THORIUM-BASED MIRRORS IN THE EXTREME ULTRAVIOLET

by

Nicole Farnsworth

Submitted to Brigham Young University in partial fulfillment
of graduation requirements for University Honors

Department of Physics and Astronomy

March 2005

R. Steven Turley, Advisor

Eric N. Jellen, Honors Representative

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. R. Steven Turley for the countless hours he has spent with me on this project. I would also like to thank Dr. David D. Allred for his assistance and care. I am grateful to all of the BYU Thin Films Research Group for their valuable assistance on this project. This project was funded in part by the BYU Physics Department and by the National Science Foundation. I would also like to thank my parents and my sisters for their love and support through my undergraduate experience.

Contents

List of Tables

List of Figures

ABSTRACT

THORIUM-BASED MIRRORS IN THE EXTREME ULTRAVIOLET

Nicole Farnsworth

Department of Physics and Astronomy

Bachelor of Science

As applications for optics in the EUV ($100 - 1000$ Angstroms) have increased, the demand for better reflectors in this region has also increased. Here, thorium and thorium dioxide films were deposited using RF-sputtering and characterized with atomic force microscopy (AFM), x-ray diffraction (XRD), and x-ray photoelectron spectroscopy (XPS). We have measured the reflection and transmission of these films at the Advanced Light Source in order to determine the optical constants of thorium and thorium oxide. Thorium films were found to be characteristically rough with an RMS roughness of 36 Å over a 1000×1000 Å area and an RMS roughness of 43 Å over a $10,000 \times 10,000$ Å area. We have also found that thorium tends to oxidize on the surface. In our sample we found nearly linear oxidation for 50 Å. We have attempted to account for this roughness and oxidation using three methods, the Debye-Waller and Nevot-Croce scalar correction factors and the finite differences approximation. We conclude that the best way to account for roughness and surface oxidation in our program is to employ a combination of the Nevot-Croce factor and the finite

differences approximation. We have found that the best way to avoid problems with oxidation is to use reactively sputtered thorium dioxide films. We have found values for δ for thorium oxide in the region from $115-250$ Å. We have found that we cannot find β values that we believe because reflectance and transmittance measurements give us two different and irreconcilable sets of data. We are unsure at this point why the two measurements give us different answers.

Chapter 1

Introduction

1.1 Interest in the Extreme Ultraviolet and Thorium-based Mirrors

The extreme ultraviolet (EUV), $100-1000$ Å, has been increasing in importance in recent years as its applications have increased. One of the most important applications of EUV wavelengths is in computer chip lithography. The computer industry is rapidly approaching the limit of the size of features that can be fabricated with current optical lithography technology. Using EUV wavelengths of $110-130$ Å would allow lithography of smaller, faster, and more efficient circuits that operate at lower temperatures. For biological and medical research, smaller images can be taken with EUV light than can be taken with optical wavelengths, due to the fact that light diffracts around an object that is on the order of its wavelength. Also, in the range of $20-40$ Å wavelengths, water is reasonably transparent while carbon is opaque. Thus samples can be imaged without complex dehydration and staining procedures. EUV astronomy is also a blossoming field. Helium, the second most prevalent element in the universe, has both an atomic and an ionic emission line in the extreme ultraviolet and thus can be imaged in this wavelength region. For all of these developing applications, it is necessary not only to have well-designed and well-researched optics, but highly

reflective materials to make processes accurate and efficient. For this reason, the optical properties of new materials must be measured in this region for use in thin film optics design.

1.2 Project Background

One of the main reasons why technology used for visible light optics has not become available in the extreme ultraviolet is the lack of reliable optical constants in this region. For several years, Brigham Young University's Thin Films Research Group has been studying uranium as a possible candidate for high-reflectance in the EUV. Theoretically, uranium's high density and large number of electrons would make it a good reflector in this region. Although uranium has been shown to be very reflective, it has many problems associated with its application, namely, oxidation. When exposed to oxygen, pure uranium will oxidize in a matter of minutes to one of several states including UO_2, U_2O_5, UO_3, and often two or more of these states will coexist in the same sample. These oxidation states have significantly different optical properties than elemental uranium. Because the exact state of the material in a uranium film is not known, it is nearly impossible to predict how it will reflect[1].

Thorium-based films provide a possible alternative to uranium films in that they also may be more reflective than commonly used EUV reflectors, but they avoid problems with multiple oxidation states because thorium only has one oxidation state, ThO_2. Thorium should be a good reflector in the EUV because it also has large numbers of electrons, and although its density (11.7 g/cm^3) is considerably less than that of pure uranium (19.3 g/cm^3), its density is very similar to that of uranium dioxide (10.96 g/cm^3).

The Center for X-ray Optics at Lawrence Berkeley National Labs[2] provides a program at this website where reflectance is calculated by interpolation between measured points and atomic scattering factor estimates. For compounds, CXRO weights

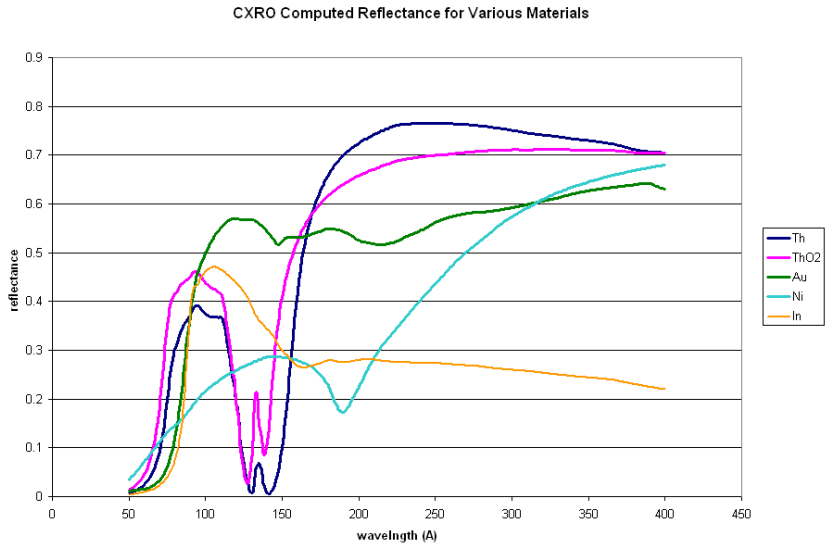

Figure 1.1: CXRO computed reflectance as a function of wavelength at 15°. Although gold, nickel, and indium are commonly used reflector materials in this region, thorium based mirrors may be more reflective at wavelengths longer than 160 Å.

elemental scattering factors according to their relative abundances. In Figure 1.1, CXRO reflectance data for thorium metal and thorium oxide is plotted with the data of two commonly used reflectance materials in this region, gold and nickel. According to CXRO, thorium-based films may be more reflective than gold or nickel films for wavelengths longer than 160 Å. In order to take advantage of its potentially high reflectance, it is necessary to accurately determine the optical constants for thorium based films in the EUV.

1.3 Theory

1.3.1 Optical Constants

In order to design multilayer optics using a certain material, we need to know how that material will reflect, transmit, and absorb different wavelengths of light. Reflec-

tion, transmission, and absorption in a material are governed by the way the material behaves when placed in an electromagnetic field. In matter, Maxwell's equations are given by

$$\nabla \cdot \mathbf{D} = 0 \tag{1.1}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{1.2}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{1.3}$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} \tag{1.4}$$

An amorphous material, as is our case, can be approximated as a linear, homogeneous material where

$$\mathbf{D} = \epsilon \mathbf{E} \tag{1.5}$$

$$\mathbf{H} = \frac{1}{\mu}\mathbf{B} \tag{1.6}$$

and ϵ and μ do not vary point to point. In this case, Maxwell's equations become

$$\nabla \cdot \mathbf{E} = 0 \tag{1.7}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{1.8}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{1.9}$$

$$\nabla \times \mathbf{B} = \mu\epsilon\frac{\partial \mathbf{E}}{\partial t} \tag{1.10}$$

These equations are almost identical to Maxwell's equations for electromagnetic waves in vacuum, with ϵ_0 and μ_0 replaced with ϵ and μ. Thus the speed of an electromagnetic wave in a material is given by

$$v = \frac{1}{\sqrt{\epsilon\mu}} = \frac{c}{n} \tag{1.11}$$

where n, the index of refraction of the material is given by

$$n = \sqrt{\frac{\epsilon\mu}{\epsilon_0\mu_0}}. \tag{1.12}$$

For most materials, μ is approximately equal to μ_0, making

$$n \approx \sqrt{\frac{\epsilon}{\epsilon_0}} = \sqrt{\epsilon_r} \tag{1.13}$$

where ϵ_r is the dielectric constant[3]. n is typically a complex number, and so is defined as

$$n = 1 - \delta + i\beta \tag{1.14}$$

where δ is the real part of the index of refraction and β is the imaginary part of the index of refraction or the absorption coefficient. Once this index of refraction of a material is known, it is relatively easy to calculate the reflection and transmission of s and p polarized light from a perfect boundary using boundary conditions. These boundary conditions are deduced from Maxwell's equations in integral form, which are

$$\oint_S \mathbf{D} \cdot d\mathbf{a} = Q_{f,enc} \tag{1.15}$$

$$\oint_S \mathbf{B} \cdot d\mathbf{a} = 0 \tag{1.16}$$

$$\oint_P \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_S \mathbf{B} \cdot d\mathbf{a} \tag{1.17}$$

$$\oint_P \mathbf{H} \cdot d\mathbf{l} = I_{f,enc} + \frac{d}{dt} \int_S \mathbf{D} \cdot d\mathbf{a} \tag{1.18}$$

Applying equation 1.15 to a very thin Gaussian pillbox extending just slightly into a linear material on either side of the boundary (Figure 1.2), we obtain

$$\epsilon_1 E_1^s = \epsilon_2 E_2^s \tag{1.19}$$

if there is no free charge on the surface. The same reasoning applied to equation 1.16 gives

$$B_1^s = B_2^s \tag{1.20}$$

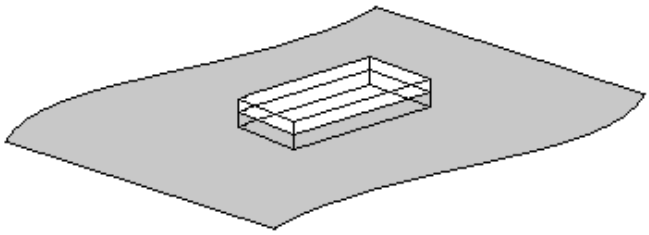

Figure 1.2: A thin Gaussian pillbox.

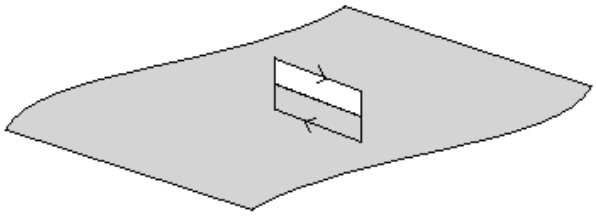

Figure 1.3: A thin Amperian loop.

Equations 1.17 and 1.18 applied to a very thin Amperian loop (figure 1.3) as the width of the loop goes to zero (zero flux) and with no free surface currents give

$$E_1^p = E_2^p \tag{1.21}$$

$$\frac{1}{\mu_1} B_1^p = \frac{1}{\mu_2} B_2^p. \tag{1.22}$$

These equations are the origin of the Fresnel coefficient equations, which are

$$f_{s,m} = \frac{q_m - q_{m-1}}{q_m + q_{m-1}} \tag{1.23}$$

$$f_{p,m} = \frac{n_{m-1}^2 q_m - n_m^2 q_{m-1}}{n_{m-1}^2 q_m + n_m^2 q_{m-1}} \tag{1.24}$$

$$g_{s,m} = \frac{2q_m}{q_{m-1} + q_m} \tag{1.25}$$

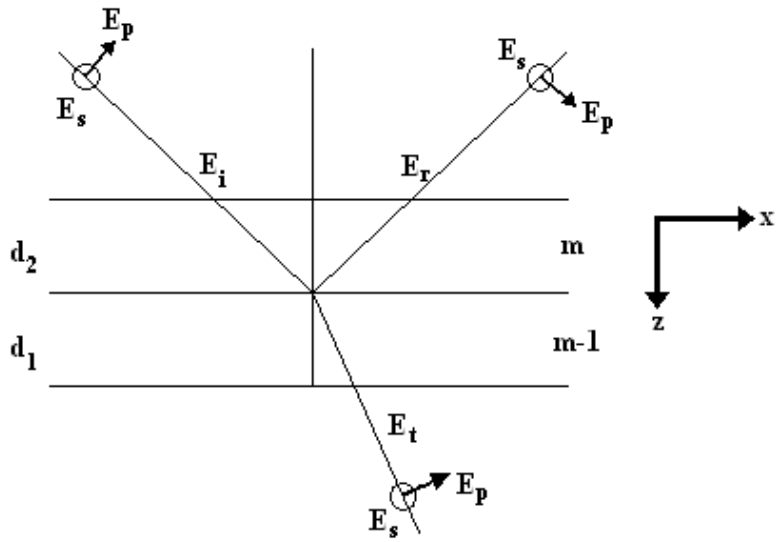

Figure 1.4: The geometry used to calculate theoretical reflectance and transmittance.

$$g_{p,m} = \frac{2n_{m-1}q_m}{n_m q_{m-1} + n_{m-q} q_m} \tag{1.26}$$

where q_m is given by

$$q_m = \sqrt{n_m^2 - \cos^2\theta_m}, \tag{1.27}$$

θ is the angle of incidence measured from grazing, and m is the m^{th} interface in the multilayer. In the visible region, the index of refraction is the preferred description of the optical properties of a material, while in the x-ray region, the preferred description is the atomic scattering factor

$$f = f_1 + if_2 \tag{1.28}$$

The relations between these two quantities are as follows

$$\delta = \frac{a_0\lambda^2}{2\pi}\frac{\rho}{A}f_1 \tag{1.29}$$

$$\beta = \frac{a_0\lambda^2}{2\pi}\frac{\rho}{A}f_2 \tag{1.30}$$

where λ is the vacuum wavelength, ρ is the density, and A is the atomic weight. These relations are derived by calculating both quantities for a free electron gas and are general descriptions of the way materials behave in the x-ray region if f_1 is the number of free electrons per atom. In the EUV electrons are not free, so the complex atomic scattering factor describes the "effective" number of free electrons for an atom. The values of f_1 and f_2 are obtained by finding the number of electrons needed to obtain the right δ and β in equations 1.21 and 1.22[4].

1.3.2 Experimentally determining optical constants

1.3.2.1 Reflectance and transmittance measurements

Experimentally determining the optical constants of a material requires taking reflectance or transmittance measurements off of a film and working backwards with the Fresnel coefficient equations to find δ and β. Reflectance and transmittance measurements are taken using a monochromator (see figure 1.5). In a monochromator, one wavelength of light is isolated using gratings, filters, and pinholes and is then either reflected off of or transmitted through a film. For reflectance, the film and the detector are rotated through a theta-two theta scan, giving a data set of reflected intensity as a function of sample angle. For transmittance, the detector is held fixed while the film is rotated around 90 degrees.

1.3.2.2 Fitting

Once reflection measurements are taken, we fit the data to the Parratt recursion relation[5,6]

$$r_{s,m} = C_m^4 \frac{f_{s,m} + r_{s,m-1}}{1 + f_{s,m} r_{s,m-1}} \tag{1.31}$$

$$r_{p,m} = C_m^4 \frac{f_{p,m} + r_{p,m-1}}{1 + f_{p,m} r_{p,m-1}} \tag{1.32}$$

where

$$C_m = \exp(-\frac{i\pi q_m d_m}{\lambda_m}) \tag{1.33}$$

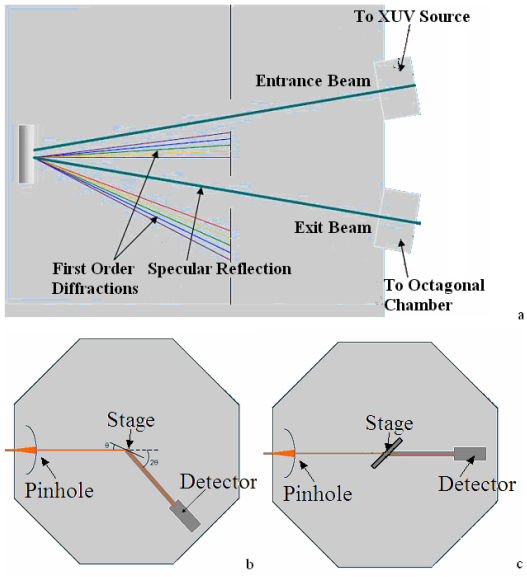

Figure 1.5: a) a monochromator b) reflection measurement c) transmission measurement

d_m is the thickness of the mth layer and λ_m is the wavelength in the layer. From these values, reflection in any layer is given by

$$R_s = |r_{s,m}|^2 \ \ and \ R_p = |r_{p,m}|^2 \tag{1.34}$$

We solve this equation by realizing that at the substrate $C_1 = 1$ because this layer is essentially infinite. We can then recursively find the reflection at each interface. For transmittance, we use a matrix formulation to calculate transmission at each interface. The implementation of this procedure will be discussed in the next section. Our fitting program uses a least-squares fitting technique to find the set of parameters that were most likely to have created the data set. The fitting library we used to analyze our data, MINUIT[7], was developed at CERN in Switzerland. Our interface, a program called matfit, defines the function and supplies the data to this library.

1.3.2.3 Transmission Calculation using Matrices

Here is a sample of my code:

```
/**Computes the reflectance and transmission of a stack at an
angle
     * theta(in degrees) using matrices.
     * @param theta angle (in degrees)
     * @param stack Vector of layer objects describing the stack.
     * Be sure to include the vacuum layer (with a thickness
     * of zero) and substrate(be sure to include thickness if you want
     * transmission data. If not, thickness=0).
     * The first layer in the stack should be the substrate.
     * Has been checked against the test program eariler in
     * this class and against the old way of computing reflectance
     * (i.e. the Parratt formula).
     * Returns t, but could return both r and t.
     */
    public double refl_trans(double theta,Vector stack){
        double kx;
        Vector klayer = new Vector();
        Complex n1,n2,k1,k2;
        Complex Fs21,Fs12,Gs21,Gs12;
        Complex Fp21,Fp12,Gp21,Gp12;
        Complex C1,C2;
        Complex Rs=new Complex(0);Complex Rp=new Complex(0);
        Complex Ts=new Complex(0);Complex Tp=new Complex(0);
        Matrix As=new Matrix(1,0,0,1);
        Matrix Ap=new Matrix(1,0,0,1);
        Matrix Bs=new Matrix(1,0,0,1);
        Matrix Bp=new Matrix(1,0,0,1);
        double r; double t;
```

kx is the x component of the vacuum wave vector k.

```
        kx=k*Math.cos(theta*Math.PI/180);
```

Gets n from the input stack and computes the z component of k in layer i.

```
        for(int i=0; i<stack.size(); i++){
            Complex n=((layer)(stack.get(i))).n;
            n=(sqr(n.times(k)).minus(sqr(kx))).sqrt();
```

```
        klayer.add(n);
    }
    for (int i=0;i<stack.size()-1;i++){
        k2=(Complex)klayer.get(i+1);
        k1=(Complex)klayer.get(i);
        n2=sqr(((layer)(stack.get(i+1))).n);
        n1=sqr(((layer)(stack.get(i))).n);
```

Fs21 is the Fresnel coefficient for s polarization giving the ratio of the amplitudes of the reflected wave to the incident wave in layer 2.

```
Fs21 = (k2.minus(k1)).over(k1.plus(k2));
Fs12 = (k1.minus(k2)).over(k2.plus(k1));
```

Gs21 is the Fresnel coefficient for s polarization giving the ratio of the amplitudes of the transmitted wave in layer 1 to the incident wave in layer 2.

```
Gs21 = new Complex(2,0).times(k1).over(k2.plus(k1));
Gs12 = new Complex(2,0).times(k2).over(k1.plus(k2));
```

Fp21 and Fp12 give the reflection coefficients for p polarization. Gp21 and Gp12 give the transmission coefficients for p polarization.

```
Fp21 =
    (sqr(n1).times(k2).minus(sqr(n2).times(k1))).over(
        sqr(n2).times(k1).plus(sqr(n1).times(k2)));
Fp12 =
    (sqr(n2).times(k1).minus(sqr(n1).times(k2))).over(
        sqr(n1).times(k2).plus(sqr(n2).times(k1)));
Gp21 =
    new Complex(2, 0).times(
        (sqr(n2).times(k1)).over(
            sqr(n1).times(k2).plus(sqr(n2).times(k1))));
Gp12 =
    new Complex(2, 0).times(
        (sqr(n1).times(k2)).over(
            sqr(n2).times(k1).plus(sqr(n1).times(k2))));
```

The coefficient C_m gives the phase and amplitude modulation of the field while propagating through the half layer m.

$$C_m = \exp\left(\frac{ik_z h}{2}\right) \tag{1.35}$$

where h is the thickness of the layer.

```
            C1 = ((new Complex(0,(((layer)(stack.get(i)))
                .thick)/2)).times(k1)).exp();
            C2 = ((new Complex(0,(((layer)(stack.get(i+1)))
                .thick)/2)).times(k2)).exp();

            As = Am(Gs21,Gs12,Fs21,Fs12,C1,C2);
            Bs = As.times(Bs);

            Ap = Am(Gp21,Gp12,Fp21,Fp12,C1,C2);
            Bp = Ap.times(Bp);
        }
        Rs = (Bs.A12).over(Bs.A22);
        Ts = new Complex(1, 0).over(Bs.A22);

        Rp = (Bp.A12).over(Bp.A22);
        Tp = new Complex(1, 0).over(Bp.A22);

        r = sFrac * sqr(Rs.mag()) + pFrac * sqr(Rp.mag());
        t = sFrac * sqr(Ts.mag()) + pFrac * sqr(Tp.mag());

        return t;
    }

    private static Matrix Am(Complex gpm,Complex gmp,Complex fpm,
            Complex fmp,Complex cm,Complex cp){

        Complex A11 = (gpm.times(cm).times(cp))
            .minus(fpm.times(fmp.times(cm.times(cp))).over(gmp));
        Complex A12 = fpm.times(cp).over(gmp.times(cm));
        Complex A21 = new Complex(-1,0).times(fmp).times(cm)
            .over(gmp.times(cp));
        Complex A22 = new Complex(1,0).over(cm.times(cp).times(gmp));
        return new Matrix(A11,A12,A21,A22);
    }
```

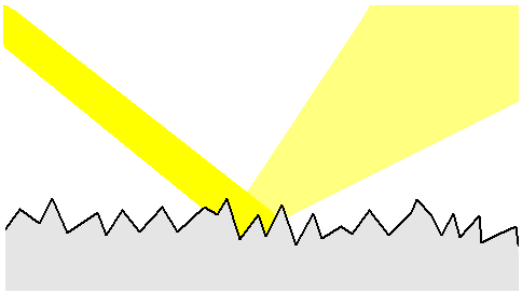

Figure 1.6: Large scale roughness results in diffuse scattering, decreasing the reflectivity of the surface at the specular angle.

