) ./ (S_1n -
S_2n));
        end
        if ((simulationChoice == 2) || (simulationChoice == 3)) %Kinematical: Bartels-
Hornstra-Lobeek/Speriosu-Vreeland
            Xkin(n,:) = Xkin(n-1,:) .* exp(-1i .*2 .* eta(n,:) .* T(n)) + (1 - exp(1i .*
2 .* eta(n,:) .* T(n))) ./ (2*eta(n,:));
        end

    end % for n = 2:n_laminae % n=2 corresponds to layer 1, the first layer above
substrate

    comp_time = toc(tStart); %stop stopwatch

    switch simulationChoice
        case 1 %dynamical
            tictoc_str = 'Takagi-Taupin';
        case 2 %kinematical
            tictoc_str = 'Bartels-Hornstra-Lobeek';
        case 3 %both
            tictoc_str = 'Takagi-Taupin and Bartels-Hornstra-Lobeek';
    end
    if comp_time < 60
        fprintf('The %.0f recursion(s) of the %s equation(s) took %.0f
seconds.\n',(n_laminae-1),tictoc_str,comp_time);
    else
        fprintf('The %.0f recursion(s) of the %s equation(s) took %.1f
minutes.\n',(n_laminae-1),tictoc_str,(comp_time/60));
    end
    %disp([' -- The ' num2str(n_laminae-1) ' recursion(s) of the ' tictoc_str '
equation(s) took ' num2str(comp_time) ' seconds.']);

end % if n_laminae > 1
```

```
delete(w);

%% (4.3) Intensity and Reflectivity

%calculate the reflectivity P_H
P_H_dyn = abs(F_H(n_laminae)/F_HBar(n_laminae)) * abs(Xdyn(n_laminae, :)) .^ 2;
P_H_kin = abs(F_H(n_laminae)/F_HBar(n_laminae)) * abs(Xkin(n_laminae, :)) .^ 2;

%calculate the intensity as the magnitudes squared of the scattering amplitudes
intensity_dyn = abs(Xdyn(n_laminae, :)) .^ 2;
intensity_kin = abs(Xkin(n_laminae, :)) .^ 2;



%% (5) ADD INSTRUMENTAL BROADENING
intensityBroadened_dyn = instr_broaden(intensity_dyn, numPtsPerArcSec, config);
intensityBroadened_kin = instr_broaden(intensity_kin, numPtsPerArcSec, config);
P_H_Broadened_dyn = instr_broaden(P_H_dyn, numPtsPerArcSec, config);
P_H_Broadened_kin = instr_broaden(P_H_kin, numPtsPerArcSec, config);
%-------end (5)-----------------------------------------------------



%% (6) PLOT RESULTS
x_scale = (theta - theta_B(1)) * rad2arcsec; %plot in arc sec

%if nargin == 0 %only produce plots if not run as a function

    %6a plot the scattering intensity
    titlestr = strrep([reflection(2:end-1) ' '
sourceWavelengths_str{radiationChoice}(1:findstr(sourceWavelengths_str{radiationChoice},'
(')-2)],'alpha','\alpha');
    titlestr = [titlestr(1:end-1) '_' titlestr(end)]; %insert the underscore for
subscript 1
    RockingCurve = rockingcurve_plot(x_scale, intensity_dyn, titlestr);


    %add alpha and beta limit lines
    if exist('alpha_start','var')
        alpha_start_AS = (min(theta_B(2:length(theta_B)) + alpha1) - theta_B(1)) *
rad2arcsec;
        alpha_end_AS =   (max(theta_B(2:length(theta_B)) - alpha1) - theta_B(1)) *
rad2arcsec;
        beta_start_AS = (min(theta_B(2:length(theta_B)) + beta1) - theta_B(1)) *
rad2arcsec;
        beta_end_AS =   (max(theta_B(2:length(theta_B)) - beta1) - theta_B(1)) *
rad2arcsec;


        line( [alpha_start_AS alpha_start_AS], [1e-8 1e-5], 'Color','r' );
        line( [beta_start_AS  beta_start_AS],  [1e-8 1e-5], 'Color','g' );
        line( [alpha_end_AS alpha_end_AS], [1e-8 1e-5], 'Color','r' );
        line( [beta_end_AS  beta_end_AS],  [1e-8 1e-5], 'Color','g' );
    end


    %rockingcurve_plot(x_scale, intensity_dyn, intensity_kin, 'XRD Rocking Curve -
Scattering Amplitude. ');
    if ((length(intensityBroadened_dyn) ~= 1) || (length(intensityBroadened_kin) == -1))
%user did select a broadening fn, so plot it
        RockingCurveBroadened = rockingcurve_plot(x_scale, intensityBroadened_dyn,
[titlestr ' (broadened)']);


        if exist('alpha_start','var')
            line( [alpha_start_AS alpha_start_AS], [1e-8 1e-5], 'Color','r' );
            line( [beta_start_AS  beta_start_AS],  [1e-8 1e-5], 'Color','g' );
            line( [alpha_end_AS alpha_end_AS], [1e-8 1e-5], 'Color','r' );
            line( [beta_end_AS  beta_end_AS],  [1e-8 1e-5], 'Color','g' );
        end
```

```
        xlim( [(x_scale(1)+2*broadening_width) (x_scale(end)-2*broadening_width)] );

    end



    %6b plot the reflectivity
    %rockingcurve_plot(x_scale, P_H_dyn, P_H_kin, 'XRD Rocking Curve - Reflectivity (no
broadening function)');
    %if ((length(P_H_Broadened_dyn) ~= 1) || (length(P_H_Broadened_kin) == -1)) %user did
select a broadening fn, so plot it
    %    rockingcurve_plot(x_scale, P_H_Broadened_dyn, P_H_Broadened_kin, 'XRD Rocking
Curve - Reflectivity (w/ broadening function)');
    %end
%end
%----end (6)-------------------------------------------------



%% (7) DATA OUTPUT AND LAYER SUMMARIES
%% (7.1) layer summary
display_layersummary = config('display_layersummary');
if display_layersummary == -inf
    display_layersummary = menu('Display layer summary in the command window?', 'yes',
'no');
end


summary_matrix = {'Lamina ' 'Layer Type' 'Material' 'Thickness(μm)' 'Relaxed lattice
constant a0 (Angstroms)' 'Structure Factor F_H'}; %build the header, then append this
with rows of data in the loop below.


for n = 1:n_laminae
    if numel(findstr(layer_material{n}, '(x)')) == 1 %check if the layer is a ternary, in
which case need to report composition too
        material_str = strrep(layer_material{n},'(x)',strcat('(',num2str(x(n)),')'));
        material_str = strrep(material_str,'(1-x)',strcat('(',num2str(1-x(n)),')'));
    else
        material_str = layer_material{n};
    end
    summary_str{n} = ['LAMINA ' num2str(n-1) ': Type: ' layer_type{n} '. Material: '
material_str '. Thickness: ' num2str(layer_thickness(n)/10000) ' microns. Relaxed lattice
constant a0: ' num2str(a0{n}) 'A. Structure Factor F_H: ' num2str(F_H(n)) '. D = '
num2str(D(n) * 1e8) 'e8cm-2'];
    layer_data_str = {num2str(n-1) layer_type{n} layer_material{n}
num2str(layer_thickness(n)/10000) num2str(a0{n}) num2str(F_H(n))};
    summary_matrix = [summary_matrix; layer_data_str];


    if display_layersummary == 1
        disp(summary_str{n});
    end
end


%% (7.2) lattice profile
display_latticeprofile = config('display_latticeprofile');
if display_latticeprofile == -inf
    if strainchoice == 1 %user selected completely relaxed structure, so only ask if they
want to see profile or not
        display_latticeprofile = menu('Display lattice constant profile?', 'yes', 'no');
        if display_latticeprofile == 2  %if they selected no...
            display_latticeprofile = 6; %...then set the variable to 6, the value used
herein for 'no'.
        end
    else %user selected some sort of strain so give all profile options
        display_latticeprofile = menu('Which lattice constant profile(s) would you like
to view?', 'relaxed lattice constant (a_0)', 'in-plane (a)', 'out-of-plane (c)', 'both
(a) and (c) in same plot', 'both (a) and (c) in different plots', '(none)');
    end
```