1.3.3 Roughness

Reflectance from a boundary is calculated by assuming that film boundaries are infinitely sharp or transition infinitely fast. In reality, film boundaries have a finite width. Contributions from different depths within a boundary will add amplitudes with different phases, resulting in a different reflectivity than expected. The three types of roughness we will be dealing with in this paper are large-scale roughness, small-scale roughness, and interdiffusion. Large-scale roughness is defined as roughness that is much larger than the wavelength of light we are using. Large-scale roughness results in non-specular or diffuse scattering, that is, the reflected angle is no longer equal to the incident angle of light. Large-scale roughness will decrease the measured reflectance of a surface because at the expected angle, a rough surface will only reflect a fraction of the light that a smooth surface will.

Accordingly, small-scale roughness is roughness that is on the order of or smaller than the wavelength of light we are using. Small-scale roughness changes the way that light interferes when it is reflected. In a perfectly smooth plane of atoms, a reflected plane wave interferes destructively at all angles except the specular angle. If atoms are displaced, however, this reflectance will be nearly specular, but will have

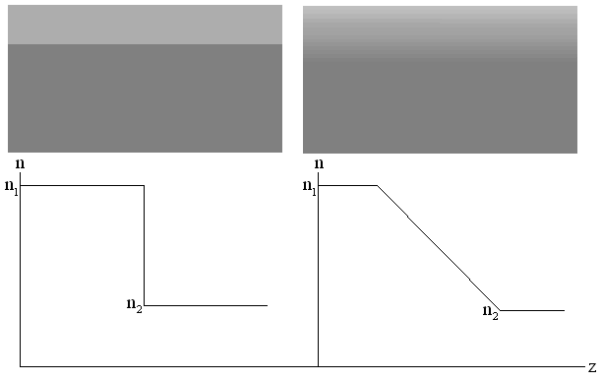

Figure 1.7: Interdiffusion of layers. In an ideal multilayer, composition changes infinitely fast from one material to another. In real multilayers, interfaces have a finite width. left: an infinitely sharp boundary right: a diffuse boundary

an amplitude that is smaller and perhaps a greater width.

Interdiffusion between layers and on the surface of a film is not roughness in the typical sense of the word. However, it is still a mechanical imperfection that affects optical properties of our materials. Interdiffusion has to do with the fact that in reality, there is no such thing as an atomically abrupt transition from one material to another. Since we are only dealing with monolayer films and we don't believe our sample is diffusing with our substrate, the interdiffusion we will deal with is surface oxidation of our sample.

1.3.4 Accepted methods of accounting for roughness

In the literature[8,9], the accepted methods of accounting for roughness are scalar correction factors. There are two prevalent scalar factors used, the Debye-Waller factor and the Nevot-Croce factor[10,11]. Both of these factors are derived using a Fourier transform method. Let's assume that the reflectivity of a boundary is smeared

out to a Gaussian distribution of width sigma around the ideal boundary.

$$r(z) = \frac{r_0}{\sigma\sqrt{2\pi}} \exp(-z^2/(2\sigma^2)) \tag{1.36}$$

where r_0 is the amplitude reflectivity of a perfectly smooth surface and σ is the RMS roughness of the surface. This sort of Gaussian distribution is obtained by adding a large number of different periodic amplitudes with random phases. Debye and Waller justified this distribution by picturing the boundary as a random superposition of lattice vibrations. Taking the Fourier transform of this boundary gives us the amplitude reflectivity of

$$r(q) = r_0 \exp(-q^2\sigma^2/2) \tag{1.37}$$

with q being the momentum transfer

$$q = \frac{4\pi}{\lambda} n \sin(\theta) \tag{1.38}$$

where λ is the vacuum wavelength, n is the index of refraction, and θ is measured from grazing. The reflected intensity is then given by

$$\begin{aligned} R(q) &= R_0 \exp(-q^2\sigma^2) && (1.39) \\ &= R_0 \exp(-(\frac{4\pi\sigma}{\lambda} n \sin\theta)^2) && (1.40) \end{aligned}$$

This factor $\exp(-q^2\sigma^2)$ is called the Debye-Waller factor. In this derivation we have assumed that the momentum transfer q is constant within the material transition. That is, this factor does not take into account how the changing index of refraction changes the propagation angle as the light refracts. Nevot and Croce produced a factor that was a correction to the Debye-Waller factor that replaced q^2 with the geometric average $q_1 q_2$. Thus the Nevot-Croce factor is given by

$$R(q) = Ro \exp(-q_1 q_2 \sigma^2) \tag{1.41}$$

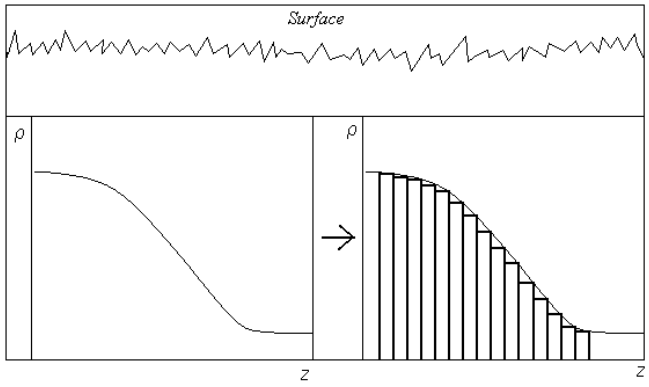

Figure 1.8: The gradient in density at an interface brought about by surface roughness can be approximated as a series of steps in density from vacuum to the material.

There is another method of accounting for roughness called the finite differences approximation. In equations 1.29 and 1.30 we can see that in the atomic scattering factor model, δ and β are proportional to the material density. Thus if the density is varying in the transition between two materials, we can say that its optical constants are behaving in approximately the same way. A varying transitional density may come about by interdiffusion between two layers, surface oxidation, or surface roughness (see Figure 1.8). In surface roughness, this density gradient is brought about by the varying heights of the surface at the interface. The zeroth order finite difference approximation is a linear variation in optical constants (see Figure 1.9).

1.3.5 Project Focus

This paper will focus on finding the optical constants of thorium oxide. I will discuss the problems we went through with thorium films including roughness and oxidation and how we decided to move to thorium oxide films. I will then discuss the processes we went through to determine the optical constants of thorium oxide and the problems with the data and results we have obtained.

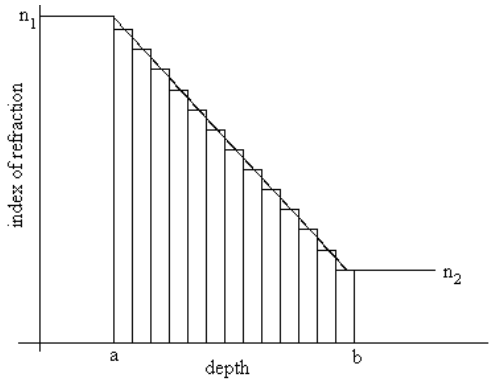

Figure 1.9: A schematic of the finite differences approximation. A gradient in the index of refraction between points a and b is approximated as a finite number of layers whose indices of refraction vary linearly from n1 to n2.

1.3.6 Film Deposition

The thorium films used in this project were deposited using RF sputtering with a base pressure of 6×10^{-7} torr and a sputtering pressure of 5 mtorr. The thorium oxide films used in this project were reactively sputtered using RF sputtering with a base pressure of 9.7×10^{-7} Torr, an oxygen partial pressure of 1.3 mtorr, and a sputtering pressure of 8.3 mtorr. With our thorium oxide samples, we had some compositional inconsistencies run to run, but the particular samples we used were determined by X-ray photoelectron spectroscopy to be homogeneous.

1.3.7 Characterization theory

1.3.7.1 X-ray Photoelectron Spectroscopy (XPS)

X-ray photoelectron spectroscopy (XPS) uses the photoelectric effect to determine the composition of a sample (see figure 1.10). The sample is irradiated with either aluminum or magnesium K-α x-rays and the energy of the ejected electrons are measured. The difference between the energy of these ejected electrons and the

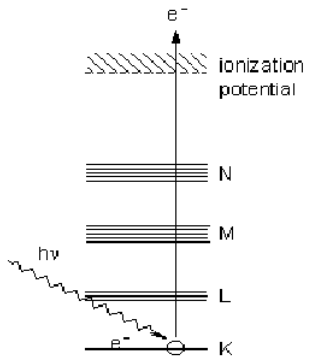

Figure 1.10: The photoelectric effect

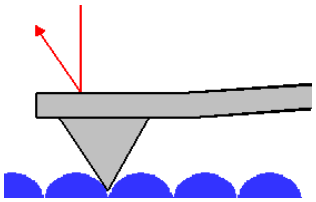

Figure 1.11: AFM tip

incident x-rays is the characteristic binding energy of the irradiated material. The measured energies can tell us the relative abundance of elements in a sample. Depth profiling can be achieved using this technique by interspersing compositional analysis with argon ion sputtering. In this way, data points can be taken of composition as a function of depth.

1.3.7.2 Atomic Force Microscopy (AFM)

Atomic force microscopy(AFM) measures surface characteristics of a sample directly by dragging a cantilever over the surface and measuring displacement using a laser (see Figure 1.11).

1.3.7.3 X-ray Diffraction (XRD)

In x-ray diffraction (XRD), copper K-α radiation ($\lambda = 1.5406$ Å) is reflected off of a sample near grazing. This scattering can be approximated as Bragg diffraction.

$$m\lambda = 2d\sin\theta \tag{1.42}$$

where m is the diffraction order, λ is the wavelength of light in the material, d is the thickness of the film, and θ is the incident angle. The thickness of a film can be estimated from XRD data by using the difference in angle between interference peaks in this equation.

Chapter 2

Roughness

Effectively modelling reflection and transmission data in order to determine the optical constants of a material requires the ability to accurately calculate the scattering of radiation off of that material, a feat that necessitates an understanding of interfaces and a robust theory of the interaction of radiation with nonideal surfaces. It is simple to calculate the reflectance and transmittance of radiation from an ideal interface using Fresnel's equations and employing a few simple assumptions, such as the assumption that the interface is perfectly smooth. However, in practice it is easily seen that these assumptions do not describe reality, and are often not a good enough approximation for our purposes. Thus, one must attempt to describe and account for these imperfections in our optics.

Our group was interested in determining the optical constants of thorium as previously discussed. Although all of the calculations made in this chapter were performed using data from a thorium sample, we hope that the same calculations could be employed for any sample. With that in mind, we first began fitting reflectance data for a single layer of thorium on a silicon substrate (sample Th04) at 150 Å with our program that assumed an ideal interface, both superficially smooth and with ideal

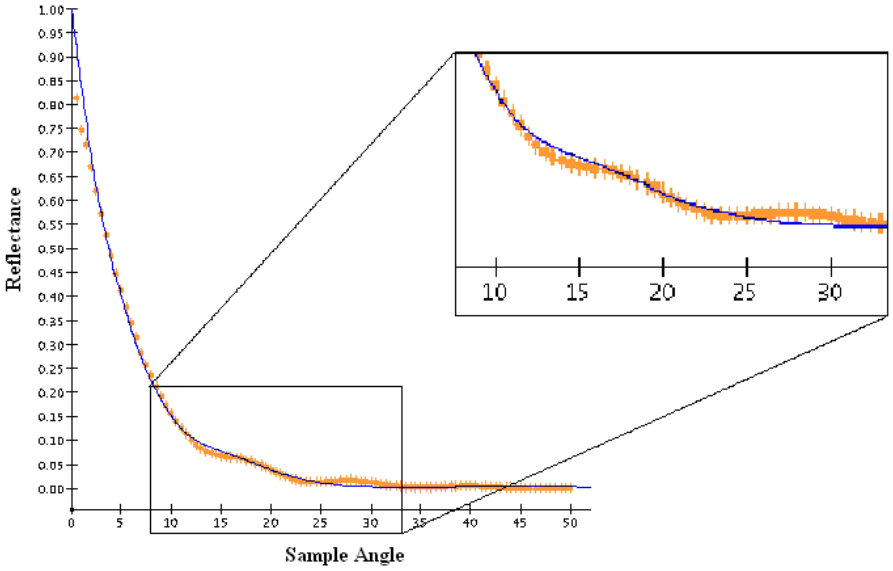

Figure 2.1: Reflectance data for a single layer of thorium on a Silicon substrate at 150 Å. Along the x-axis is plotted angle, and along the y-axis is plotted reflectance. The fit assumes smooth boundaries and abrupt interfaces. The inset shows the poorness of the fit at middle angles $(10 - 30°)$.

interfacial transitions. These fits were not very good. Referring to Figure 2.1 we can see that our fit (the blue line) is not a very good representation of the data (the orange dots). This is especially true in the middle angles from 10 to 30° where the data has some very interesting oscillations that the fit does not follow. These oscillations are the most important part of the data because they are what make the reflectance data for thorium unique. Thus if we cannot fit the oscillations well, we won't know that we have the correct fit and thus the correct optical constants.

Because of these poor fits, we are prompted to ask the question, what in our sample is not being described well in our program? We hypothesized surface roughness as well as surface oxidation. This hypothesis was supported by AFM data as well as

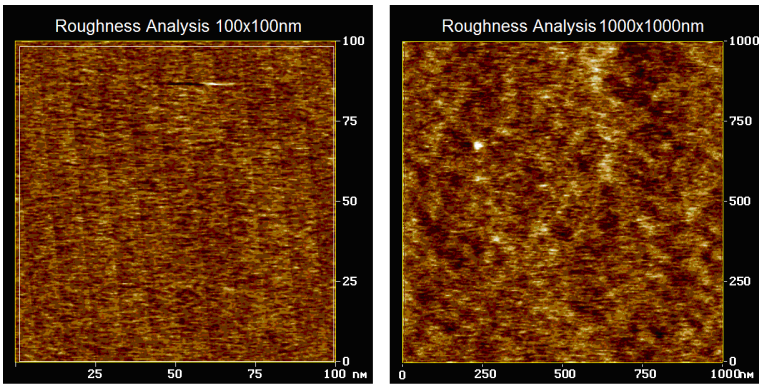

Figure 2.2: Atomic force microscopy profiles of thorium at two length scales: 1000x1000 Å and 10,000x10,000 Å. At the 1000x1000 Å length scale, RMS roughness equals 36 Å. At the 1000x1000 Å length scale, RMS roughness equals 43 Å.

XPS depth profile data. Our program assumed both a perfectly smooth surface and perfectly abrupt boundaries. Our goal then was to incorporate roughness into our program, and thus fit our data more accurately.

2.1 Characterization of Roughness

In order to characterize the magnitude and length-scale of our surface roughness, we used AFM to find the root mean squared (RMS) roughness of our sample at two length scales and the power spectral density function of our sample. On a 1000 Å by 1000 Å length scale, we found our RMS roughness to be 36 Å. This tells us that our small-scale roughness is of significant magnitude. On a 10,000 Å by 10,000 Å scale, we have an RMS roughness of 43 Å. This tells us that we also have large-scale roughness of significant magnitude (Figure 2.2). A power spectral density plot tells us the relative amounts of roughness our sample has at each horizontal length scale. From Figure 2.3 we can see that we have the highest density of roughness from 400 to 800 Å, which is on the order of the wavelength of light we are using (150 Å). This

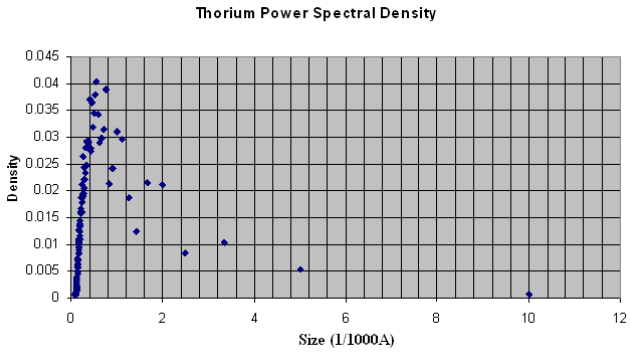

Figure 2.3: Power spectral density plot of sputtered thorium. The highest density of roughness occurs in the range from 400 to 800 Å, small-scale roughness.

tells us that although the vertical magnitude of our large-scale roughness is significant (a number we got from AFM measurements), it does not comprise a large percent of the roughness of our sample. Thus we know that we should concern ourselves only with small-scale roughness, which is significant in both horizontal density and vertical magnitude. In order to characterize the surface oxidation of our sample, we did an XPS depth profile of the surface. We found that our sample oxidized nearly linearly for about 50 Å (see Figure 2.4). The thorium-silicon edge in our plot gives us an idea about the resolution of this technique because we assume that there is no interdiffusion between our sample and the substrate.

2.2 Problems with Characterization

After characterizing our sample using AFM and XPS, we wanted to determine how far we could trust our data. One major concern we had was AFM measurements where the size of the roughness was on the order of the size of the tip used to measure it.

As can be seem from Figure 2.5, a large tip cannot characterize high-frequency

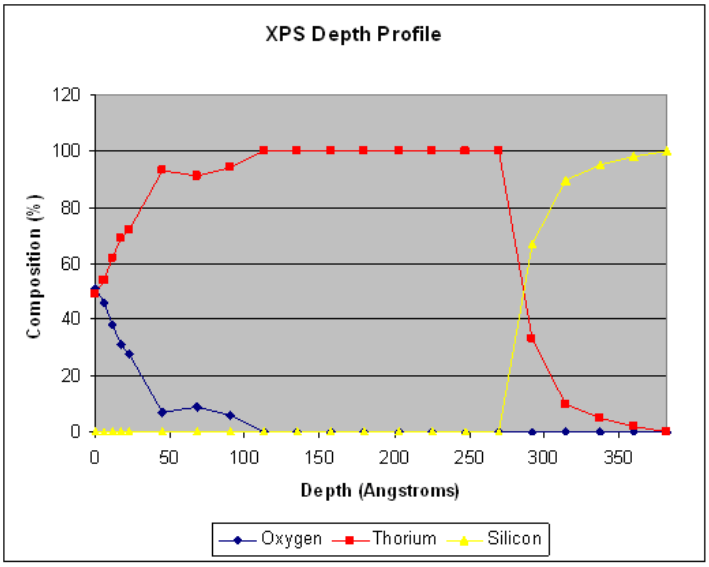

Figure 2.4: X-ray photoelectron spectroscopy depth profile of thorium. The surface of the sample has oxidized nearly linearly for about 50 Å. The thorium-silicon edge gives as an idea of the resolution of this technique because we assume that this boundary is sharp.

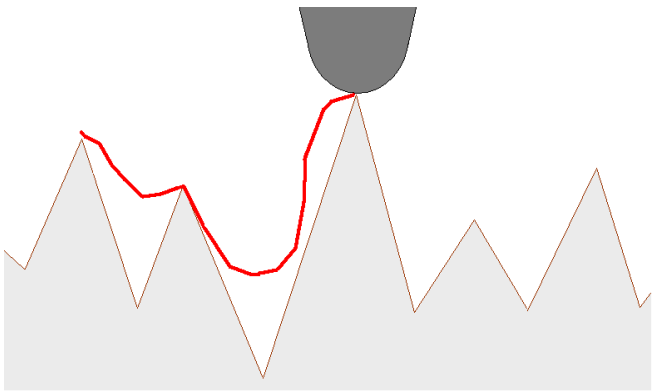

Figure 2.5: An AFM tip that is on the order of the size of the roughness being measured cannot characterize roughness accurately.

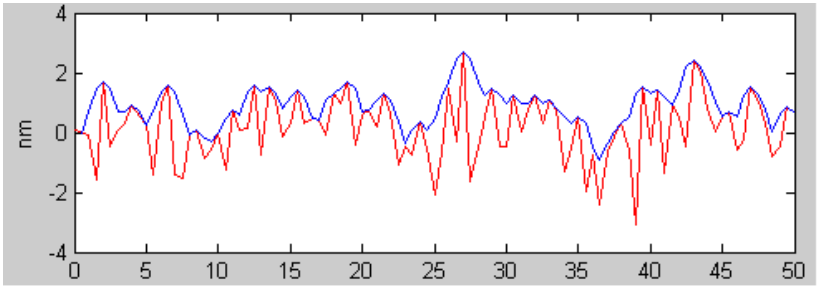

Figure 2.6: A rough surface approximated by Gaussian random numbers (red line). The blue line is the surface as measured by the AFM tip. The RMS roughness of the surface is 10.3 Å while the RMS roughness detected by the tip was 2.3 Å.

roughness accurately. In order to see how great an effect this had on the data given to us by AFM, we modeled it in Matlab. We generated a surface, dragged a tip of non-zero width across it, and computed the RMS roughnesses and power spectral densities, one of the real surface and one of the surface as it was detected by the tip. Our first approximation of our surface was a Gaussian random distribution, with mean 0 and standard deviation 1. A characteristic example is displayed in Figure 2.6. The red line represents the surface, while the blue line represents the surface as detected by the tip. For this surface, we found that the real RMS roughness was 10.3 Å while the RMS roughness reported by the finite tip was 2.3 Å. For this surface, the real RMS roughness of the surface is four times the RMS roughness reported by the tip.

The power spectral density reported by the tip is similarly skewed. Figure 2.7 shows a plot of the power spectral density of the surface (red) and the surface detected by the tip (blue). The power spectral density of the surface is flat because the roughness on the surface is random, point to point. For this surface, the tip not only fails to detect high frequency roughness on the surface, it actually detects more low frequency roughness than is actually present.