```
end

if display_latticeprofile ~= 6 %user wants lattice constant profile(s)

    %create vectors for lattice constants a0, a, c, and distance from interface
    dist_vec(1) = -1.5*10^4;
    a0_vec(1) = a0{1};
    a_vec(1) = a(1);
    c_vec(1) = c(1);
    dist_vec(2) = 0;
    a0_vec(2) = a0{1};
    a_vec(2) = a(1);
    c_vec(2) = c(1);


    idx = 3;
    for k = 2:length(a0)
        dist_vec(idx) = dist_vec(idx-1) + 0.00001;
        a0_vec(idx) = a0{k};
        a_vec(idx) = a(k);
        c_vec(idx) = c(k);
        idx = idx + 1;
        dist_vec(idx) = dist_vec(idx-2) + layer_thickness(k);
        a0_vec(idx) = a0{k};
        a_vec(idx) = a(k);
        c_vec(idx) = c(k);
        idx = idx + 1;
    end


    if display_latticeprofile == 1 %user wants relaxed lattice constant profile
        figure; %open a new figure window
        plot(dist_vec/10^4, a0_vec);
        xlabel('distance from interface (\mum)');
        ylabel('relaxed lattice constant a_0 (in angstroms)');
        title('Relaxed Lattice Constant Profile');
    end
    if (display_latticeprofile == 2 || display_latticeprofile == 5) %user wants in-plane
lattice constant profile
        figure; %open a new figure window
        plot(dist_vec/10^4, a_vec);
        xlabel('distance from interface (\mum)');
        ylabel('in-plane lattice constant a (in angstroms)');
        title('In-Plane Lattice Constant Profile');
    end
    if (display_latticeprofile == 3 || display_latticeprofile == 5) %user wants out-of-
plane lattice constant profile
        figure; %open a new figure window
        plot(dist_vec/10^4, c_vec);
        xlabel('distance from interface (\mum)');
        ylabel('out-of-plane lattice constant c (in angstroms)');
        title('Out-of-Plane Lattice Constant Profile');
    end
    if display_latticeprofile == 4 %user wants both in- and out- of-plane lattice
constant profiles in same plot
        figure; %open a new figure window
        plot(dist_vec/10^4, a_vec, dist_vec/10^4, c_vec)
        xlabel('distance from interface (\mum)');
        ylabel(strcat('lattice constant (', char(197), ')'));
        title('In- and Out- of-Plane Lattice Constant Profile');
        legend('in-plane', 'out-of-plane', 'Location', 'Best');
    end
end


%% (7.3) strain profile
if strainchoice ~= 1 %check if user selected completely relaxed structure; if not,
proceed by asking which type of strain profile they want to view.
    disp_strainprofile = config('display_strainprofile');
    if disp_strainprofile == -inf
```

```
        disp_strainprofile = menu('Which strain profile(s) would you like to view?', 'in-
plane strain', 'out-of-plane strain', 'in- and out- of-plane strains in same plot', 'in-
and out- of-plane strains in different plots', '(none)');
    end

    if disp_strainprofile ~= 5 %user wants strain profile(s)

        %create vectors for elastic constants and distance from interface
        dist_vec(1) = -1.5*10^4;
        epsilon_par_vec(1) = epsilon_par(1);
        epsilon_perp_vec(1) = epsilon_perp(1);
        dist_vec(2) = 0;
        epsilon_par_vec(2) = epsilon_par(1);
        epsilon_perp_vec(2) = epsilon_perp(1);

        idx = 3;
        for k = 2:length(epsilon_par)
            dist_vec(idx) = dist_vec(idx-1) + 0.00001;
            epsilon_par_vec(idx) = epsilon_par(k);
            epsilon_perp_vec(idx) = epsilon_perp(k);
            idx = idx + 1;
            dist_vec(idx) = dist_vec(idx-2) + layer_thickness(k);
            epsilon_par_vec(idx) = epsilon_par(k);
            epsilon_perp_vec(idx) = epsilon_perp(k);
            idx = idx + 1;
        end