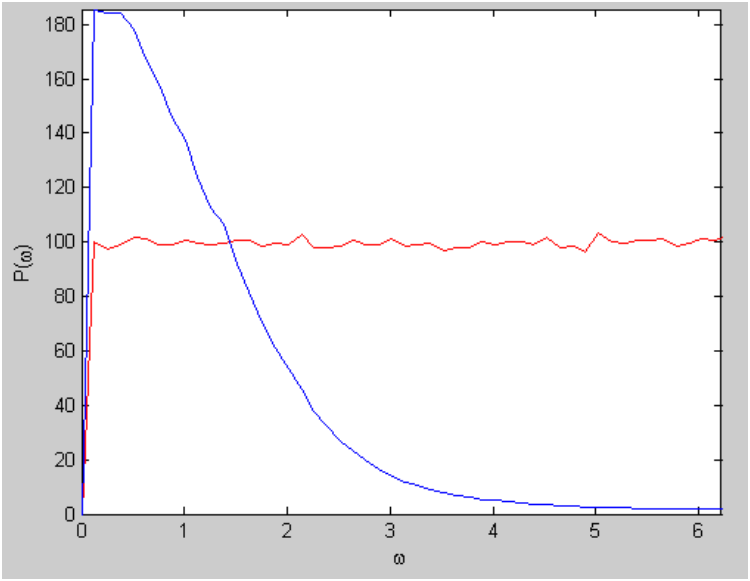

Figure 2.7: The power spectral density of the surface (red) and the surface detected by the tip (blue). For this surface, the tip not only fails to detect high frequency roughness on the surface, it actually detects more low frequency roughness than is actually present.

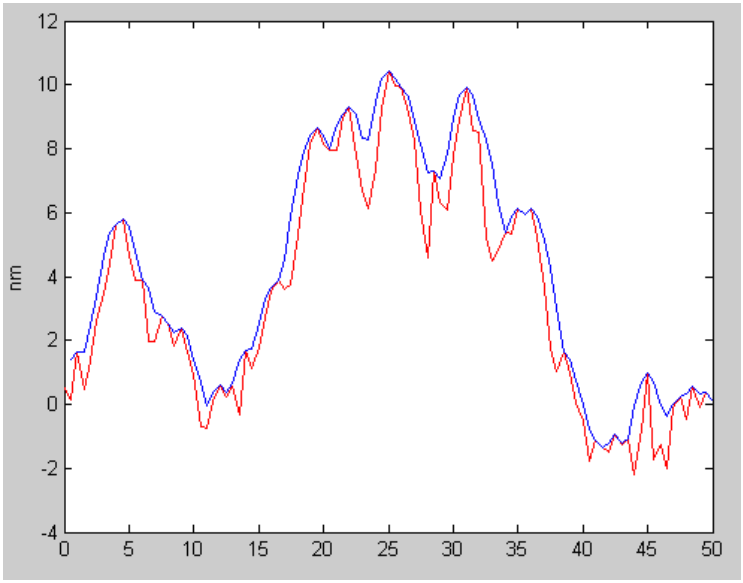

Figure 2.8: A rough surface approximated by correlated Gaussian random numbers (red line). The blue line is the surface as measured by the AFM tip. The RMS roughness of the surface is 14.9 Å while the RMS roughness detected by the tip was 13.3 Å.

This model of the surface is probably not completely accurate because on real surfaces each point is not independent of the points around it; there is some amount of correlation point to point. The next approximation of our surface was a surface where each point was random around the point previous to it. A characteristic example is displayed in Figure 2.8. The red line represents the surface, while the blue line represents the surface as detected by the tip. For this surface, we found that the real RMS roughness was 14.9 Å while the RMS roughness reported by the finite tip was 13.3 Å. For this surface, the real RMS roughness of the surface is only 1.12 times the RMS roughness reported by the tip.

Similarly, the power spectral density reported by the tip for this surface corre-

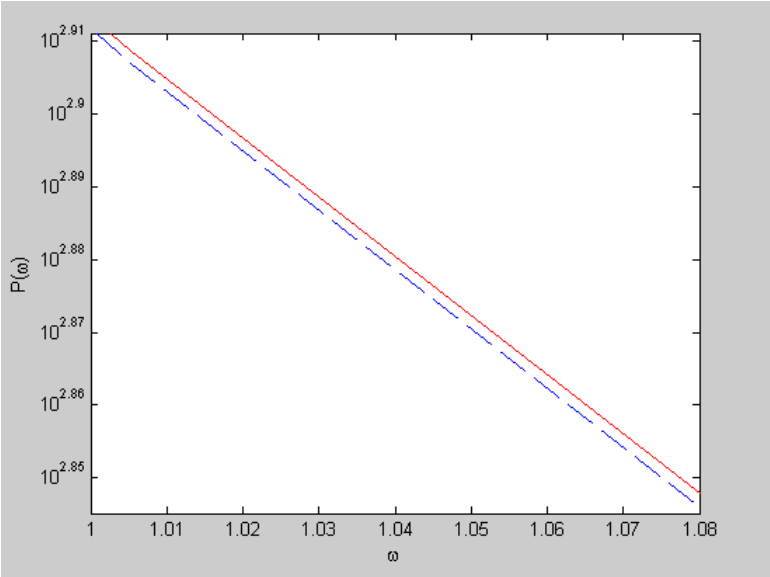

Figure 2.9: The power spectral density of the surface (red) and the surface detected by the tip (blue). For this surface, the PSD of the real surface and the surface detected by the tip match almost identically.

sponds to the actual power spectral density of the surface (see Figure 2.9). They have the same shape, but the surface detected by the tip has less roughness than the actual surface universally. If the power spectral densities were normalized, the two plots would match almost identically.

These two models represent the two extremes in the possibilities of the correlation of our surface. Our surface is probably neither completely uncorrelated nor completely correlated, but somewhere in between. Our third approximation for our surface was derived by taking a completely uncorrelated surface and at each point multiplying it by a Gaussian of width σ. By changing the value of σ, the surface can be given different length scales of correlation. As we change this value, we can see the power spectral densities of the surface and of the surface reported by the tip move closer to

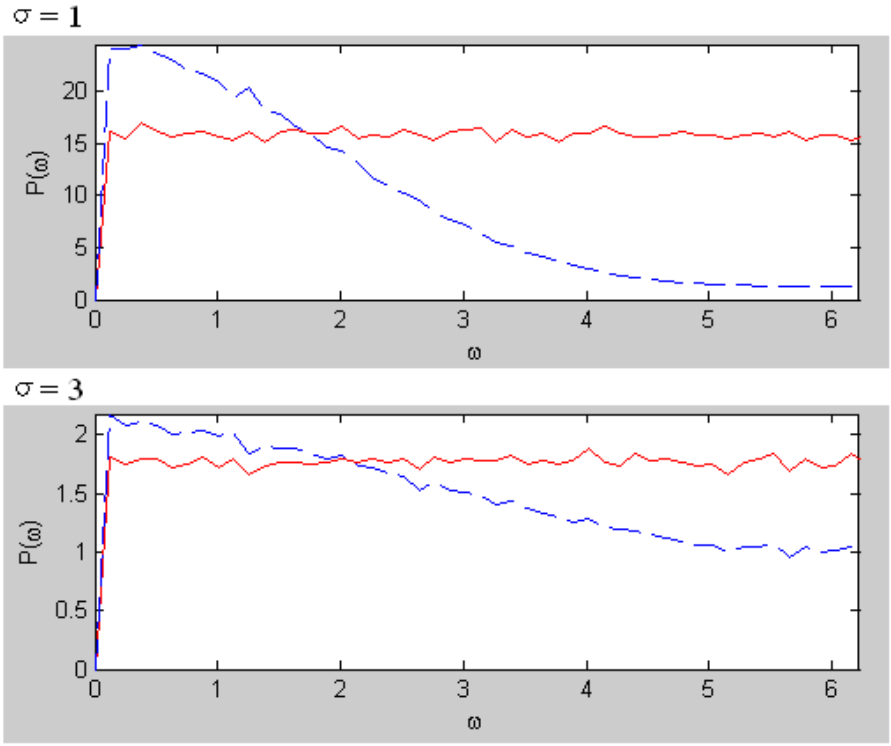

Figure 2.10: The power spectral density of the surface (red) and the surface detected by the tip blue. As the length scale of correlation gets longer, the PSD measured by the tip gets closer to the actual PSD of the surface.

one another.

2.3 Accounting for Roughness

We analyzed the efficacy of methods of accounting for roughness by correcting our data and then fitting it to see how the fit improved. First we corrected our data using the Debye-Waller scalar correction factor. The fit was still not very good, especially at our important middle angles, but the fit did improve at low angles (see Figure 2.11). Although the fit was not drastically better, we know that the correction was a step in the right direction because our data at zero degrees was closer to 100%.

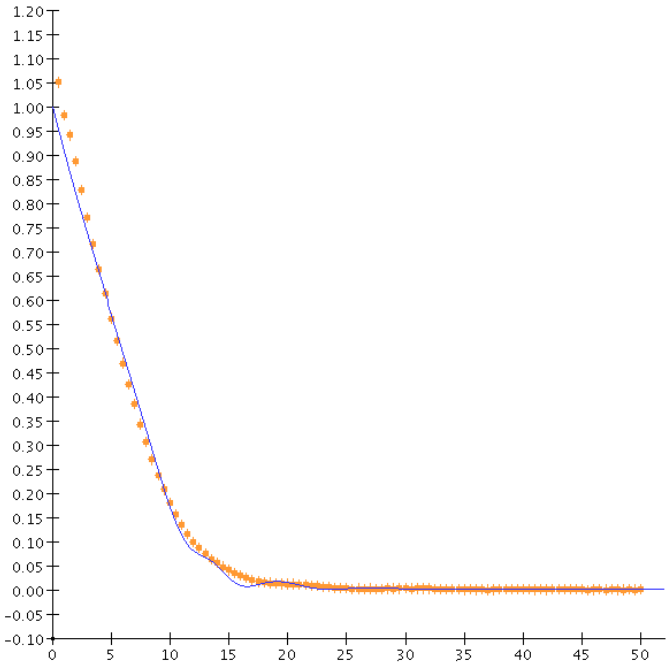

Figure 2.11: Fit of thorium reflection data after correction using the Debye-Waller factor. The factor improved the fit only at low angles.

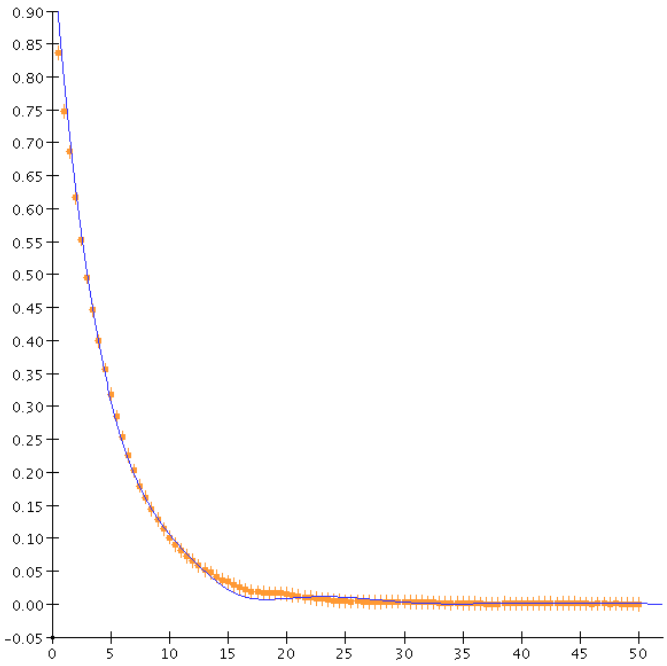

Figure 2.12: Fit of thorium reflection data after correction using the Nevot-Croce factor. The factor improved the fit significantly at low and high angles.

The fit after correction using the Nevot-Croce factor was vastly improved (see Figure 2.12). The fit was excellent at low angles and much better at high angles. The fit at middle angles was improved, but still not very good.

We employed the method of finite differences, approximating the 50 Å transition layer of oxidation and roughness in thorium with five perfect layers of 10 Å each whose index of refraction varied linearly between them. The fit we got was excellent at low and middle angles (see Figure 2.13). This was the zeroth order approximation for this type of a correction. A good area for future research would be a non-linear variation of optical constants or a trapezoidal or polynomial rather than rectangular fit.

After examining the effects of roughness and oxidation on our thorium samples,

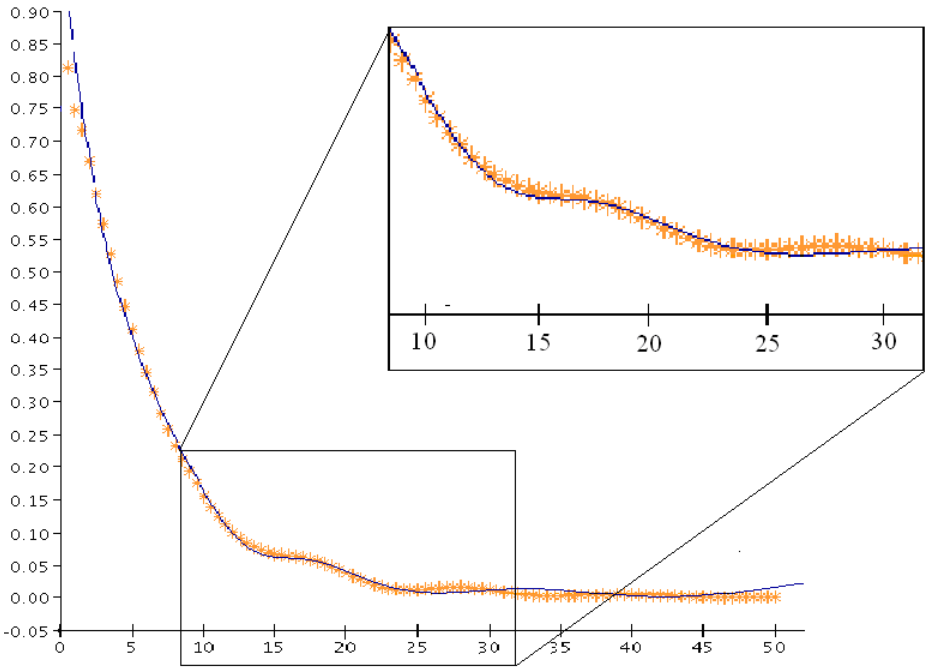

Figure 2.13: Fit of thorium reflection data with a 50 Angstrom transition layer approximated as five 10 Angstrom layers whose indices of refraction varied linearly. The approximation improved the fit significantly at low and middle angles.

we decided that dealing with an unknown thickness of naturally oxidized thorium that may or may not be changing in time was too difficult to deal with. For this reason, our focus shifted to reactively sputtered thorium dioxide, which, is inherently more stable than thorium.

Chapter 3

Finding the Optical Constants of Thorium Oxide

3.1 Reflectance and Transmittance Measurements

To find the optical constants of thorium oxide, we took both reflectance and transmittance data in order to decrease the degrees of freedom in our fitting program. We deposited thorium oxide by reactive RF sputtering on two substrates: a silicon wafer and half of a thin membrane of polyimide provided by Moxtek[12] leaving the other half of the film uncoated (sample ThO2-072604). XPS data showed this sample to be thorium dioxide and uniform through the sample (see Figure 3.1). AFM showed that this sample was not significantly rough, with an RMS roughness of 14.7 Å on a 5000x5000 Å length scale. Measurements were taken at the Advanced Light Source at Lawrence Berkeley National Lab[13]. The ALS is a synchotron light source which accelerates electrons moving at relativistic veleocities using magnetic fields. These accelerating particles emit photons at essentially continuous wavelengths from 0.2 Å-0.6 mm. As electrons emit light and also as experience collisions, they lose energy. Thus, over the space of a few hours, the beam current decreases and our reflectance

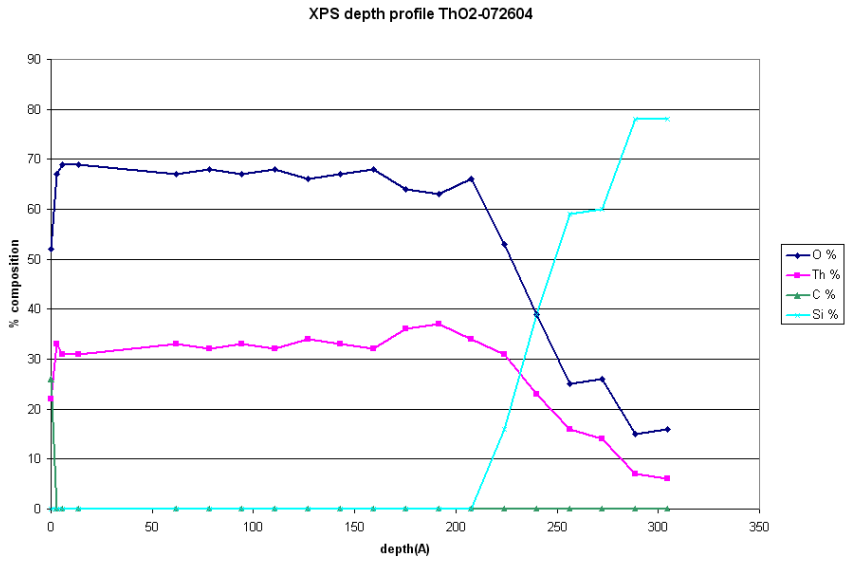

Figure 3.1: XPS depth profile of reactively sputtered ThO_2, sample ThO2-072604. The composition ratios in this profile show the sample to be consistently thorium dioxide.

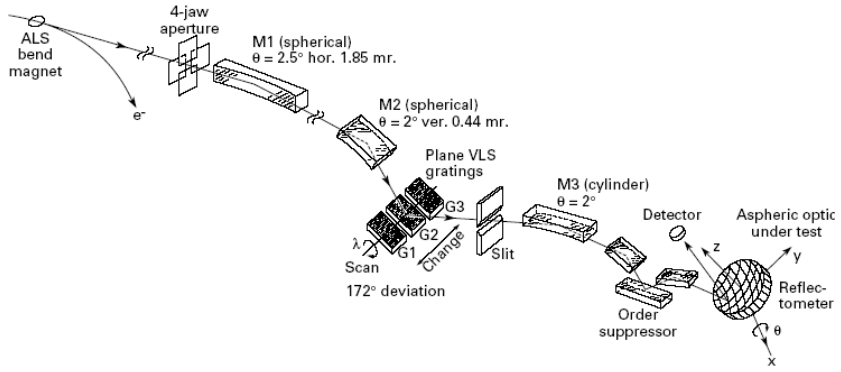

Figure 3.2: A schematic layout of ALS beamline 6.3.2.

measurements must be normalized to this beam current. This normalization procedure will be discussed shortly. Our reflectance and transmittance measurements were taken at beamline 6.3.2, a beamline with a bend magnet source and an energy range of 50-1300 eV ($9.5 - 250$ Å). A schematic of the beamline is shown in Figure 3.2. The beamline's configuration provides high spectral resolution and purity, beginning with a spherical mirror which focuses the beam onto one of three variable-line-spaced grating designed to correct for aberrations of the mirror (200 l/mm, 600 l/mm, and 1200 l/mm). Then the beam passes through a series of filters and a triple-mirror configuration intended to suppress higher order light. The wavelength is varied by rotating the grating with respect to a fixed exit slit. The configuration's resolving power ($E/\Delta E$) is 7000, and the beam size at the sample is $300 \times 10\mu$m. The beam intensity at the sample for the 600 l/mm grating and the 1200 l/mm grating is given in Figure 3.3. The beam intensity for the 200 l/mm is not shown. The resolution of the monochromator for each of the three gratings is shown in Figure 3.4. The filters used in this beamline setup are Al ($0.3\mu m$), Si ($0.3\mu m$), B ($0.5\mu m$), C ($2\mu m$), Ti ($0.6\mu m$), Cr ($0.55\mu m$), Co ($0.33\mu m$), and Cu ($0.64\mu m$). Table 3.1 gives the filter, order sorter

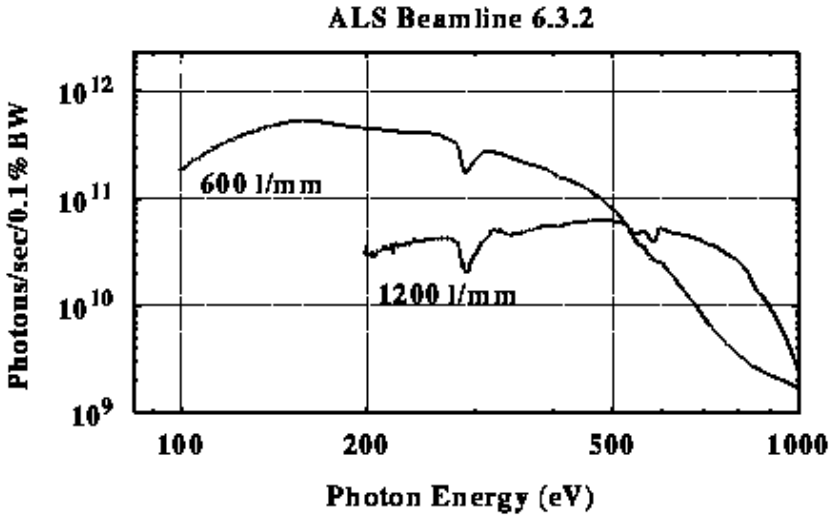

Figure 3.3: Beam intensity at the sample position.

mirror and angle, and grating used to achieve pure wavelengths in a given range. The grating, filter, and order sorter sets overlap at the edges of the wavelength ranges they are meant to produce. This becomes a problem when measurements made with two different filter sets do not overlap. We experienced this problem repeatedly in the region where the silicon and aluminum filters overlap, the region from $172 - 188$ Å. Figure 3.5 shows reflectance of ThO_2 on silicon from $165 - 195$ Å. The two measurements never agree, and are dissimilar by as much as 0.013. To find out which of these filter sets gave us better data, we placed a second grating inside the beamline's reflectometer at the sample stage. At a fixed wavelength, we rotated this grating with respect to a fixed detector. For a spectrally pure beam we should see a sharp spike at one grating angle. What we saw is depicted in Figure 3.6. The inset shows spectral impurities in the wavelength produced by the silicon filter set. From this data, we know that when the silicon and aluminum filter sets give us conflicting data, we should believe the aluminum filter set data.

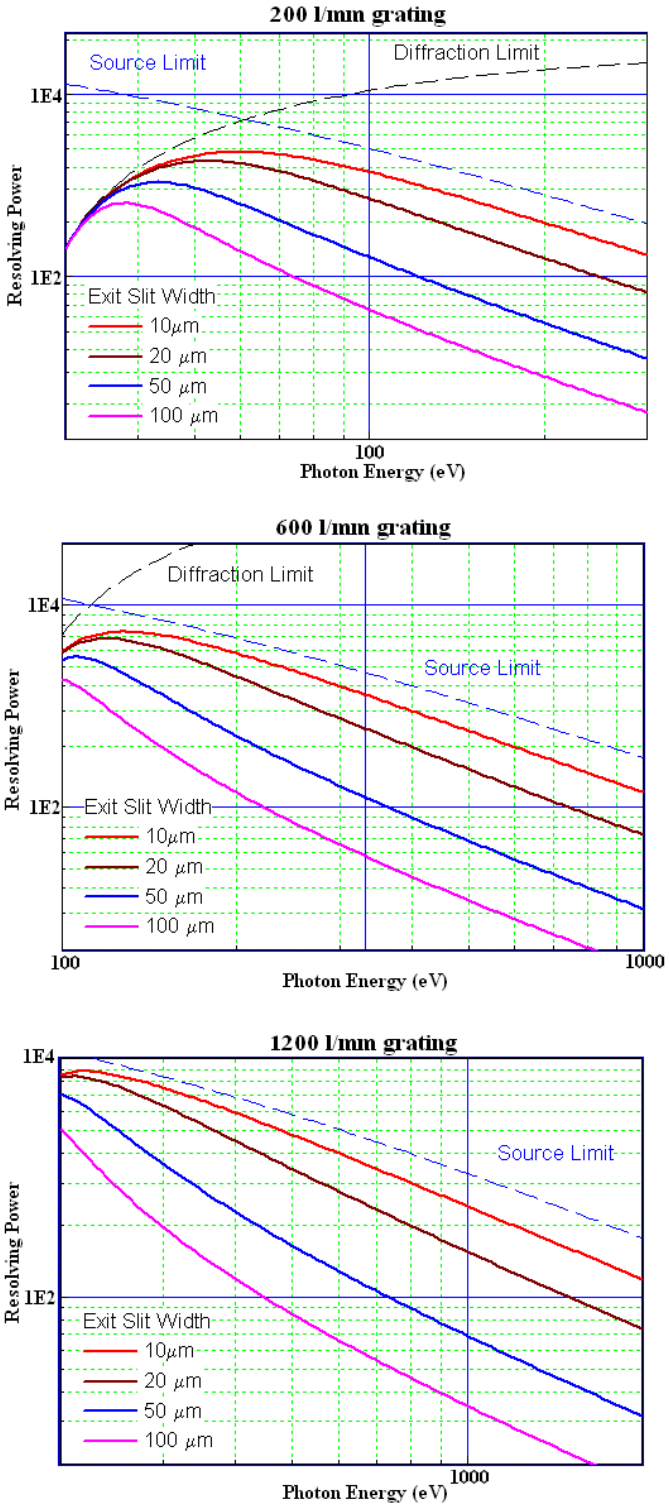

Figure 3.4: Monochromator resolution.

Wavelength Range (Å)	*Grating (l/mm)*	*Filter*	*Order Sorter, Angle*
20-28	600	Cr	Ni, 6.2°
27-48	600	Ti	Ni, 6.2°
44-68	600	C	Ni, 8°
66-88	600	B	C, 6.2°
84-116	200	B	C, 8°
110-140	200	Be	C, 8° or 10°
124-188	200	Si	C, 10°
172-250	200	Al	C, 10°

Table 3.1: Grating, filter, and order sorters used to acheive spectral purity for the given wavelength range.

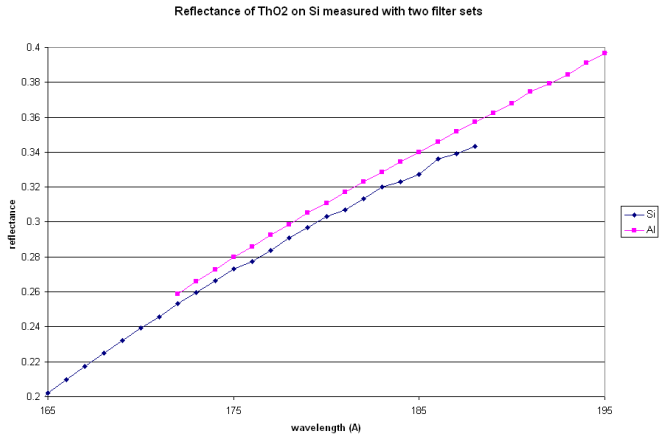

Figure 3.5: Reflectance of ThO_2 on Si measured with two filter sets at 14°. The two measurements disagree by as much as 0.013.

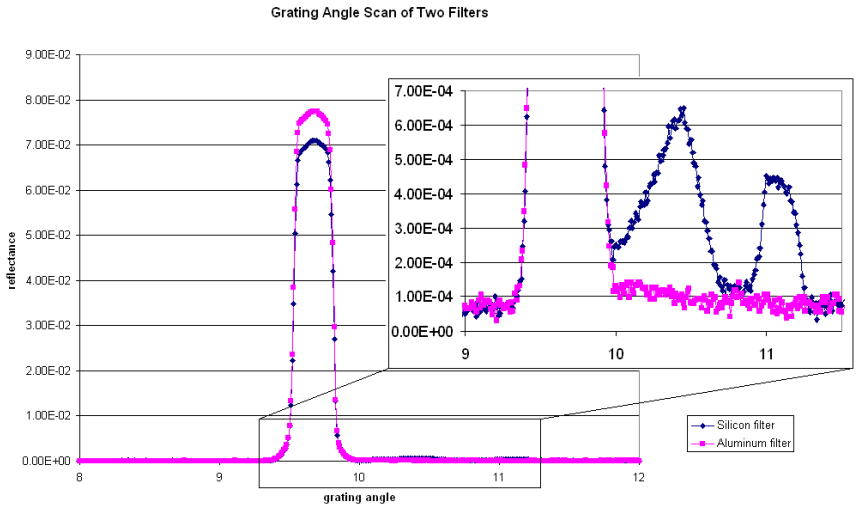

Figure 3.6: A second grating was placed at the sample stage and rotated with respect to a fixed detector. The two data sets were taken with two different filter sets with a wavelength of 180 Å. The inset shows spectral impurities in the wavelength produced by the silicon filter set.

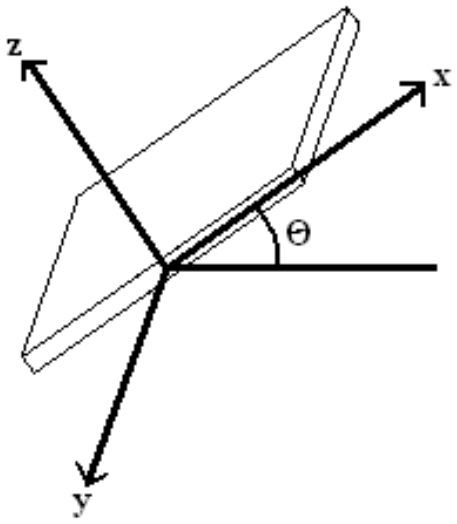

Figure 3.7: Sample stage axis specifications for beamline 6.3.2 reflectometer.

The beamline's reflectometer allows sample movement in the x, y, z, and θ directions where x and y are parallel to the sample plane, z is perpendicular to the sample plane, and θ is measured from grazing (see Figure 3.7). The translation motors can position to within 1μm and the angular motor can position to within 0.002°. The detector in the reflectometer that we used for our measurements was a photodiode. A problem we have had to deal with repeatedly while using this detector is dark current, that is, signals measured by the photodiode when no beam is on it. These signals are procured from many sources: electrical noise, visible light leakage from outside the chamber, etc. This dark current is an absolute error, and so we have to worry about it a lot more when our signal is small. Typical dark currents usually fluctuate randomly, however, dark current has been shown in some cases to depend on wavelength, sample angle, or time of day. They are usually around $1-5\times10^{-3}$. A typical scan is shown in Figure 3.8. This scan was taken as a function of wavelength,

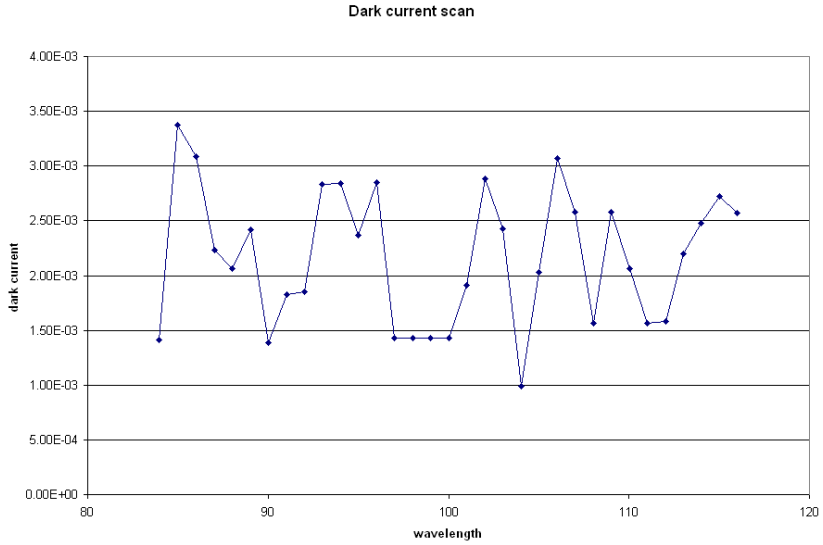

Figure 3.8: A typical dark current scan taken as a function of wavelength.

even though no light is supposed to be getting into the chamber. This scan seems to be fairly random.

Normalization procedure for our scans is as follows: first we normalize our reflectance measurements to the beam current. Second, we normalize our I_0 scan to the beam current. An I_0 scan is a wavelength scan taken with no sample, just the beam going straight into the detector. We then subtract the dark current (also normalized to the beam current) from both our reflectance measurement and our I_0 measurement. We then divide our reflectance measurement by our I_0 measurement in order to obtain percentage reflectance.

3.2 Reflectance and Transmittance Data

Figures 3.9 and 3.10 show plots of reflection as a function of wavelength and transmission as a function of wavelength for the sample deposited on polyimide film. Figure 3.11 shows a plot of reflection as a function of wavelength for the sample

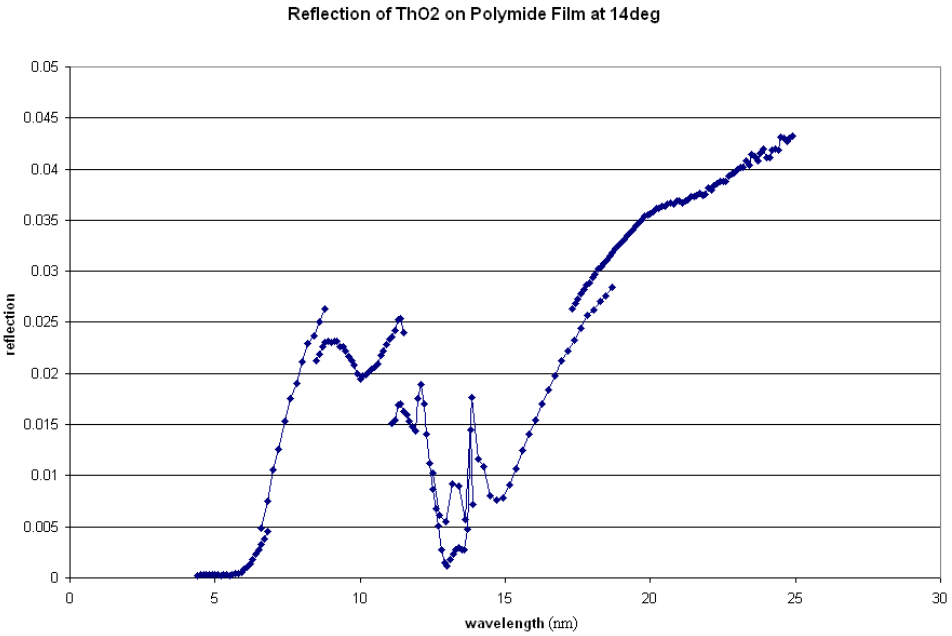

Figure 3.9: Reflection of ThO2 on polyimide film at 14 degrees by wavelength.

deposited on silicon. From the sample on polyimide film we measured both transmittance and reflectance although reflectance was only around three percent. One problem we had with these samples was that the substrate membrane was so thin that the films buckled and warped when they were deposited on. The films were visibly wavy, making it tricky to get good measurements. There were no places where the transmission looked uniform while doing a y-scan of transmission (see Figure 3.12). This tells us that something in our sample is changing across the surface. It is most likely something about the polyimide window itself rather than the coating because the transmission on the uncoated side of the window is also varying as y-position is changed. Because transmittance was up around 50% this small variation across the film did not affect our transmission measurements very much. However, this variation and the wavy nature of the film did affect our reflection measurements, both

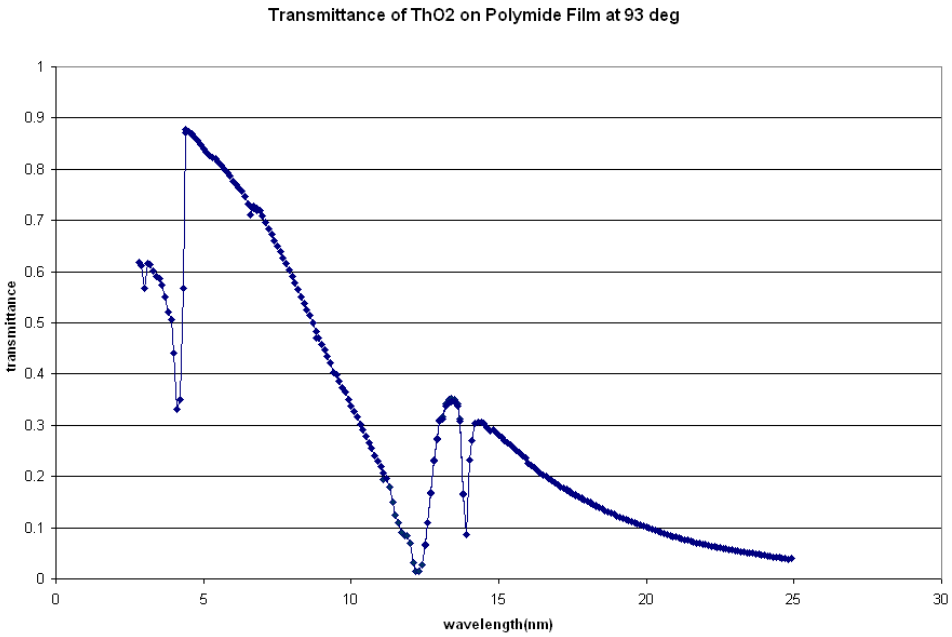

Figure 3.10: Transmission of ThO2 on polyimide film at 93 degrees by wavelength.

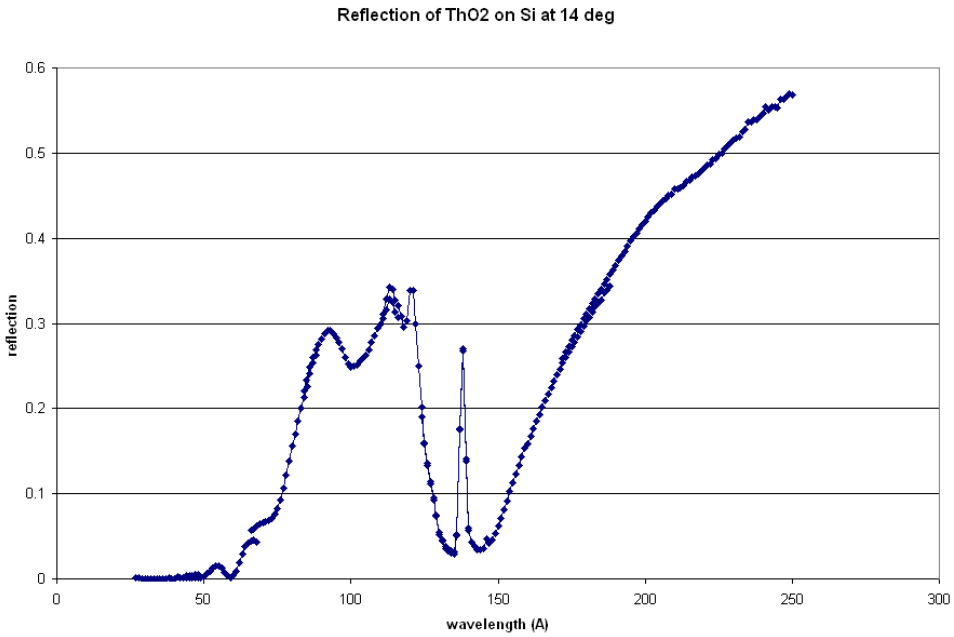

Figure 3.11: Reflection of ThO_2 on silicon by wavelength.

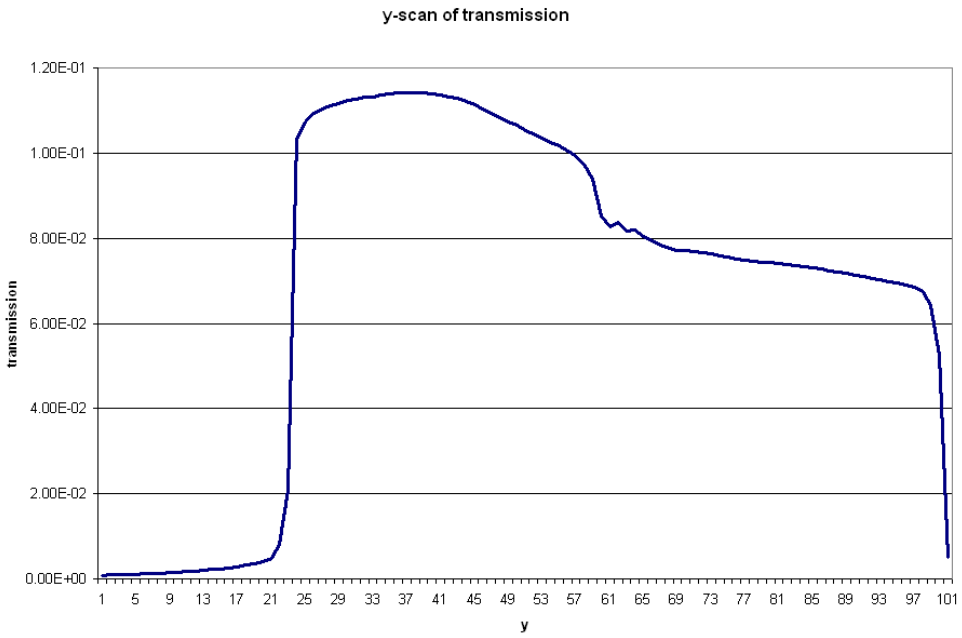

Figure 3.12: Transmission through our sample as a function of y position. The high transmission on the left side of the plot is through the uncoated polyimide window. The lower transmission on the right side of the plot is through the polyimide film coated with thorium oxide. There are no places on the plot where transmission is uniform as y is varied.

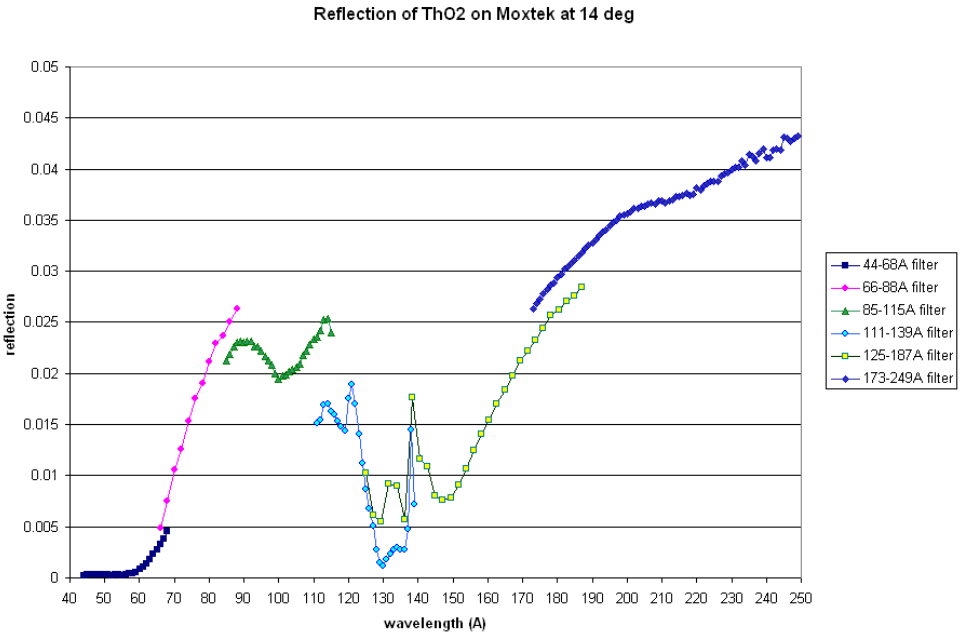

Figure 3.13: Reflection of ThO_2 on polyimide at 20 degrees as a function of wavelength. Each data set was taken with a different set of filters and order sorters in order to get the desired wavelength. In this plot, reflectance does not match up as the filter set was changed, showing us that we don't know what the data means.

because of our extremely small signal measurements and because waviness affects grazing measurements much more than normal-incidence measurements. Our small signals in reflection measurements were also affected significantly by the dark current, which was on the order of 5% of our total signal. The reflectance scans we took did not match up when we changed order sorter and filter sets (see Figure 3.13). We have had this problem before as was previously mentioned, but never on this scale. At some wavelengths, measurements disagree by as much as 32% of total reflectance. Because of this significant mismatch, we did not know what measurements to believe.

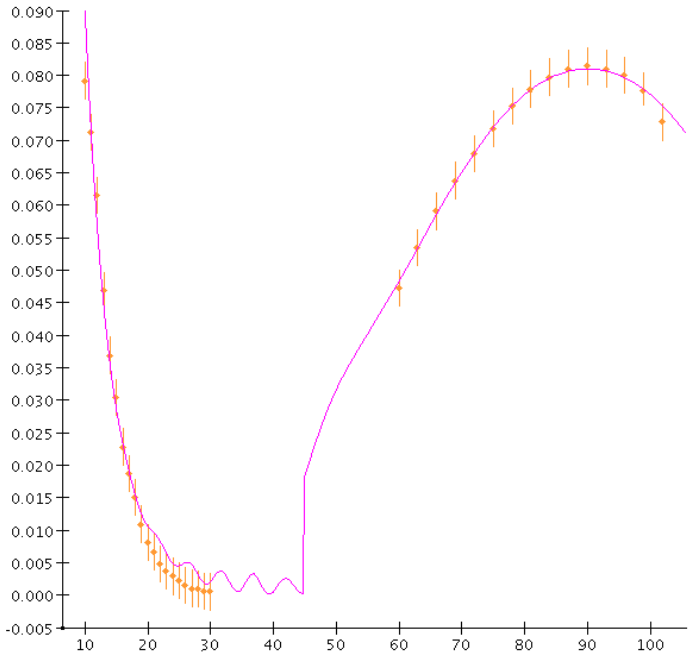

Figure 3.14: Reflection and transmission data for thorium oxide on a polyimide film at 210 Å fit simultaneously. The reflection data was not fit very well, probably due to the fact that reflectance of the film was very low, and the film was visibly warped and wavy.