        if (disp_strainprofile == 1 || disp_strainprofile == 4) %user wants in-plane
strain profile
            figure; %open a new figure window
            plot(dist_vec/10^4, epsilon_par_vec);
            xlabel('distance from interface (\mum)');
            ylabel('in-plane strain \epsilon_|_| (dimensionless)');
            title('In-Plane Strain Profile');
        end
        if (disp_strainprofile == 2 || disp_strainprofile == 4) %user wants out-of-plane
strain profile
            figure; %open a new figure window
            plot(dist_vec/10^4, epsilon_perp_vec);
            xlabel('distance from interface (\mum)');
            ylabel('out-of-plane strain \epsilon_\perp (dimensionless)');
            title('Out-of-Plane Strain Profile');
        end
        if disp_strainprofile == 3 %user wants both in- and out- of-plane strain profiles
in same plot
            figure; %open a new figure window
            plot(dist_vec/10^4, epsilon_par_vec, dist_vec/10^4, epsilon_perp_vec)
            xlabel('distance from interface (\mum)');
            ylabel('strain (dimensionless)');
            title('In- and Out- of-Plane Strain Profile');
            legend('in-plane', 'out-of-plane', 'Location', 'Best');
        end
    end
end

%% (7.4) put data on clipboard
if config('PutDataOnClipboard')

    switch simulationChoice
        case 1 %dynamical
            %stuff = {'x scale' x_scale; 'intensity_dyn' intensity_dyn}'; %tried to add
header 4/18
            if length(intensityBroadened_dyn) == 1
                stuff = [x_scale;intensity_dyn]';
            else
                stuff = [x_scale;intensity_dyn;intensityBroadened_dyn]';
            end
        case 2 %kinematical
```

```
            if length(intensityBroadened_kin) == 1
                stuff = [x_scale;intensity_kin]';
            else
                stuff = [x_scale;intensity_kin;intensityBroadened_kin]';
            end
        case 3
            if length(intensityBroadened_dyn) == 1
                stuff = [x_scale;intensity_dyn;intensity_kin]';
            else
                stuff =
[x_scale;intensity_dyn;intensityBroadened_dyn;intensity_kin;intensityBroadened_kin]';
            end
    end


    num2clip(stuff);
    fprintf('XRD data copied to clipboard.\n');
end


%% (7.5) write data to Excel
filename = config('filename');
if (filename == -inf)
    filename = char(inputdlg(['Write Data to Excel: Enter filename.xls or nothing if no
XLS desired.']));
elseif (filename == inf)
    filename = []; % make it empty if config file has inf, this means no XLS file, and no
question for XLS file
end
if (isempty(filename) == 0) %user wants to write an XLS
    success = 1;


    layerSummaryTitle = cell(5,6);
    layerSummaryTitle(1) = {'X-RAY DIFFRACTION SIMULATION SUMMARY'};
    layerSummaryTitle(2) = {strcat('Reflection:',reflection)};
    layerSummaryTitle(3) = {strcat('Radiation:',sourceWavelengths_str{radiationChoice})};


    layerSummary = [layerSummaryTitle;summary_matrix]; % add a header to the cell array
    success = xlswrite(filename, layerSummary, 'Layer Summary') * success;


    switch simulationChoice
        case 1 %dynamical
            dataArray_str = num2cell([x_scale;intensity_dyn]');
        case 2 %kinematical
            dataArray_str = num2cell([x_scale;intensity_kin]');
        case 3
            dataArray_str = num2cell([x_scale;intensity_dyn;intensity_kin]');
    end


    dataArrayWithHeader_str = {layerSummary;{['Theta-ThetaB(arcsec)']
['Intensity(a.u.)']};dataArray_str};
    success = xlswrite(filename, dataArrayWithHeader_str, 'XRD Data') * success;


    if intensityBroadened ~= -1 %user did select a broadening fn, so stick this in XLS
file too
        dataArray_Broadened_str = num2cell([x_scale;intensityBroadened]'); % format the
table properly for excel
        dataArrayWithHeader_Broadened_str = [{['Theta-ThetaB(arcsec)'] ['Broadened
Intensity (a.u.)']};dataArray_str];
        success = xlswrite(filename, dataArrayWithHeader_Broadened_str, 'Broadened XRD
Data') * success;
    end


    if success == 1
        msgbox('Excel file successfully created.')
    else
        msgbox('ERROR: Excel file creation failed.')
    end
end
```