3.3 Data Fitting

We did some simultaneous fitting of reflectance and transmittance of ThO_2 on the polyimide film in order to try and decrease the degrees of freedom in our fit. As can be seen in Figure 3.14 the transmittance data (the right side of the plot) is fit well, but the reflection data (the left side of the plot) is not. Because of the problems we had with the reflection data and because of the nature of the polyimide film, we decided to discard the reflectance data and just fit the transmittance data. We still had data from reflectance off of our sample on a silicon wafer and we could fit these two sets of data simultaneously, but this required modifications to our program that we never

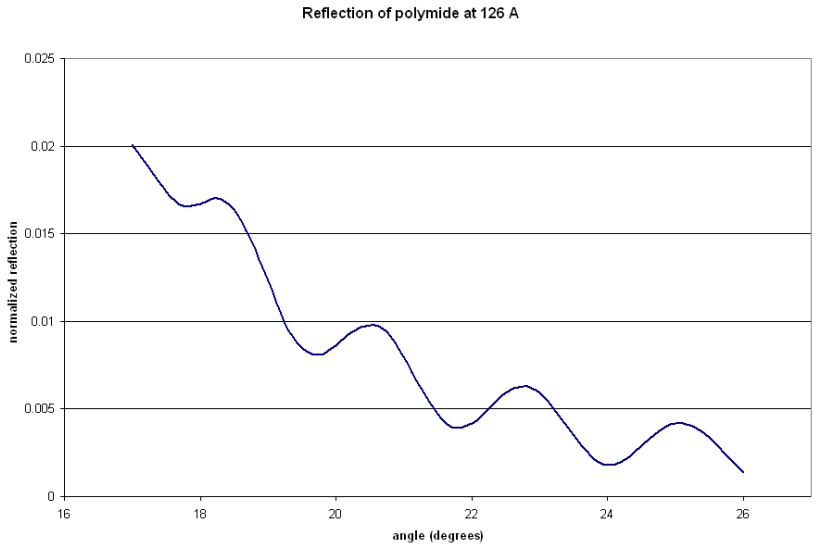

Figure 3.15: Reflection of polyimide at 126 Å as a function of sample angle. We used the fringes of interference from the back and front interfaces of the film to derive a thickness for the film.

implemented. This would be an interesting area for future research. Since the optical constants of polyimide are unknown in this wavelength region, our fitting procedure was to fit the transmission data for the uncoated film and use the constants found there in our fit of the coated data. To find the thickness of the polyimide film, we found reflectance data off of the film that contained well-defined interference fringes due to reflection off of the front and back surfaces of the film. We used the equation

$$m\lambda = 2d\sin(\theta_1 - \theta_2) \tag{3.1}$$

to find a thickness of 1603.7 Å. The thickness of the ThO_2 film was estimated to be around 200 Å when we sputtered it. We also took XRD data from our sample deposited on a silicon wafer at the same time as our transmission sample which we believe has the same thickness. From this measurement we obtained a thickness

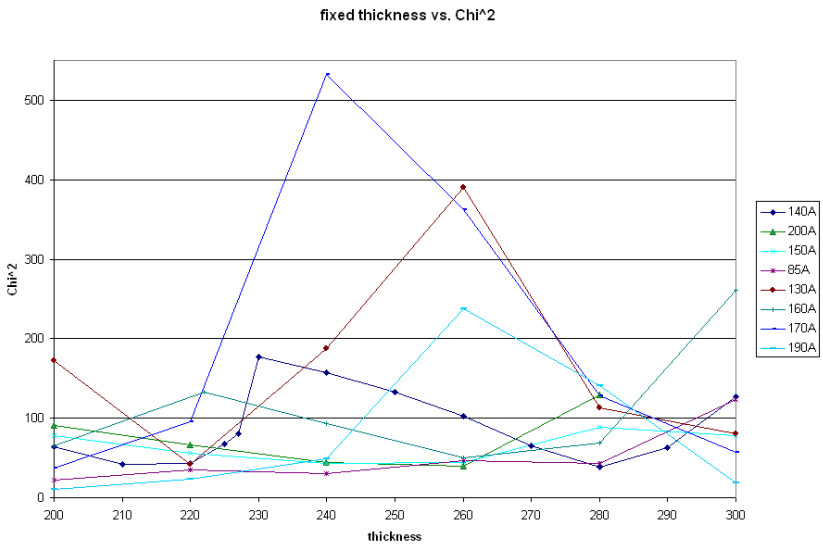

Figure 3.16: Plot of χ^2 vs. thickness. Two minima can be seen, one at 220 Å and one at 280 Å.

estimate of 220 Å ±20 Å. We fit several sets of data at thicknesses ranging from 200 Å to 300 Å and compared the χ^2 value of the fits as is shown in Figure 3.16. We found two consistent minimums in this data, one at a 220 Å thickness and one at a 280 Å thickness. Because of our XRD data, we chose to do our fits with a thickness of 220 Å. A representative fit of transmission data is shown in Figure 3.17. The optical constants derived from fits of transmittance are shown in figures 3.18 and 3.19. The values for δ given to us by these fits are all over the place and often have large error bars. The values for β look more reasonable. It makes sense that δ cannot be given very accurately from transmission data. The two places δ comes into play in transmission data is in the decrease in transmitted intensity due to reflection off the front surface of the film and the change in wavelength of light from that in vacuum in the film. In this case, the film reflects very little, especially compared to the amount

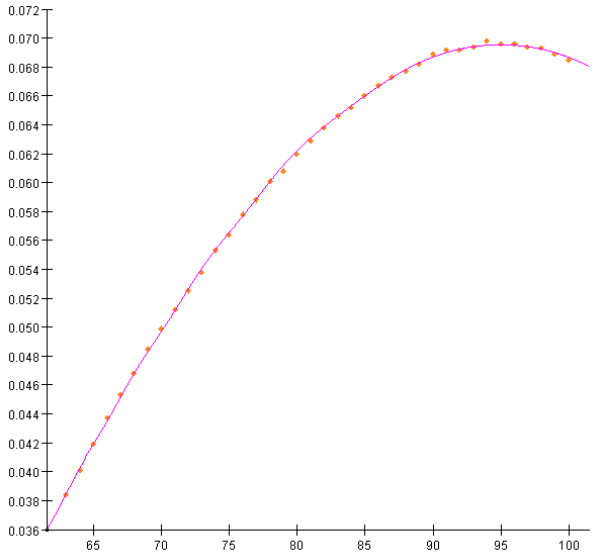

Figure 3.17: A representative fit of transmission data at 120 Å.

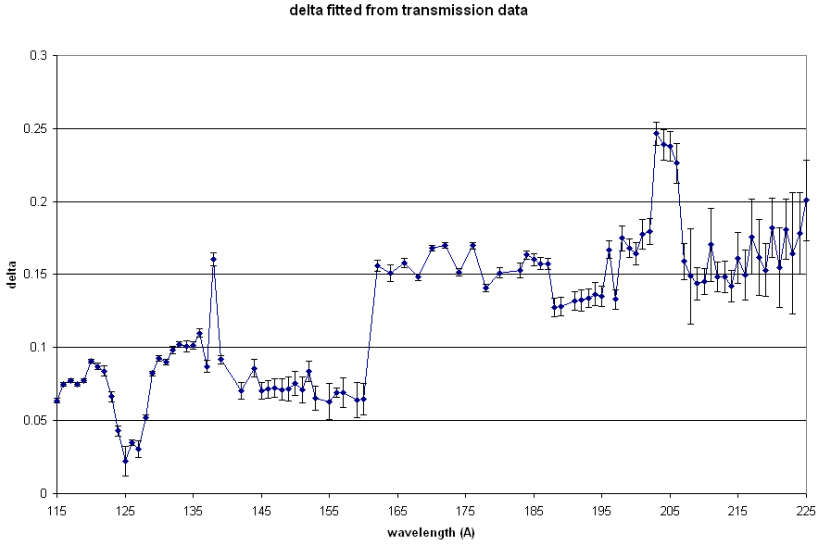

Figure 3.18: δ for thorium oxide fitted from transmission data from 115 Å to 225 Å.

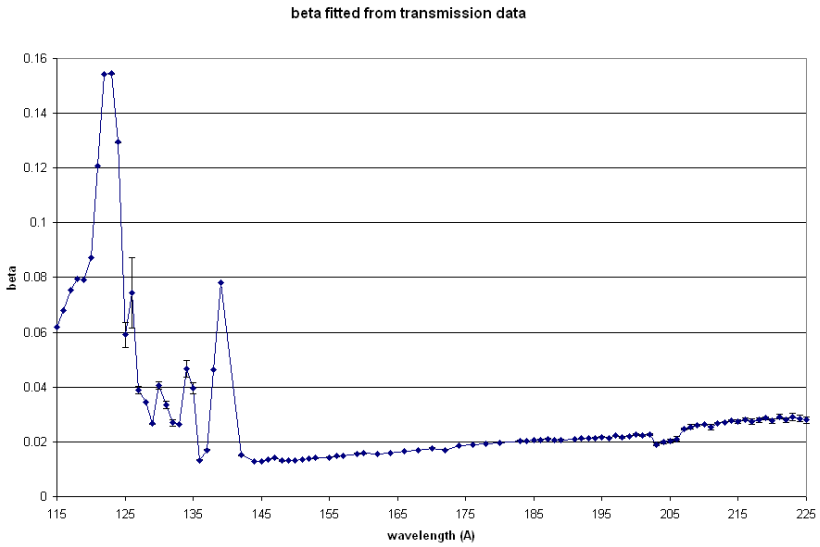

Figure 3.19: β for thorium oxide fitted from transmission data from 115 Å to 225 Å.

of light transmitted through the film at near-normal incidence. Also, δ is very small, making the amout of change in wavelength of light inside the film very small. Thus transmitttance measurements depend very little on the value of δ, and so δ cannot be derived very accurately from these measurements.

Fitting reflectance data from the sample deposited on silicon raised some interesting questions. Although we had decided on a 220 Å thickness because of fitting analysis and XRD data, some of the fits of reflection data were not very good at this thickness, but qualitatively better with a thickness of 280 Å (see figures 3.20 and 3.21). The second fit was not only qualitatively better, the χ^2 value of the fit went down by a factor of 10. A thickness of 280 Å is not altogether surprising, since in our χ^2 analysis of the transmission data we found minima at both 220 Å and 280 Å. However, XRD gave us a value close to 220 Å for the thickness of the film on silicon. We had estimated the error of this value to be $\pm$20 Å, not $\pm$60 Å. One possible reason

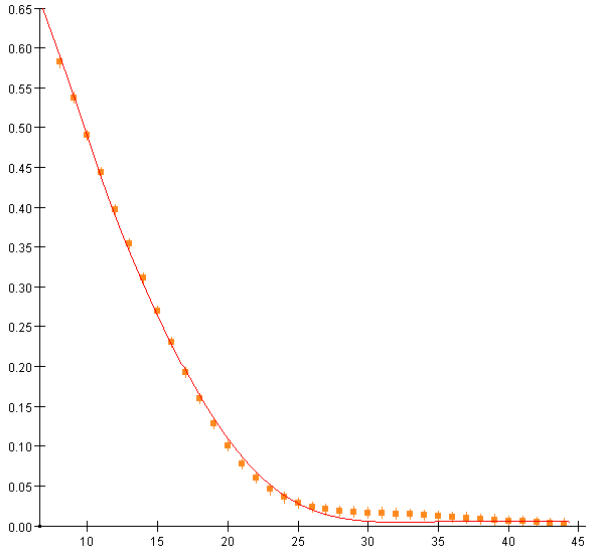

Figure 3.20: Fit of reflection of thorium oxide on silicon as a function of angle. This fit uses a thickness of 220 Å for the thorium oxide film. It is not a qualitatively good fit.

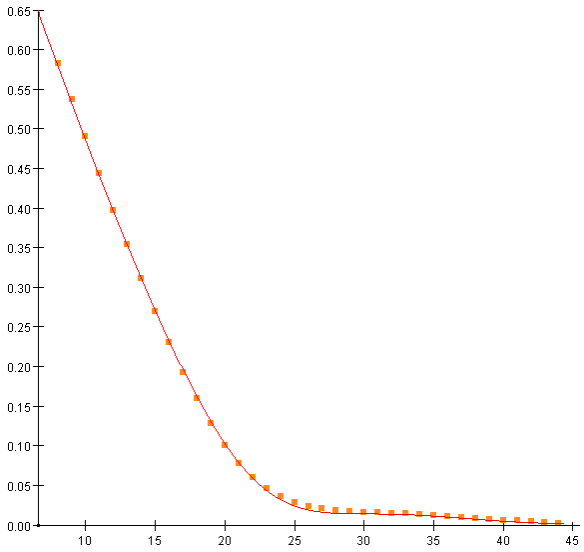

Figure 3.21: Fit of reflection of thorium oxide on silicon as a function of angle. This fit uses a thickness of 280 Å for the thorium oxide film. Qualitatively, it is a much better fit than the same data fit with a thickness of 220 Å for the thorium oxide film.

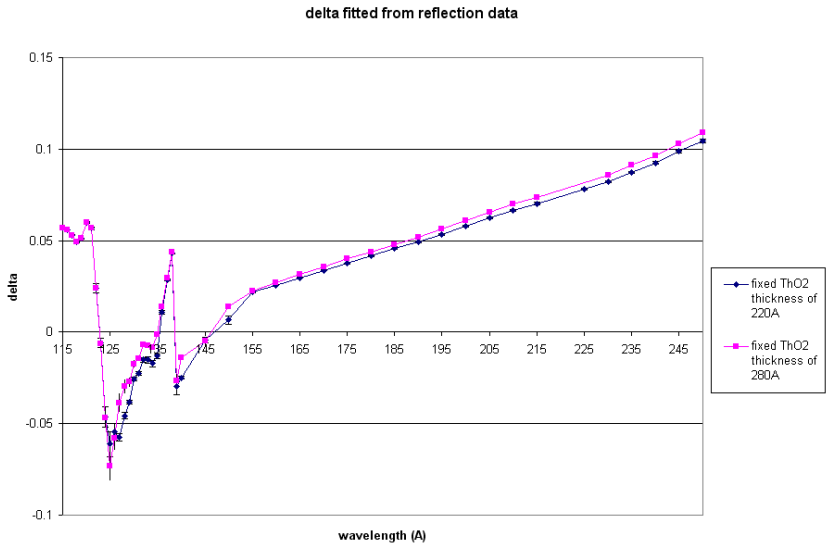

Figure 3.22: δ for thorium oxide fitted from reflectance data from 115 Å to 250 Å.

for this discrepancy is the effect of surface roughness on the film, which we failed to account for. Because of this discrepancy with the film thickness, we decided to fit the reflection data with fixed film thicknesses of 220 Å and 280 Å. The two fits gave us the values for δ and β shown in figures 3.22 and 3.23. As can be seen from Figure 3.22, the value of δ is not very dependent on the thickness of the layer. On average, the fit value of δ only changes by about 0.003 when the thickness of of the film is changed from 220 Å to 280 Å. This leads us to believe that for these wavelengths and these angles, our sample is optically thick. This is a very exciting conclusion because our questions concerning the real thickness of our sample do not affect our values for δ. Accordingly, as can be seen from Figure 3.23, the value of β does depend significantly on the film thickness. The value of β changes by about 0.006 when the thickness of the film is changed from 220 Å to 280 Å. This tells us that we cannot get a good value for β from reflection data unless we have a better knowledge of the

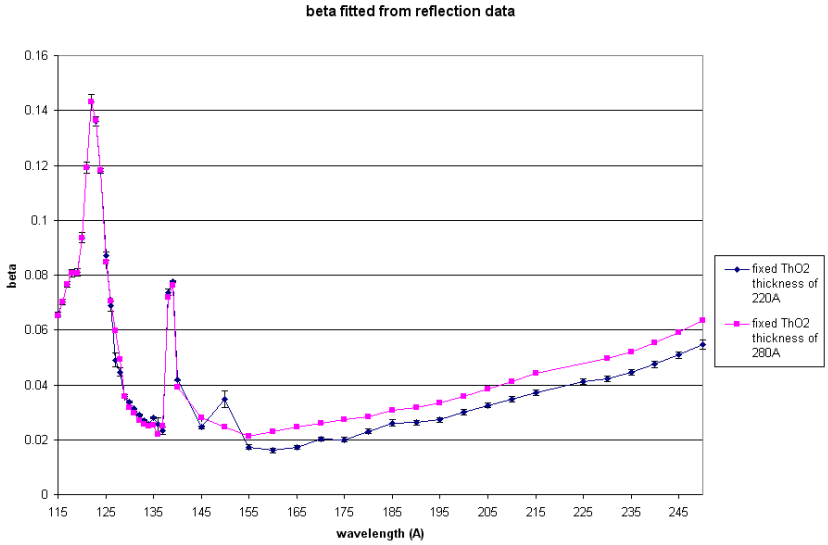

Figure 3.23: β for thorium oxide fitted from reflectance data from 115 Å to 250 Å.

thickness of the film.

Another concern with our data is that the values of δ and β fit from transmission and reflection data do not match up as can be seen in figures 3.24 and 3.25. Values of δ are dissimilar by as much as 0.2 and as little as 0.01. Values of β are dissimilar by as much as 0.06 and as little as 0.001. Because of the large error bars in δ fit with transmission and because it is difficult to obtain accurate values for δ with transmission data, we refit our transmission data, fixing δ to be those values we obtained by fitting reflection data. As can be seen from Figure 3.26, β values did not change significantly when we used fixed values for δ obtained from the fits of reflection data. This is significant because it tells us that δ values obtained from reflection data fits are not only independent of film thickness in reflection data, they are independent of β values in transmission data. This makes us very confident in the accuracy of the δ values obtained from reflection data fits.

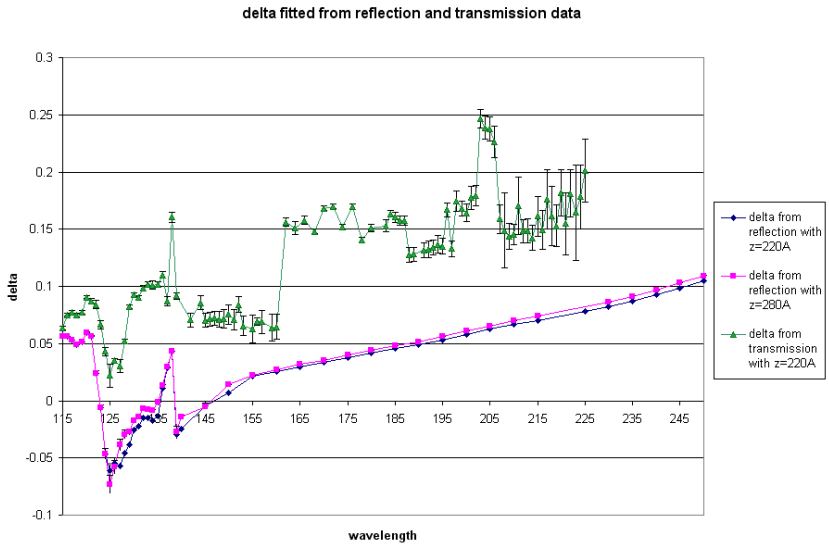

Figure 3.24: A comparison of δ values obtained by fitting reflection data and values obtained by fitting transmission data.

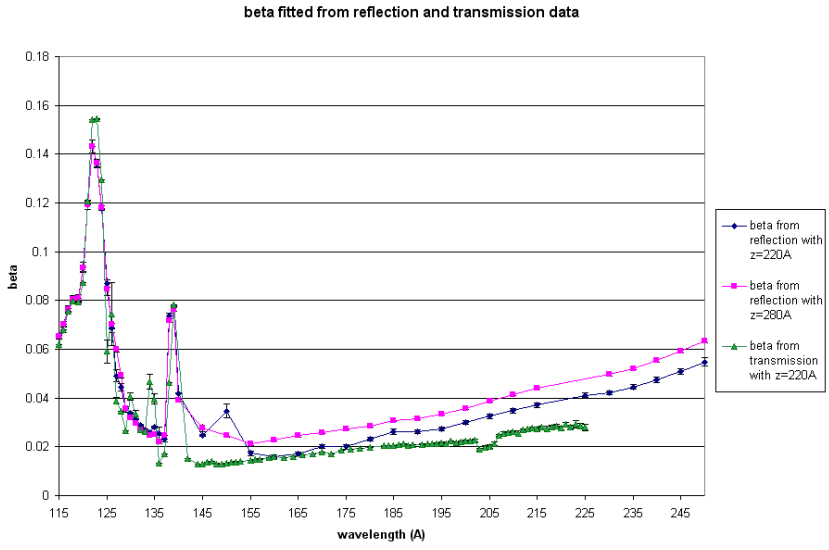

Figure 3.25: A comparison of β values obtained by fitting reflection data and values obtained by fitting transmission data.

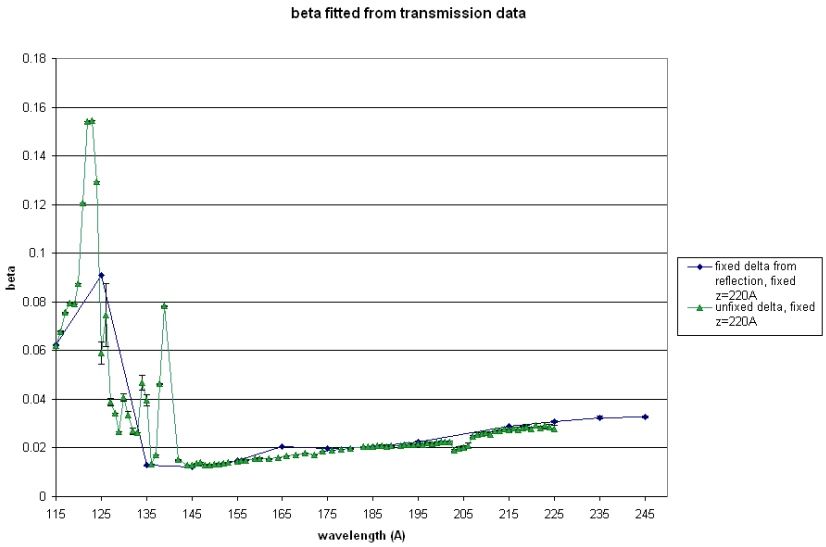

Figure 3.26: A comparison of β values obtained by fitting transmission data with an unfixed δ and then with a fixed δ obtained by fitting reflectance data.

However, since the β values did not change in the transmission data when we changed the δ values, we still have two sets of values for β, one obtained with reflection and one with transmission measurements (see Figure 3.23). The value of β also depends on the thickness of the film, which we don't know. We tried refitting our reflectance data with fixed values of β obtained from fitting transmittance data, but the fits were significantly worse. In most cases, an error bar on β of 0.02 would not make significant errors in predicting the reflectance of a multilayer. However, for thorium oxide we have found that δ and β are very similar in value. Reflectance of a boundary at normal incidence is given approximately by the equation

$$R \approx \frac{\delta^2 + \beta^2}{4} \tag{3.2}$$

If, for example, at a wavelength of 210 Å we have found δ to be 0.068. Transmittance fits give a β value of 0.026 while reflectance fits with a film thickness of 220 Å give

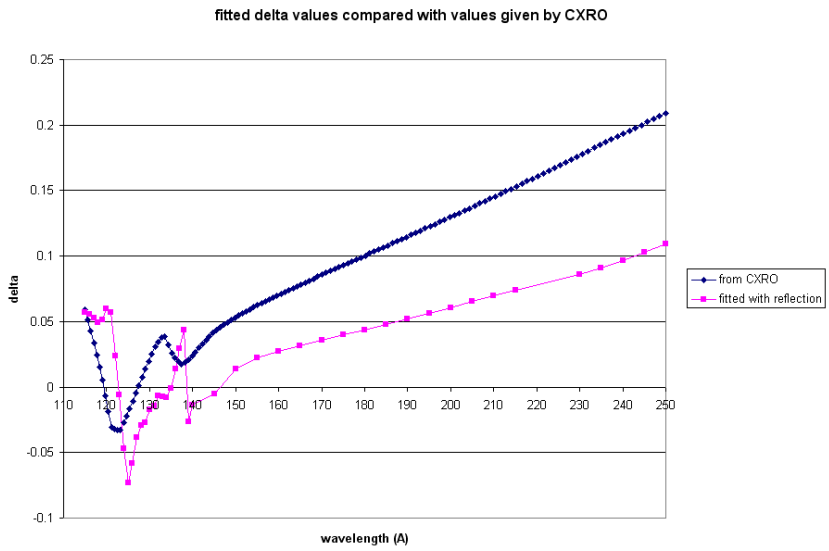

Figure 3.27: Fitted values of δ compared to those given by CXRO.

a β value of 0.035. Thus β fitted with transmittance gives a reflectance at normal incidence of approximately 0.13% while β fitted with reflectance gives a value of 0.15%.

Comparing our values for δ and β with those given by CXRO's website, we find that our values differ significantly. (see Figures 3.27 and 3.28). For δ, values are different by as much as 0.1. For β, values are different by as much as 0.05. The deviation in these two values could cause reflectance calculations to be different by as much as 0.7% at normal incidence.

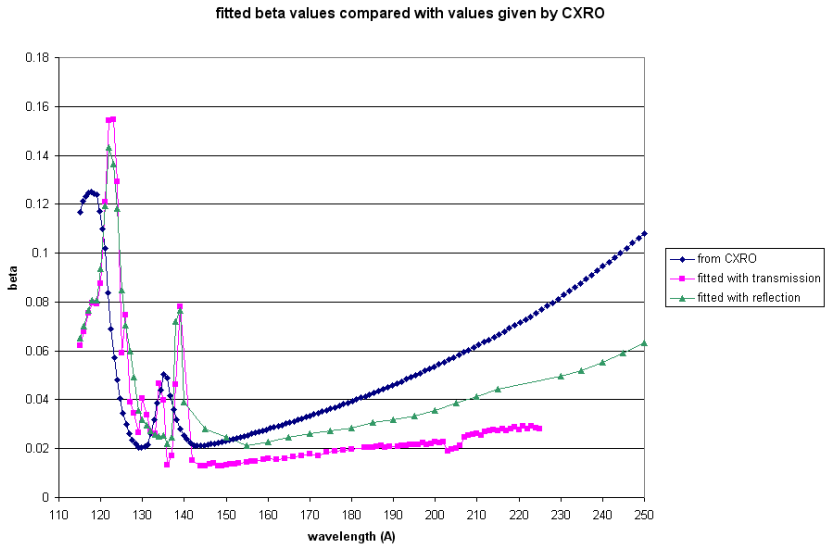

Figure 3.28: Fitted values of β compared to those given by CXRO.

Chapter 4

Conclusions

We have characterized the roughness of our sample and have attempted to account for this roughness using three methods. We have found that: our sputtered thorium has a naturally rough surface. Over a 1000x1000 Å area, RMS roughness is 36 Å. Over a 10,000x10,000 Å area, RMS roughness is 43 Å. In our sample, oxygen has interdiffused nearly linearly for 50 Å. The Debye-Waller factor corrects our data well for low angles. The Nevot-Croce factor corrects our data well for low and high angles. The finite differences approximation corrects our data very well for low and middle angles. We conclude that the best way to account for roughness in our program is to employ a combination of the Nevot-Croce factor and the finite distances approximation.

We have found values for δ for our reactively sputtered thorium dioxide in the region from $115 - 250$ Å. We have found that we cannot find β values that we believe in this region because reflectance and transmittance measurements give us two different and irreconcilable sets of data. We are unsure at this point why the two measurements give us different answers. We hope that future research (depositing on photodiodes) will help resolve this issue. We believe our values for δ despite this problem because they are relatively independent of thickness in reflection data and

relatively independent of β in transmission data. Since δ and β are so close together in this region, our value for β makes a significant difference in predicting how thorium oxide will reflect. For this reason we will continue to research this material. We have also found that our fit values for delta and beta are significantly different than those reported by CXRO. Our values are different by as much as 0.1 in the case of delta and 0.05 in the case of beta. The discrepancies in these values could change calculated reflectances by as much as 0.7% at normal incidence.

Appendix A

MATFIT Source Code

This is the source code for the program used to fit our data. The program calls a library called Minuit developed at CERN that minimizes the function. FitFrame is the main class. The class mirror is composed of layers that define the properties of each material. The properties of each material are defined as FitParameters that can either be free, fixed, or constrained to be within a minimum and a maximum value. The function is defined in refl and the fitting is done in MirrorFunc. The pieces I worked on are the refltrans, Am, and kz methods in refl, the Matrix class, and the OffsetDialog class.

A.1 FitFrame.class

```
/*
 * FitFrame.java
 *
 * Created on December 30, 2003, 12:46 PM
 *
 * Change History:
 * Version 1.0:  Initial debugged release used at ALS 2/12/04 to
 * 2/15/04
 * Version 1.1:  Allow various polarizations
 *      Retain memory of directories from lookup
 */
```

```
/**
 *
 * @author  turley
 */

import javax.swing.ImageIcon; import javax.swing.JOptionPane;
import javax.swing.JRadioButton; import javax.swing.JFileChooser;
import javax.swing.UIManager;
// import org.freehep.swing.JDirectoryChooser;
import java.util.LinkedList; import java.io.*;

public class FitFrame extends javax.swing.JFrame {

    /** optical element to be fit */
    private Mirror optic = new Mirror();

    private int iButton=0;
    private int iElement=0; // element attached to pop-up menu
    private LinkedList buttonList=new LinkedList();
    private ReflFit rfit=null;
    public static javax.swing.JFrame MainFrame=null;
    private static File loadFile=null;

    /** Creates new form FitFrame */
    public FitFrame() {
        javax.swing.ImageIcon icon = new
        ImageIcon("matfitIcon.GIF");
        setIconImage(icon.getImage());
        setTitle("matfit 1.0");
        initComponents();
        initOptic(); // Add private components for Mirror
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this
     * method is always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        buttonGroup1 = new javax.swing.ButtonGroup();
        LayerPopupMenu = new javax.swing.JPopupMenu();
```

```
deleteLayerMenuItem = new javax.swing.JMenuItem();
addLayerAboveMenuItem = new javax.swing.JMenuItem();
addLayerBelowMenuItem = new javax.swing.JMenuItem();
editLayerMenuItem = new javax.swing.JMenuItem();
cancelLayerMenuItem = new javax.swing.JMenuItem();
MenuBar = new javax.swing.JMenuBar();
FileMenu = new javax.swing.JMenu();
ReadMenuItem = new javax.swing.JMenuItem();
CreateMenuItem = new javax.swing.JMenuItem();
jSeparator1 = new javax.swing.JSeparator();
FitMenuItem = new javax.swing.JMenuItem();
ScaleMenuItem = new javax.swing.JMenuItem();
jSeparator2 = new javax.swing.JSeparator();
OffsetMenuItem = new javax.swing.JMenuItem();
jSeparator3 = new javax.swing.JSeparator();
ExitMenuItem = new javax.swing.JMenuItem();
StackMenu = new javax.swing.JMenu();
LoadMenuItem = new javax.swing.JMenuItem();
SaveMenuItem = new javax.swing.JMenuItem();
HelpMenu = new javax.swing.JMenu();
ProgramHelpMenuItem = new javax.swing.JMenuItem();
AboutMenuItem = new javax.swing.JMenuItem();

deleteLayerMenuItem.setText("delete layer");
deleteLayerMenuItem.addActionListener
    (new java.awt.event.ActionListener() {
    public void actionPerformed(
        java.awt.event.ActionEvent evt) {
            deleteLayerMenuItemActionPerformed(evt);
    }
});

LayerPopupMenu.add(deleteLayerMenuItem);

addLayerAboveMenuItem.setText("add layer above");
addLayerAboveMenuItem.addActionListener
    (new java.awt.event.ActionListener() {
    public void actionPerformed(
        java.awt.event.ActionEvent evt) {
            addLayerAboveMenuItemActionPerformed(evt);
    }
});
```

```
        LayerPopupMenu.add(addLayerAboveMenuItem);

        addLayerBelowMenuItem.setText("add layer below");
        addLayerBelowMenuItem.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                    addLayerBelowMenuItemActionPerformed(evt);
            }
        });

        LayerPopupMenu.add(addLayerBelowMenuItem);

        editLayerMenuItem.setText("edit layer");
        editLayerMenuItem.addActionListener
            (new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                    editLayerMenuItemActionPerformed(evt);
            }
        });

        LayerPopupMenu.add(editLayerMenuItem);

        cancelLayerMenuItem.setText("cancel");
        LayerPopupMenu.add(cancelLayerMenuItem);

        getContentPane().setLayout(new java.awt.GridLayout(6, 1));

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(
                java.awt.event.WindowEvent evt) {
                    exitForm(evt);
            }
        });

        FileMenu.setText("File");
        ReadMenuItem.setText("Read Reflectance Data...");
        ReadMenuItem.addActionListener
            (new java.awt.event.ActionListener() {
            public void actionPerformed(
```

```
            java.awt.event.ActionEvent evt) {
                ReadMenuItemActionPerformed(evt);
        }
    });

    FileMenu.add(ReadMenuItem);

    CreateMenuItem.setText("Create Reflectance Data...");
    CreateMenuItem.addActionListener
        (new java.awt.event.ActionListener() {
        public void actionPerformed(
            java.awt.event.ActionEvent evt) {
                CreateMenuItemActionPerformed(evt);
        }
    });

    FileMenu.add(CreateMenuItem);

    FileMenu.add(jSeparator1);

    FitMenuItem.setText("Fit");
    FitMenuItem.addActionListener
        (new java.awt.event.ActionListener() {
        public void actionPerformed(
            java.awt.event.ActionEvent evt) {
                FitMenuItemActionPerformed(evt);
        }
    });

    FileMenu.add(FitMenuItem);

    ScaleMenuItem.setText("Scale Error...");
    ScaleMenuItem.addActionListener
        (new java.awt.event.ActionListener() {
        public void actionPerformed(
            java.awt.event.ActionEvent evt) {
                ScaleMenuItemActionPerformed(evt);
        }
    });

    FileMenu.add(ScaleMenuItem);
```

```
        FileMenu.add(jSeparator2);

        OffsetMenuItem.setText("Add Offset");
        OffsetMenuItem.addActionListener
            (new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                    OffsetMenuItemActionPerformed(evt);
            }
        });

        FileMenu.add(OffsetMenuItem);

        FileMenu.add(jSeparator3);

        ExitMenuItem.setText("Exit");
        ExitMenuItem.addActionListener
            (new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                    ExitMenuItemActionPerformed(evt);
            }
        });

        FileMenu.add(ExitMenuItem);

        MenuBar.add(FileMenu);

        StackMenu.setText("Stack");
        LoadMenuItem.setText("Load...");
        LoadMenuItem.addActionListener
            (new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                    LoadMenuItemActionPerformed(evt);
            }
        });
        StackMenu.add(LoadMenuItem);

        SaveMenuItem.setText("Save...");
        SaveMenuItem.addActionListener
            (new java.awt.event.ActionListener() {
```

```
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                    SaveMenuItemActionPerformed(evt);
            }
        });
        StackMenu.add(SaveMenuItem);

        MenuBar.add(StackMenu);

        HelpMenu.setText("Help");
        ProgramHelpMenuItem.setText("Program Help");
        HelpMenu.add(ProgramHelpMenuItem);

        AboutMenuItem.setText("About");
        AboutMenuItem.addActionListener
            (new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                    AboutMenuItemActionPerformed(evt);
            }
        });

        HelpMenu.add(AboutMenuItem);

        MenuBar.add(HelpMenu);

        setJMenuBar(MenuBar);

        java.awt.Dimension screenSize =
        java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-440)/2,
        (screenSize.height-256)/2, 440, 256);
    }//GEN-END:initComponents

    private void addLayerBelowMenuItemActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_addLayerBelowMenuItemActionPerformed
        Film layer=new Film(1,0,"material",0);
        EditLayerDialog dialog=new EditLayerDialog(
            this, true, layer);
        dialog.show();
        if (dialog.getReturnStatus()==EditLayerDialog.RET_OK){
```

```
            // Create layer in mirror
            optic.addFilmAt(layer, iElement);
            // doesn't count vacuum
            updateOptic();
        }
    }//GEN-LAST:event_addLayerBelowMenuItemActionPerformed

    private void addLayerAboveMenuItemActionPerformed
        (java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_addLayerAboveMenuItemActionPerformed
        // Add your handling code here:
        // First get the information for the layer
        Film layer=new Film(1,0,"material",0);
        EditLayerDialog dialog=new EditLayerDialog(
            this, true, layer);
        dialog.show();
        if (dialog.getReturnStatus()==EditLayerDialog.RET_OK){
            // Create layer in mirror
            optic.addFilmAt(layer, iElement-1);
            // doesn't count vacuum
            updateOptic();
        }
    }//GEN-LAST:event_addLayerAboveMenuItemActionPerformed

    private void deleteLayerMenuItemActionPerformed
        (java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_deleteLayerMenuItemActionPerformed
        // Add your handling code here:
        // This is irreversible;  better ask for a confirmation
        String name=optic.getFilm(iElement-1).name;
        int status=JOptionPane.showConfirmDialog(
            this, "Okay to delete layer <"
            +name+">?","Confirm Delete",
                JOptionPane.YES_NO_OPTION);
        // Will return 0 for "yes" and 1 for "no"
        if (status==0){
            optic.removeFilm(iElement-1);
            updateOptic();
        }
        /*
         * Because I have shortened the list of buttons on
         * the screen, I'll need to repaint
```

```
     */
}//GEN-LAST:event_deleteLayerMenuItemActionPerformed

/** Action event called when editing a layer
 * @param evt
 */
private void editLayerMenuItemActionPerformed(
    java.awt.event.ActionEvent evt)
    {//GEN-FIRST:event_editLayerMenuItemActionPerformed
    // Add your handling code here:
    /*
     * Part of the fun here is going to be figuring
     * out what menu item I am on.  An ugly way would be
     * to do it with a global variable. A much nicer
     * way would be to embed the menu item in the event
     * somehow, but I can't figure out how to do that.
     */
    if(iElement==0){
        // vacuum layer
        new EditLayerDialog(this, true, optic.vacuum).show();
        ((JRadioButton)buttonList.get(iElement)).setText(
            optic.vacuum.name);
    } else if (iElement>=optic.numFilms()-1) {
        // substrate layer
        new EditLayerDialog(this, true, optic.substrate)
            .show();
        ((JRadioButton)buttonList.get(iElement)).setText(
            optic.substrate.name);
    } else {
        // thin film layer
        new EditLayerDialog(this, true,
            optic.getFilm(iElement-1)).show();
        ((JRadioButton)buttonList.get(iElement)).setText(
            optic.getFilm(iElement-1).name);
    }
}//GEN-LAST:event_editLayerMenuItemActionPerformed

private void buttonActionPerformed(int element){
    /*
     * Give user a choice to:
     *  delete this layer (unless vacuum or substrate)
     *  add a layer above (unless vacuum)
```

```
         *  add a layer below (unless substrate)
         *  edit layer
         *  cancel
         */
        // JOptionPane.showMessageDialog(this,
            "Selected layer "+element);
        // Check which options to disable
        if(element==0){
            deleteLayerMenuItem.setEnabled(false);
            addLayerAboveMenuItem.setEnabled(false);
            addLayerBelowMenuItem.setEnabled(true);
        } else if (element==optic.numFilms()-1) {
            deleteLayerMenuItem.setEnabled(false);
            addLayerAboveMenuItem.setEnabled(true);
            addLayerBelowMenuItem.setEnabled(false);
        } else {
            deleteLayerMenuItem.setEnabled(true);
            addLayerAboveMenuItem.setEnabled(true);
            addLayerBelowMenuItem.setEnabled(true);
        }
        iElement=element; // keep track of which element
                          // the menu is attached to
        LayerPopupMenu.show(this,150,50);
    }

    private void FitMenuItemActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_FitMenuItemActionPerformed
        // ReflFit.main(null);
        // If rfit is initialized too soon, this won't work
        if(rfit==null){
            // Fitter is not initialized
            //(probably because I didn't read in data yet)
            JOptionPane.showMessageDialog(this,
                "No data read in yet",
                    "No Data",JOptionPane.ERROR_MESSAGE);
            return;
        }
        // hep.aida.ref.fitter.FitResult result=rfit.fit();
        hep.aida.ref.fitter.FitResult result=rfit.fit(optic);
        rfit.plot(result);
        if(rfit.print(result, this)){
```

```
            rfit.updateStack(result, optic);
        }
    }//GEN-LAST:event_FitMenuItemActionPerformed

    private void ScaleMenuItemActionPerformed(
        java.awt.event.ActionEvent evt) {
        if(rfit==null){
            // Fitter is not initialized
            //(probably because I didn't read in data yet)
            JOptionPane.showMessageDialog(this,
                "No data read in yet","No Data",
                    JOptionPane.ERROR_MESSAGE);
            return;
        }
        String scale=JOptionPane.showInputDialog(this,
            "Scale Factor");
        double factor=Double.parseDouble(scale);
        rfit.scaleErrors(factor);
    }

    private void AboutMenuItemActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_AboutMenuItemActionPerformed
        String[] mList=new String[4];
        mList[0]="matfit: XUV Reflectance Fitting";
        mList[1]="Version 1.0";
        mList[2]="Niki Farnsworth";
        mList[3]="Copyright 2004";
        JOptionPane.showMessageDialog(this, mList,"About",
            JOptionPane.INFORMATION_MESSAGE);
    }//GEN-LAST:event_AboutMenuItemActionPerformed

    private void SaveMenuItemActionPerformed(
        java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        /*
        // Note: source for ExampleFileFilter can be
        // found in FileChooserDemo, under the demo/jfc
        // directory in the Java 2 SDK, Standard Edition.
        ExampleFileFilter filter = new ExampleFileFilter();
        filter.addExtension("jpg");
        filter.addExtension("gif");
```

```
        filter.setDescription("JPG & GIF Images");
        chooser.setFileFilter(filter);
        */
        chooser.setDialogTitle("Save Stack File");
        if(loadFile != null) {
            chooser.setCurrentDirectory(loadFile.getParentFile());
        }
        int returnVal = chooser.showSaveDialog(this);
        if(returnVal == JFileChooser.APPROVE_OPTION) {
            loadFile = chooser.getSelectedFile();
            // Write stack data to a file
            try{
                ObjectOutputStream fout = new ObjectOutputStream(
                    new FileOutputStream(loadFile));
                fout.writeObject(optic);
            } catch (Exception e) {
                System.err.println("Exception caught: " +
                    e.getMessage());
                e.printStackTrace();
            }
        }

    }

    private void LoadMenuItemActionPerformed(
        java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        /*
        // Note: source for ExampleFileFilter can be found in
        // FileChooserDemo, under the demo/jfc
        //  directory in the Java 2 SDK, Standard Edition.
        ExampleFileFilter filter = new ExampleFileFilter();
        filter.addExtension("jpg");
        filter.addExtension("gif");
        filter.setDescription("JPG & GIF Images");
        chooser.setFileFilter(filter);
        */
        chooser.setDialogTitle("Load Stack File");
        if(loadFile != null) {
            chooser.setCurrentDirectory(loadFile.getParentFile());
        }
        int returnVal = chooser.showOpenDialog(this);
```

```
        if(returnVal == JFileChooser.APPROVE_OPTION) {
            loadFile = chooser.getSelectedFile();
            try{
                ObjectInputStream fin = new ObjectInputStream(
                    new FileInputStream(loadFile));
                optic=(Mirror)fin.readObject();
            } catch (Exception e) {
                System.err.println("Exception caught: " +
                    e.getMessage());
                e.printStackTrace();
            }
            updateOptic(); // update display
        }
    }

    private void ExitMenuItemActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_ExitMenuItemActionPerformed
        // Add your handling code here:
        System.exit(0);
    }//GEN-LAST:event_ExitMenuItemActionPerformed

    private void ReadMenuItemActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_ReadMenuItemActionPerformed
        if(rfit == null){
            rfit=new ReflFit();
            // Set up class for fitting and storing data
        }
        JFileChooser chooser = new JFileChooser();
        if(loadFile != null) {
            chooser.setCurrentDirectory(loadFile.getParentFile());
        }
        /*
        // Note: source for ExampleFileFilter can be found
        // in FileChooserDemo, under the demo/jfc directory
        // in the Java 2 SDK, Standard Edition.
        ExampleFileFilter filter = new ExampleFileFilter();
        filter.addExtension("jpg");
        filter.addExtension("gif");
        filter.setDescription("JPG & GIF Images");
        chooser.setFileFilter(filter);
```

```
        */
        chooser.setDialogTitle("Open Reflectance File");
        int returnVal = chooser.showOpenDialog(this);
        if(returnVal == JFileChooser.APPROVE_OPTION) {
            // User chose a file she likes
            loadFile = chooser.getSelectedFile();
            // Get error information
            (new ErrorDialog(this, true)).show();
            rfit.data(chooser.getSelectedFile(),
                ErrorDialog.error, ErrorDialog.absolute,
                    ErrorDialog.ref);
        }
    }//GEN-LAST:event_ReadMenuItemActionPerformed

    private void CreateMenuItemActionPerformed(
        java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        chooser.setDialogTitle("Create Reflectance File");
        int returnVal = chooser.showSaveDialog(this);
        if(returnVal == JFileChooser.APPROVE_OPTION) {
            if(rfit==null){
                rfit=new ReflFit();
            }
            rfit.createData(chooser.getSelectedFile());
        }
    } // CreateMenuItemActionPerformed

    private void OffsetMenuItemActionPerformed(
        java.awt.event.ActionEvent evt) {
        (new OffsetDialog(this, true)).show();
    }

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt)
    {//GEN-FIRST:event_exitForm
        System.exit(0);
    }//GEN-LAST:event_exitForm

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
```

```
        // Set Operating System as Look and Feel (usually XP)
        try {
            UIManager.setLookAndFeel(UIManager.
                getSystemLookAndFeelClassName());
        } catch (Exception e) {
            System.err.println("Exception caught: " +
                e.getMessage());
        }
        MainFrame = new FitFrame();
        MainFrame.show();
    }

    /** displays current mirror elements on screen */
    private void initOptic() {
        int nButtons=optic.numFilms();
        // includes substrate and vacuum
        // Set the Grid Layout to the right size
        getContentPane().setLayout(new java.awt.GridLayout(
            Math.max(nButtons,6), 1));
        /*
         * Add vacuum layer button
         */
        JRadioButton button=new JRadioButton();
        button.setText(optic.vacuum.name);
        button.addActionListener(new layerListener(0));
        getContentPane().add(button);
        buttonGroup1.add(button);
        buttonList.add(button);
        /*
         * Add the films
         */
        for(int i=1; i<nButtons-1; i++){
            iButton=i;
            button=new JRadioButton();
            button.setText(optic.getFilm(i-1).name);
            button.addActionListener(new layerListener(i));
            getContentPane().add(button);
            buttonGroup1.add(button);
            buttonList.add(button);
        }
        /*
         * Add the substrate button
```

```
         */
        button=new JRadioButton();
        button.setText(optic.substrate.name);
        button.addActionListener(new layerListener(nButtons-1));
        getContentPane().add(button);
        buttonGroup1.add(button);
        buttonList.add(button);
    }

    /** Clear the current buttons from the screen and redisplay
     * the screen.
     */
    public void updateOptic() {
        java.util.ListIterator bIterator=buttonList
            .listIterator(0);
        JRadioButton button=null;
        while(bIterator.hasNext()){
            button=(JRadioButton)bIterator.next();
            // now remove all of the action listeners
            java.awt.event.ActionListener[] l=button
                .getActionListeners();
            for(int i=0; i<l.length; i++){
                button.removeActionListener(l[i]);
            }
            buttonGroup1.remove(button);
            button.setVisible(false);
            // make sure it really disappears
            getContentPane().remove(button);
            // remove from pane
        }
        // remove the buttons in the list
        buttonList.clear();
        // initialize the screen again
        initOptic();
        getContentPane().validate();
        // needed to resize things for layout manager
    }

    class layerListener implements java.awt.event
        .ActionListener {

        /** Layer number that created this action listener */
```

```
        private int layerNumber=0;

        public layerListener(int element) {
            layerNumber=element;
        }

        public void actionPerformed(
            java.awt.event.ActionEvent evt) {
            buttonActionPerformed(layerNumber);
        }

    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JMenuItem AboutMenuItem;
    private javax.swing.JMenuItem CreateMenuItem;
    private javax.swing.JMenuItem OffsetMenuItem;
    private javax.swing.JMenuItem ExitMenuItem;
    private javax.swing.JMenu FileMenu;
    private javax.swing.JMenuItem FitMenuItem;
    private javax.swing.JMenuItem ScaleMenuItem;
    private javax.swing.JMenu HelpMenu;
    private javax.swing.JPopupMenu LayerPopupMenu;
    private javax.swing.JMenuItem LoadMenuItem;
    private javax.swing.JMenuBar MenuBar;
    private javax.swing.JMenuItem ProgramHelpMenuItem;
    private javax.swing.JMenuItem ReadMenuItem;
    private javax.swing.JMenuItem SaveMenuItem;
    private javax.swing.JMenu StackMenu;
    private javax.swing.JMenuItem addLayerAboveMenuItem;
    private javax.swing.JMenuItem addLayerBelowMenuItem;
    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.JMenuItem cancelLayerMenuItem;
    private javax.swing.JMenuItem deleteLayerMenuItem;
    private javax.swing.JMenuItem editLayerMenuItem;
    private javax.swing.JMenuItem interfaceMenuItem;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JSeparator jSeparator3;
    // End of variables declaration//GEN-END:variables

}
```