```
%% (7.6) report the rocking curve width across the whole thing.
if ((simulationChoice == 1) || (simulationChoice == 3))
    rockingCurveWidth_Dynamical =
myfwhm(x_scale,intensity_dyn,x_scale(1),x_scale(numel(x_scale)));
    fprintf('rockingCurveWidth_Dynamical = %.2f"\n',rockingCurveWidth_Dynamical);
    if length(intensityBroadened_dyn) > 1
        rockingCurveWidth_Dynamical_Broadened =
myfwhm(x_scale,intensityBroadened_dyn,x_scale(1),x_scale(numel(x_scale)));
        fprintf('rockingCurveWidth_Dynamical_Broadened =
%.2f"\n',rockingCurveWidth_Dynamical_Broadened);
    end
end
if ((simulationChoice == 2) || (simulationChoice == 3))
    rockingCurveWidth_Kinematical =
myfwhm(x_scale,intensity_kin,x_scale(1),x_scale(numel(x_scale)));
    fprintf('rockingCurveWidth_Kinematical = %.2f"\n',rockingCurveWidth_Kinematical);
    if length(intensityBroadened_kin) > 1
        rockingCurveWidth_Kinematical_Broadened =
myfwhm(x_scale,intensityBroadened_kin,x_scale(1),x_scale(numel(x_scale)));
        fprintf('rockingCurveWidth_Kinematical_Broadened =
%.2f"\n',rockingCurveWidth_Kinematical_Broadened);
    end
end
fprintf('XRD Simulation complete.\n');

%% (7.7) return the XRD data (function)
if nargin > 0 %only bother to do this if function was called with input config file
    switch simulationChoice
        case 1 %dynamical
            if length(intensityBroadened_dyn) == 1
                header = [{'Theta-ThetaB(")'} {'IntensityDyn'}];
                XRDdata = [header; num2cell(x_scale') num2cell(intensity_dyn')];
            else
                header = [{'Theta-ThetaB(")'} {'IntensityDyn'}
{'IntensityDynBroadened'}];
                XRDdata = [header; num2cell(x_scale') num2cell(intensity_dyn')
num2cell(intensityBroadened_dyn')];
            end
        case 2 %kinematical
            if length(intensityBroadened_kin) == 1
                header = [{'Theta-ThetaB(")'} {'IntensityKin'}];
                XRDdata = [header; num2cell(x_scale') num2cell(intensity_kin')];
            else
                header = [{'Theta-ThetaB(")'} {'IntensityKin'}
{'IntensityKinBroadened'}];
                XRDdata = [header; num2cell(x_scale') num2cell(intensity_kin')
num2cell(intensityBroadened_kin')];
            end
        case 3 %both
            if length(intensityBroadened_dyn) == 1
                header = [{'Theta-ThetaB(")'} {'IntensityDyn'} {'IntensityKin'}];
                XRDdata = [header; num2cell(x_scale') num2cell(intensity_dyn')
num2cell(intensity_kin')];
            else
                header = [{'Theta-ThetaB(")'} {'IntensityDyn'} {'IntensityDynBroadened'}
{'IntensityKin'} {'IntensityKinBroadened'}];
                XRDdata = [header; num2cell(x_scale') num2cell(intensity_dyn')
num2cell(intensityBroadened_dyn') num2cell(intensity_kin')
num2cell(intensityBroadened_kin')];
            end
    end
end
if ~exist('RockingCurveBroadened', 'var')
    RockingCurveBroadened = [];

end
```