A.2 ErrorDialog.class

```
/*
 * Dialog.java
 *
 * Created on May 11, 2004, 3:56 PM
 */

/**
 *
 * @author  farnsworth
 */
public class ErrorDialog extends javax.swing.JDialog {
    /** A return status code - returned if Cancel button has
    been pressed */
    public static final int RET_CANCEL = 0;
    /** A return status code - returned if OK button has
    been pressed */
    public static final int RET_OK = 1;

    /** Creates new form Dialog */
    public ErrorDialog(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }

    /** @return the return status of this dialog - one of
    RET_OK or RET_CANCEL */
    public int getReturnStatus() {
        return returnStatus;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this
     *  method is always regenerated by the Form Editor.
     */
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        buttonGroup1 = new javax.swing.ButtonGroup();
        buttonGroup2 = new javax.swing.ButtonGroup();
        buttonPanel = new javax.swing.JPanel();
```

```
okButton = new javax.swing.JButton();
cancelButton = new javax.swing.JButton();
jPanel1 = new javax.swing.JPanel();
jRadioButton1 = new javax.swing.JRadioButton();
jRadioButton2 = new javax.swing.JRadioButton();
jRadioButton3 = new javax.swing.JRadioButton();
jRadioButton4 = new javax.swing.JRadioButton();
jLabel1 = new javax.swing.JLabel();
jTextField1 = new javax.swing.JTextField();

setTitle("Set Error");
setName("ErrDialog");
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(
        java.awt.event.WindowEvent evt) {
        closeDialog(evt);
    }
});

buttonPanel.setLayout(new java.awt.FlowLayout(
    java.awt.FlowLayout.RIGHT));

okButton.setText("OK");
okButton.addActionListener(new java.awt.event.
    ActionListener() {
    public void actionPerformed(java.awt.event.
        ActionEvent evt) {
        okButtonActionPerformed(evt);
    }
});

buttonPanel.add(okButton);

cancelButton.setText("Cancel");
cancelButton.addActionListener(
    new java.awt.event.ActionListener() {
    public void actionPerformed(
        java.awt.event.ActionEvent evt) {
        cancelButtonActionPerformed(evt);
    }
});
```

```
buttonPanel.add(cancelButton);

getContentPane().add(buttonPanel,
    java.awt.BorderLayout.SOUTH);

jPanel1.setLayout(new java.awt.GridBagLayout());

jRadioButton1.setSelected(true);
jRadioButton1.setText("absolute");
buttonGroup1.add(jRadioButton1);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.insets = new java.awt.
    Insets(20, 17, 0, 0);
gridBagConstraints.anchor =
    java.awt.GridBagConstraints.WEST;
jPanel1.add(jRadioButton1, gridBagConstraints);

jRadioButton2.setText("relative");
buttonGroup1.add(jRadioButton2);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.insets = new java.awt.
    Insets(19, 21, 0, 7);
gridBagConstraints.anchor =
    java.awt.GridBagConstraints.WEST;
jPanel1.add(jRadioButton2, gridBagConstraints);

jRadioButton3.setSelected(true);
jRadioButton3.setText("reflectance");
buttonGroup2.add(jRadioButton3);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.insets = new java.awt.
    Insets(22, 17, 7, 0);
gridBagConstraints.anchor =
    java.awt.GridBagConstraints.WEST;
jPanel1.add(jRadioButton3, gridBagConstraints);
```

```
        jRadioButton4.setText("transmittance");
        buttonGroup2.add(jRadioButton4);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.insets = new java.awt.
            Insets(22, 21, 7, 17);
        gridBagConstraints.anchor =
            java.awt.GridBagConstraints.WEST;
        jPanel1.add(jRadioButton4, gridBagConstraints);

        jLabel1.setText("Error ");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.insets = new java.awt.Insets(15, 0, 0, 1);
        gridBagConstraints.anchor =
            java.awt.GridBagConstraints.EAST;
        jPanel1.add(jLabel1, gridBagConstraints);

        jTextField1.setText("0.01");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.fill = java.awt.
            GridBagConstraints.HORIZONTAL;
        gridBagConstraints.insets = new java.awt.
            Insets(15, 2, 0, 50);
        gridBagConstraints.anchor =
            java.awt.GridBagConstraints.WEST;
        jPanel1.add(jTextField1, gridBagConstraints);

        getContentPane().add(jPanel1,
            java.awt.BorderLayout.CENTER);

        pack();
    }

    private void okButtonActionPerformed(
        java.awt.event.ActionEvent evt) {
        error = Double.parseDouble(jTextField1.getText());
        absolute = jRadioButton1.isSelected();
        ref = jRadioButton3.isSelected();
        doClose(RET_OK);
    }
```

```
    private void cancelButtonActionPerformed(
        java.awt.event.ActionEvent evt) {
        doClose(RET_CANCEL);
    }

    /** Closes the dialog */
    private void closeDialog(java.awt.event.WindowEvent evt) {
        doClose(RET_CANCEL);
    }

    private void doClose(int retStatus) {
        returnStatus = retStatus;
        setVisible(false);
        dispose();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new ErrorDialog(new javax.swing.JFrame(), true).show();
    }

    // Variables declaration - do not modify
    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.ButtonGroup buttonGroup2;
    private javax.swing.JPanel buttonPanel;
    private javax.swing.JButton cancelButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JRadioButton jRadioButton1;
    private javax.swing.JRadioButton jRadioButton2;
    private javax.swing.JRadioButton jRadioButton3;
    private javax.swing.JRadioButton jRadioButton4;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JButton okButton;
    // End of variables declaration

    private int returnStatus = RET_CANCEL;
    public static double error = .01;
    public static boolean absolute = true;
    public static boolean ref = true;
```

```
}
```

A.3 LightDialog.class

```
/*
 * LightDialog.java
 *
 * Created on February 12, 2004, 9:35 PM
 */

/**
 *
 * @author  turley
 */
public class LightDialog extends javax.swing.JDialog {
    /** A return status code - returned if Cancel
     button has been pressed */
    public static final int RET_CANCEL = 0;
    /** A return status code - returned if OK button
    has been pressed */
    public static final int RET_OK = 1;

    /** Creates new form ErrorDialog */
    public LightDialog(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }

    /** @return the return status of this dialog - one of
    RET_OK or RET_CANCEL */
    public int getReturnStatus() {
        return returnStatus;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of
     * this method is always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        java.awt.GridBagConstraints gridBagConstraints;

        buttonGroup1 = new javax.swing.ButtonGroup();
```

```
        buttonPanel = new javax.swing.JPanel();
        okButton = new javax.swing.JButton();
        cancelButton = new javax.swing.JButton();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        waveTextField = new javax.swing.JTextField();
        percentTextField = new javax.swing.JTextField();

        setTitle("Set Light Parameters");
        setName("LightDialog");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(
                java.awt.event.WindowEvent evt) {
                closeDialog(evt);
            }
        });

        buttonPanel.setLayout(new java.awt.FlowLayout(
            java.awt.FlowLayout.RIGHT));

        okButton.setText("OK");
        okButton.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                okButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(okButton);

        cancelButton.setText("Cancel");
        cancelButton.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                cancelButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(cancelButton);
```

```
getContentPane().add(buttonPanel,
    java.awt.BorderLayout.SOUTH);

jPanel1.setLayout(new java.awt.GridBagLayout());

jLabel1.setText("Wavelength");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.insets = new java.awt.
    Insets(13, 100, 13, 10);
jPanel1.add(jLabel1, gridBagConstraints);

if(nkLookup.wavelength>0){
    waveTextField.setText(
        Double.toString(nkLookup.wavelength));
}
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.ipadx = 45;
gridBagConstraints.insets =
    new java.awt.Insets(0, 0, 0, 100);
jPanel1.add(waveTextField, gridBagConstraints);

jLabel2.setText("Percent s polarization");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.insets =
    new java.awt.Insets(0, 0, 13, 10);
gridBagConstraints.anchor =
    java.awt.GridBagConstraints.EAST;
jPanel1.add(jLabel2, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.ipadx = 30;
gridBagConstraints.anchor = java.awt.
    GridBagConstraints.NORTHWEST;
jPanel1.add(percentTextField, gridBagConstraints);

getContentPane().add(
    jPanel1, java.awt.BorderLayout.CENTER);
```

```
        pack();
        percentTextField.setText(Double.toString(sPercent));
    }//GEN-END:initComponents

    private void okButtonActionPerformed(
        java.awt.event.ActionEvent evt) {
        nkLookup.wavelength=Double.parseDouble(
            waveTextField.getText());
        sPercent = Double.parseDouble(
            percentTextField.getText());
        doClose(RET_OK);
    }

    private void cancelButtonActionPerformed(
        java.awt.event.ActionEvent evt) {
        doClose(RET_CANCEL);
    }

    /** Closes the dialog */
    private void closeDialog(java.awt.event.WindowEvent evt) {
        doClose(RET_CANCEL);
    }

    private void doClose(int retStatus) {
        returnStatus = retStatus;
        setVisible(false);
        dispose();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new LightDialog(new javax.swing.JFrame(), true).show();
    }

    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.JPanel buttonPanel;
    private javax.swing.JButton cancelButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
```

```
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField waveTextField;
    private javax.swing.JTextField percentTextField;
    private javax.swing.JButton okButton;

    private int returnStatus = RET_CANCEL;

    public static double sPercent = 90.0;
}
```

A.4 OffsetDialog.class

```
/*
 * Created on May 15, 2004
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;
 * Code and Comments
 */

/**
 * @author farnsworth
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;
 * Code and Comments
 */
/*
 * Dialog.java
 *
 * Created on May 11, 2004, 3:56 PM
 */
public class OffsetDialog extends javax.swing.JDialog {
    /** A return status code - returned if Cancel button
    has been pressed */
    public static final int RET_CANCEL = 0;
    /** A return status code - returned if OK button
    has been pressed */
    public static final int RET_OK = 1;

    /** Creates new form Dialog */
    public OffsetDialog(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
```

```
        initComponents();
    }

    /** @return the return status of this dialog - one of
    RET_OK or RET_CANCEL */
    public int getReturnStatus() {
        return returnStatus;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code.
     * The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        buttonGroup1 = new javax.swing.ButtonGroup();
        buttonPanel = new javax.swing.JPanel();
        okButton = new javax.swing.JButton();
        cancelButton = new javax.swing.JButton();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jRadioButton1 = new javax.swing.JRadioButton();
        jRadioButton2 = new javax.swing.JRadioButton();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();

        setTitle("Set Offset");
        setName("Offset Dialog");
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(
                java.awt.event.WindowEvent evt) {
                closeDialog(evt);
            }
        });

        buttonPanel.setLayout(new java.awt.FlowLayout(
            java.awt.FlowLayout.RIGHT));
```

```
        okButton.setText("OK");
        okButton.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                okButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(okButton);

        cancelButton.setText("Cancel");
        cancelButton.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                cancelButtonActionPerformed(evt);
            }
        });

        buttonPanel.add(cancelButton);

        getContentPane().add(buttonPanel,
            java.awt.BorderLayout.SOUTH);

        jPanel1.setLayout(new java.awt.GridBagLayout());

****

        jPanel1.setBorder(new javax.swing.border.TitledBorder(
            null, "", javax.swing.border.TitledBorder
                .DEFAULT_JUSTIFICATION, javax.swing.border.
                    TitledBorder.DEFAULT_POSITION, new java.awt.Font(
                        "Dialog", 1, 12)));
        jLabel1.setText("Offset ");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.insets = new java.awt.Insets(
            20, 20, 20, 1);
        gridBagConstraints.anchor = java.awt
            .GridBagConstraints.EAST;
```

```
jPanel1.add(jLabel1, gridBagConstraints);

jTextField1.setText("1.0");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 3;
gridBagConstraints.insets = new java.awt.Insets(
    20, 2, 20, 20);
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
jPanel1.add(jTextField1, gridBagConstraints);

jRadioButton1.setSelected(true);
jRadioButton1.setText("No");
buttonGroup1.add(jRadioButton1);
jRadioButton1.addActionListener(
    new java.awt.event.ActionListener() {
    public void actionPerformed(
        java.awt.event.ActionEvent evt) {
        jRadioButton1ActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.insets = new java.awt.Insets(
    20, 20, 0, 20);
jPanel1.add(jRadioButton1, gridBagConstraints);

jRadioButton2.setText("Yes");
buttonGroup1.add(jRadioButton2);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.insets = new java.awt.Insets(
    20, 20, 0, 0);
jPanel1.add(jRadioButton2, gridBagConstraints);

jLabel2.setText("Do you want to ");
jPanel1.add(jLabel2, new java.awt.GridBagConstraints());
```

```
        jLabel3.setText("offset your data?");
        jPanel1.add(jLabel3, new java.awt.GridBagConstraints());

        getContentPane().add(jPanel1, java.awt.BorderLayout.CENTER);

        pack();
    }

    private void jRadioButton1ActionPerformed(
        java.awt.event.ActionEvent evt) {
        // Add your handling code here:
    }

    private void okButtonActionPerformed(
        java.awt.event.ActionEvent evt) {
        offset = jRadioButton2.isSelected();
        delta = Double.parseDouble(jTextField1.getText());
        doClose(RET_OK);
    }

    private void cancelButtonActionPerformed(
        java.awt.event.ActionEvent evt) {
        doClose(RET_CANCEL);
    }

    /** Closes the dialog */
    private void closeDialog(java.awt.event.WindowEvent evt) {
        doClose(RET_CANCEL);
    }

    private void doClose(int retStatus) {
        returnStatus = retStatus;
        setVisible(false);
        dispose();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new OffsetDialog(new javax.swing.JFrame(), true).show();
```

```
    }

    // Variables declaration - do not modify
    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.JPanel buttonPanel;
    private javax.swing.JButton cancelButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JRadioButton jRadioButton1;
    private javax.swing.JRadioButton jRadioButton2;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JButton okButton;
    // End of variables declaration

    private int returnStatus = RET_CANCEL;
    public static boolean offset = false;
    public static double delta = 1.0;
}
```

A.5 nkLookup.class

```
/*
 * nkLookup.java
 *
 * Created on February 9, 2004, 12:44 PM
 */

/**
 *
 * @author  turley
 */
public class nkLookup extends javax.swing.JDialog {
    /** A return status code - returned if Cancel
    button has been pressed */
    public static final int RET_CANCEL = 0;
    /** A return status code - returned if
    OK button has been pressed */
    public static final int RET_OK = 1;

    /** Creates new form nkLookup */
    public nkLookup(java.awt.Frame parent, boolean modal) {
```

```
        super(parent, modal);
        initComponents();
    }

    /** @return the return status of this dialog -
    one of RET_OK or RET_CANCEL */
    public int getReturnStatus() {
        return returnStatus;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content
     * of this method is always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        java.awt.GridBagConstraints gridBagConstraints;

        buttonPanel = new javax.swing.JPanel();
        okButton = new javax.swing.JButton();
        cancelButton = new javax.swing.JButton();
        jTitleLabel = new javax.swing.JLabel();
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(
                java.awt.event.WindowEvent evt) {
                closeDialog(evt);
            }
        });

        buttonPanel.setLayout(new java.awt.FlowLayout(
            java.awt.FlowLayout.RIGHT));

        okButton.setText("OK");
        okButton.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
```

```
            java.awt.event.ActionEvent evt) {
            okButtonActionPerformed(evt);
        }
    });

    buttonPanel.add(okButton);

    cancelButton.setText("Cancel");
    cancelButton.addActionListener(
        new java.awt.event.ActionListener() {
        public void actionPerformed(
            java.awt.event.ActionEvent evt) {
            cancelButtonActionPerformed(evt);
        }
    });

    buttonPanel.add(cancelButton);

    getContentPane().add(buttonPanel,
        java.awt.BorderLayout.SOUTH);

    jTitleLabel.setFont(new java.awt.Font("Dialog", 1, 18));
    jTitleLabel.setHorizontalAlignment(
        javax.swing.SwingConstants.CENTER);
    jTitleLabel.setText("Get n and k from IMD data file");
    getContentPane().add(jTitleLabel,
        java.awt.BorderLayout.NORTH);

    jPanel1.setLayout(new java.awt.GridBagLayout());

    jPanel1.setBorder(new javax.swing.border.EtchedBorder());
    jLabel1.setText("Compound");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.insets = new java.awt.Insets(
        7, 1, 13, 11);
    jPanel1.add(jLabel1, gridBagConstraints);

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.insets = new java.awt.Insets(
        7, 0, 0, 0);
    gridBagConstraints.anchor = java.awt.
        GridBagConstraints.NORTHWEST;
```

```
        gridBagConstraints.ipadx = 60;
        jPanel1.add(jTextField1, gridBagConstraints);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.anchor = java.awt.
            GridBagConstraints.NORTHWEST;
        gridBagConstraints.ipadx = 60;
        jPanel1.add(jTextField2, gridBagConstraints);

        jLabel2.setText("Wavelength");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.insets = new java.awt.Insets(
            0, 0, 7, 0);
        gridBagConstraints.anchor = java.awt.
            GridBagConstraints.NORTHWEST;
        jPanel1.add(jLabel2, gridBagConstraints);

        getContentPane().add(jPanel1, java.awt.BorderLayout.CENTER);

        java.awt.Dimension screenSize = java.awt.
            Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-400)/2, (
            screenSize.height-200)/2, 400, 200);
    }//GEN-END:initComponents

    private void okButtonActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_okButtonActionPerformed
        // Parse input and save it in static variables
        compound = jTextField1.getText();
        String wlStr = jTextField2.getText();
        if(wlStr!=null){
            try{
                wavelength = Double.parseDouble(
                    jTextField2.getText());
            } catch (java.lang.NumberFormatException e){
                wavelength = -1; // flag as wavelength not set
            }
```

```
        }
        doClose(RET_OK);
    }//GEN-LAST:event_okButtonActionPerformed

    private void cancelButtonActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_cancelButtonActionPerformed
        doClose(RET_CANCEL);
    }//GEN-LAST:event_cancelButtonActionPerformed

    /** Closes the dialog */
    private void closeDialog(java.awt.event.WindowEvent evt)
    {//GEN-FIRST:event_closeDialog
        doClose(RET_CANCEL);
    }//GEN-LAST:event_closeDialog

    private void doClose(int retStatus) {
        returnStatus = retStatus;
        setVisible(false);
        dispose();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new nkLookup(new javax.swing.JFrame(), true).show();
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JPanel buttonPanel;
    private javax.swing.JButton cancelButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JLabel jTitleLabel;
    private javax.swing.JButton okButton;
    // End of variables declaration//GEN-END:variables

    private int returnStatus = RET_CANCEL;
```

```
    public static double wavelength = -1;
    public static String compound = null;

}
```

A.6 rindex.class

```
/*
 * rindex.java
 *
 * Created on November 6, 2003, 2:27 PM
 */

/**
 *
 * @author  turley
 */

import java.io.*; import java.net.*; import java.util.*;

/** Retrieves index information from file. */ public class rindex{

    /** Creates a new instance of rindex */

    /** Returns index from file for given material and wavelength */
    static public index nk(String material, double wavelength)
            throws WavelengthException, MalformedURLException,
                 IOException{
        index temp;
        temp=new index();
        // Set the directory based on system
        final String URLdir="http://volta.byu.edu/imd/";
        final String diskdir="c:/BYU/imd/";
        URL url = new URL(URLdir+material+".nk");
        URLConnection connection = url.openConnection();

        BufferedReader in=null;
        try{
            in = new BufferedReader(
                new InputStreamReader(connection.getInputStream()));
        } catch (Exception e) {
            // Can't open file, try a local direction
```

```
            in = new BufferedReader(
                new InputStreamReader(
                new FileInputStream(diskdir+material+".nk")));
        }
        String line;
        do{
            in.mark(150); // mark begining of line
            line=in.readLine();
        } while (line.charAt(0)==';');
        in.reset();  // reposition to start of first data line
        double wl=0, n=0, k=0, lwl=0, ln=0, lk=0;
        while ((line=in.readLine()) != null){
            // Now the line needs to be parsed
            StringTokenizer t = new StringTokenizer(line);
            wl=Double.parseDouble(t.nextToken());
            n=Double.parseDouble(t.nextToken());
            k=Double.parseDouble(t.nextToken());
            if(wavelength<wl){
                // I found what I'm looking for, interpolate and exit
                in.close();
                temp.n=ln+(n-ln)*(wavelength-lwl)/(wl-lwl);
                temp.k=lk+(k-lk)*(wavelength-lwl)/(wl-lwl);
                return temp;
            }
            // Save these value for possible interpolation
            lwl=wl;
            ln=n;
            lk=k;
        }
        // I should not have gotten this far
        in.close();
        WavelengthException ex=new WavelengthException(
            "Wavelength "+wavelength+" not found.");
        throw (ex);
    }
}

class WavelengthException extends RuntimeException {
    WavelengthException(){}
    WavelengthException(String msg){super(msg);}
}
```

A.7 EditLayerDialog.class

```
/*
 * OKCancelDialog.java
 *
 * Created on December 31, 2003, 8:03 AM
 */

/**
 *
 * @author  turley
 */

import javax.swing.JOptionPane;

public class EditLayerDialog extends javax.swing.JDialog {
    /** A return status code - returned if Cancel
    button has been pressed */
    public static final int RET_CANCEL = 0;
    /** A return status code - returned if OK
    button has been pressed */
    public static final int RET_OK = 1;
    /** copy of passed mirror information */
    private Film layer;

    /** Creates new form OKCancelDialog */
    public EditLayerDialog(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }

    /** Creates new form OKCancelDialog
     * @param parent parent frame
     * @param modal true is this is a modal dialog box
     * @param film Use to set initial values in dialog box.
     * If the OK button is used to exit the dialog box,
     * the values fromt the dialog are passed back to
     * here.  Otherwise, they will remain unchanged.
     */
    public EditLayerDialog(java.awt.Frame parent,
        boolean modal, Film layer) {
        super(parent, modal);
```

```
        this.layer=layer;
        // make a copy for passing back parameters
        initComponents();
        /*
         * Initialize buttons according to passed information
         */
        NameTextField.setText(layer.name);
        // n
        nValue.setText(Double.toString(layer.n.value));
        nFixedBox.setSelected(layer.n.fixed);
        nConstrainedBox.setSelected(layer.n.constrained);
        nConstrainedBox.setEnabled(!layer.n.fixed);
        nMin.setText(Double.toString(layer.n.min));
        nMin.setEnabled(layer.n.constrained && !layer.n.fixed);
        nMax.setText(Double.toString(layer.n.max));
        nMax.setEnabled(layer.n.constrained && !layer.n.fixed);
        // k
        kValue.setText(Double.toString(layer.k.value));
        kFixedBox.setSelected(layer.k.fixed);
        kConstrainedBox.setSelected(layer.k.constrained);
        kConstrainedBox.setEnabled(!layer.k.fixed);
        kMin.setText(Double.toString(layer.k.min));
        kMin.setEnabled(layer.k.constrained && !layer.k.fixed);
        kMax.setText(Double.toString(layer.k.max));
        kMax.setEnabled(layer.k.constrained && !layer.k.fixed);
        // z
        zValue.setText(Double.toString(layer.z.value));
        zFixedBox.setSelected(layer.z.fixed);
        zConstrainedBox.setSelected(layer.z.constrained);
        zConstrainedBox.setEnabled(!layer.z.fixed);
        zMin.setText(Double.toString(layer.z.min));
        zMin.setEnabled(layer.z.constrained && !layer.k.fixed);
        zMax.setText(Double.toString(layer.z.max));
        zMax.setEnabled(layer.z.constrained && !layer.k.fixed);
    }

    /** @return the return status of this dialog - one
    of RET_OK or RET_CANCEL */
    public int getReturnStatus() {
        return returnStatus;
    }
```

```
/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of
 * this method is always regenerated by the Form Editor.
 */
private void initComponents() {//GEN-BEGIN:initComponents
    buttonPanel = new javax.swing.JPanel();
    okButton = new javax.swing.JButton();
    cancelButton = new javax.swing.JButton();
    EditTitleLabel = new javax.swing.JLabel();
    EditLayerFieldPanel = new javax.swing.JPanel();
    NameLabel = new javax.swing.JLabel();
    NameTextField = new javax.swing.JTextField();
    nLabel = new javax.swing.JLabel();
    nValue = new javax.swing.JTextField();
    nLabel1 = new javax.swing.JLabel();
    zValue = new javax.swing.JTextField();
    kValue = new javax.swing.JTextField();
    nLabel2 = new javax.swing.JLabel();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    nFixedBox = new javax.swing.JCheckBox();
    zFixedBox = new javax.swing.JCheckBox();
    kFixedBox = new javax.swing.JCheckBox();
    nConstrainedBox = new javax.swing.JCheckBox();
    kConstrainedBox = new javax.swing.JCheckBox();
    zConstrainedBox = new javax.swing.JCheckBox();
    nMin = new javax.swing.JTextField();
    zMin = new javax.swing.JTextField();
    kMin = new javax.swing.JTextField();
    zMax = new javax.swing.JTextField();
    kMax = new javax.swing.JTextField();
    nMax = new javax.swing.JTextField();

    addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(
            java.awt.event.MouseEvent evt) {
            formMouseClicked(evt);
        }
    });
```

```
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(
        java.awt.event.WindowEvent evt) {
        closeDialog(evt);
    }
});

buttonPanel.setLayout(new java.awt.FlowLayout(
    java.awt.FlowLayout.CENTER, 10, 5));

okButton.setText("OK");
okButton.addActionListener(
    new java.awt.event.ActionListener() {
    public void actionPerformed(
        java.awt.event.ActionEvent evt) {
        okButtonActionPerformed(evt);
    }
});

buttonPanel.add(okButton);

cancelButton.setText("Cancel");
cancelButton.addActionListener(
    new java.awt.event.ActionListener() {
    public void actionPerformed(
        java.awt.event.ActionEvent evt) {
        cancelButtonActionPerformed(evt);
    }
});

buttonPanel.add(cancelButton);

getContentPane().add(buttonPanel,
    java.awt.BorderLayout.SOUTH);

EditTitleLabel.setFont(new java.awt.Font(
    "Dialog", 1, 18));
EditTitleLabel.setHorizontalAlignment(
    javax.swing.SwingConstants.CENTER);
EditTitleLabel.setText("Edit Layer");
getContentPane().add(EditTitleLabel,
    java.awt.BorderLayout.NORTH);
```

```
EditLayerFieldPanel.setLayout(null);

NameLabel.setText("name");
EditLayerFieldPanel.add(NameLabel);
NameLabel.setBounds(20, 20, 40, 30);

NameTextField.setText("vacuum");
EditLayerFieldPanel.add(NameTextField);
NameTextField.setBounds(70, 20, 340, 30);

nLabel.setText("n");
EditLayerFieldPanel.add(nLabel);
nLabel.setBounds(10, 90, 10, 30);

nValue.setText("1");
nValue.addMouseListener(
    new java.awt.event.MouseAdapter() {
    public void mouseClicked(
        java.awt.event.MouseEvent evt) {
            nkMouseClicked(evt);
    }
});
EditLayerFieldPanel.add(nValue);
nValue.setBounds(30, 90, 130, 30);

nLabel1.setText("k");
EditLayerFieldPanel.add(nLabel1);
nLabel1.setBounds(10, 130, 10, 30);

zValue.setText("1");
EditLayerFieldPanel.add(zValue);
zValue.setBounds(30, 170, 130, 30);

kValue.setText("1");
kValue.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(
        java.awt.event.MouseEvent evt) {
            nkMouseClicked(evt);
    }
});
EditLayerFieldPanel.add(kValue);
```

```
        kValue.setBounds(30, 130, 130, 30);

        nLabel2.setText("z");
        EditLayerFieldPanel.add(nLabel2);
        nLabel2.setBounds(10, 170, 10, 30);

        jLabel1.setText("fixed");
        EditLayerFieldPanel.add(jLabel1);
        jLabel1.setBounds(170, 70, 30, 16);

        jLabel2.setText("constrained");
        EditLayerFieldPanel.add(jLabel2);
        jLabel2.setBounds(220, 70, 70, 16);

        jLabel3.setText("min");
        EditLayerFieldPanel.add(jLabel3);
        jLabel3.setBounds(330, 70, 41, 16);

        jLabel4.setText("max");
        EditLayerFieldPanel.add(jLabel4);
        jLabel4.setBounds(420, 70, 41, 16);

        nFixedBox.setSelected(true);
        nFixedBox.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                nFixedBoxActionPerformed(evt);
            }
        });

        EditLayerFieldPanel.add(nFixedBox);
        nFixedBox.setBounds(180, 95, 20, 21);

        zFixedBox.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                zFixedBoxActionPerformed(evt);
            }
        });
```

```
        EditLayerFieldPanel.add(zFixedBox);
        zFixedBox.setBounds(180, 175, 20, 21);

        kFixedBox.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                kFixedBoxActionPerformed(evt);
            }
        });

        EditLayerFieldPanel.add(kFixedBox);
        kFixedBox.setBounds(180, 135, 20, 21);

        nConstrainedBox.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                nConstrainedBoxActionPerformed(evt);
            }
        });

        EditLayerFieldPanel.add(nConstrainedBox);
        nConstrainedBox.setBounds(240, 95, 20, 21);

        kConstrainedBox.setSelected(true);
        kConstrainedBox.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                kConstrainedBoxActionPerformed(evt);
            }
        });

        EditLayerFieldPanel.add(kConstrainedBox);
        kConstrainedBox.setBounds(240, 135, 20, 21);

        zConstrainedBox.addActionListener(
            new java.awt.event.ActionListener() {
            public void actionPerformed(
                java.awt.event.ActionEvent evt) {
                zConstrainedBoxActionPerformed(evt);
```

```
            }
        });

        EditLayerFieldPanel.add(zConstrainedBox);
        zConstrainedBox.setBounds(240, 175, 20, 21);

        nMin.setText("1");
        nMin.setEnabled(false);
        EditLayerFieldPanel.add(nMin);
        nMin.setBounds(310, 90, 60, 30);

        zMin.setText("1");
        zMin.setEnabled(false);
        EditLayerFieldPanel.add(zMin);
        zMin.setBounds(310, 170, 60, 30);

        kMin.setText("0.8");
        EditLayerFieldPanel.add(kMin);
        kMin.setBounds(310, 130, 60, 30);

        zMax.setText("1");
        zMax.setEnabled(false);
        EditLayerFieldPanel.add(zMax);
        zMax.setBounds(400, 170, 60, 30);

        kMax.setText("1.2");
        EditLayerFieldPanel.add(kMax);
        kMax.setBounds(400, 130, 60, 30);

        nMax.setText("1");
        nMax.setEnabled(false);
        EditLayerFieldPanel.add(nMax);
        nMax.setBounds(400, 90, 60, 30);

        getContentPane().add(EditLayerFieldPanel,
            java.awt.BorderLayout.CENTER);

        java.awt.Dimension screenSize =
            java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-500)/2, (
            screenSize.height-300)/2, 500, 300);
    }//GEN-END:initComponents
```

```
private void zConstrainedBoxActionPerformed(
    java.awt.event.ActionEvent evt)
    {//GEN-FIRST:event_zConstrainedBoxActionPerformed
    // This will enable or disable the min and max boxes
    zSetConstraints();
}//GEN-LAST:event_zConstrainedBoxActionPerformed

/** Enable or disable zMin and zMax boxes
depending on whether variable is constrained
 *
 */
private void zSetConstraints(){
    if(zConstrainedBox.isSelected()){
        zMin.setEnabled(true);
        zMax.setEnabled(true);
    } else {
        zMin.setEnabled(false);
        zMax.setEnabled(false);
    }
}

private void kConstrainedBoxActionPerformed(
    java.awt.event.ActionEvent evt)
    {//GEN-FIRST:event_kConstrainedBoxActionPerformed
    // This will enable or disable the min and max boxes
    kSetConstraints();
}//GEN-LAST:event_kConstrainedBoxActionPerformed

/** Enable or disable kMin and kMax boxes depending
on whether variable is constrained */
private void kSetConstraints(){
    if(kConstrainedBox.isSelected()){
        kMin.setEnabled(true);
        kMax.setEnabled(true);
    } else {
        kMin.setEnabled(false);
        kMax.setEnabled(false);
    }
}

private void nConstrainedBoxActionPerformed(
```

```
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_nConstrainedBoxActionPerformed
        // This will enable or disable the min and max boxes
        nSetConstraints();
    }//GEN-LAST:event_nConstrainedBoxActionPerformed

    /** Enable or disable nMin and nMax boxes
    depending on whether variable is constrained
     *
     */
    private void nSetConstraints(){
        if(nConstrainedBox.isSelected()){
            nMin.setEnabled(true);
            nMax.setEnabled(true);
        } else {
            nMin.setEnabled(false);
            nMax.setEnabled(false);
        }
    }

    private void nFixedBoxActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_nFixedBoxActionPerformed
        if(nFixedBox.isSelected()){
            nConstrainedBox.setEnabled(false);
            nMin.setEnabled(false);
            nMax.setEnabled(false);
        } else {
            nConstrainedBox.setEnabled(true);
            nSetConstraints();
        }
    }//GEN-LAST:event_nFixedBoxActionPerformed

    /**
     * User right-clicked on the mouse in the n or k boxes.
     * Let them read in the n and k values from a file.
     * @param evt MouseEvent created by the user
     * clicking on the box.
     */
    private void nkMouseClicked(java.awt.event.MouseEvent evt) {
        if(evt.getButton()==java.awt.event.MouseEvent.BUTTON3){
            new nkLookup(new javax.swing.JFrame(), true).show();
```

```
            if(nkLookup.wavelength>0 && nkLookup.compound!=null){
                try{
                    index idx=rindex.nk(nkLookup.compound,
                        nkLookup.wavelength);
                    nValue.setText(Double.toString(idx.n));
                    kValue.setText(Double.toString(idx.k));
                } catch (Exception e){
                    JOptionPane.showMessageDialog(this,
                        "material or wavelength not found",
                        "Error Finding Index",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }

    private void kFixedBoxActionPerformed(
        java.awt.event.ActionEvent evt) {
        if(kFixedBox.isSelected()){
            kConstrainedBox.setEnabled(false);
            kMin.setEnabled(false);
            kMax.setEnabled(false);
        } else {
            kConstrainedBox.setEnabled(true);
            kSetConstraints();
        }
    }

    private void zFixedBoxActionPerformed(
        java.awt.event.ActionEvent evt) {
        if(zFixedBox.isSelected()){
            zConstrainedBox.setEnabled(false);
            zMin.setEnabled(false);
            zMax.setEnabled(false);
        } else {
            zConstrainedBox.setEnabled(true);
            zSetConstraints();
        }
    }

    private void formMouseClicked(
        java.awt.event.MouseEvent evt)
```

```
        {//GEN-FIRST:event_formMouseClicked
        // Add your handling code here:
    }//GEN-LAST:event_formMouseClicked

    private void okButtonActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_okButtonActionPerformed
        /*
         * Set mirror optics according to values in dialog box
         */
        layer.name=NameTextField.getText();
        // n
        try{
            layer.n.value=Double.parseDouble(nValue.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in n field: "+nValue.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        layer.n.fixed=nFixedBox.isSelected();
        layer.n.constrained=nConstrainedBox.isSelected();
        try{
            layer.n.min=Double.parseDouble(nMin.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in n min field: "+nMin.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        try{
            layer.n.max=Double.parseDouble(nMax.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in n max field: "+nMax.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
```

```
            return;
        }
        // k
        try{
            layer.k.value=Double.parseDouble(kValue.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in k field: "+kValue.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        layer.k.fixed=kFixedBox.isSelected();
        layer.k.constrained=kConstrainedBox.isSelected();
        try{
            layer.k.min=Double.parseDouble(kMin.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in k min field: "+nValue.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        try{
            layer.k.max=Double.parseDouble(kMax.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in k max field: "+kValue.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        // z
        try{
            layer.z.value=Double.parseDouble(zValue.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in z field: "+zValue.getText(),
```

```
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        layer.z.fixed=zFixedBox.isSelected();
        layer.z.constrained=zConstrainedBox.isSelected();
        try{
            layer.z.min=Double.parseDouble(zMin.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in z min field: "+zMin.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        try{
            layer.z.max=Double.parseDouble(zMax.getText());
        } catch (Exception e){
            JOptionPane.showMessageDialog(
                this,
                "bad data in z max field: "+zMax.getText(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
                return;
        }
        doClose(RET_OK);
    }//GEN-LAST:event_okButtonActionPerformed

    private void cancelButtonActionPerformed(
        java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_cancelButtonActionPerformed
        doClose(RET_CANCEL);
    }//GEN-LAST:event_cancelButtonActionPerformed

    /** Closes the dialog */
    private void closeDialog(java.awt.event.WindowEvent evt)
    {//GEN-FIRST:event_closeDialog
        doClose(RET_CANCEL);
    }//GEN-LAST:event_closeDialog

    private void doClose(int retStatus) {
```

```
        returnStatus = retStatus;
        setVisible(false);
        dispose();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new EditLayerDialog(new javax.swing.JFrame(), true).show();
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JPanel EditLayerFieldPanel;
    private javax.swing.JLabel EditTitleLabel;
    private javax.swing.JLabel NameLabel;
    private javax.swing.JTextField NameTextField;
    private javax.swing.JPanel buttonPanel;
    private javax.swing.JButton cancelButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JCheckBox kConstrainedBox;
    private javax.swing.JCheckBox kFixedBox;
    private javax.swing.JTextField kMax;
    private javax.swing.JTextField kMin;
    private javax.swing.JTextField kValue;
    private javax.swing.JCheckBox nConstrainedBox;
    private javax.swing.JCheckBox nFixedBox;
    private javax.swing.JLabel nLabel;
    private javax.swing.JLabel nLabel1;
    private javax.swing.JLabel nLabel2;
    private javax.swing.JTextField nMax;
    private javax.swing.JTextField nMin;
    private javax.swing.JTextField nValue;
    private javax.swing.JButton okButton;
    private javax.swing.JCheckBox zConstrainedBox;
    private javax.swing.JCheckBox zFixedBox;
    private javax.swing.JTextField zMax;
    private javax.swing.JTextField zMin;
    private javax.swing.JTextField zValue;
```

```
    // End of variables declaration//GEN-END:variables

    private int returnStatus = RET_CANCEL;
}
```

A.8 Mirror.class

```
/*
 * Mirror.java
 *
 * Created on December 30, 2003, 11:46 AM
 */

/**
 *
 * @author  turley
 */
import java.util.Vector;

public class Mirror implements java.io.Serializable {

    /** Text description of mirror */
    public String description = "";

    /** Used as a normlization of the reflectance when fitting
     * unnormalized reflectance measurements (such as XRD data)
     */
    public FitParameter norm = new FitParameter(1.);

    /** Top (external) layer for the mirror.  Default is to assume
     * it is a vacuum.
     */
    public Film vacuum;

    /** Vector of Film for each thin film in the stack
     * (excluding vacuum and substrate)
     */
    private Vector films=new Vector();

    /** Bottom layer in the stack, assumed to be
    infinitely thick. */
    public Film substrate;
```

```
/** Creates a new instance of Mirror */
public Mirror() {
    vacuum=new Film(1,0,"vacuum",0);
    substrate=new Film(1,0,"substrate",0);
    // I don't know the wavelength, so treat it
    //as a vacuum for now
}

public String toString() {
    StringBuffer buf=new StringBuffer(description+'\n');
    buf.append("normalization="+norm+'\n');
    buf.append(vacuum.toString()+'\n');
    for(int i=0; i<films.size(); i++){
        buf.append(films.elementAt(i).toString()+'\n');
    }
    buf.append(substrate.toString());
    return buf.toString();
}

/** Adds a layer to the bottom of the stack (
next to the substrate) */
public void addFilm(Film layer) {
    films.add(layer);
}

/** Adds a layer to the stack at position location, where
 * position 0 is closest to the vacuum.
 */
public void addFilmAt(Film layer, int location) {
    films.add(location, layer);
}

/** Remove the layer at position <B>location</B> where
 * <B>location=0</B> is next to the vacuum.
 */
public void removeFilm(int location) {
    films.remove(location);
}

/** returns number of films in the mirror
 * (including vacuum and substrate)
 */
```

```
    public int numFilms() {
        return 2+films.size();
    }

    /** returns the film at <B>location</B>.
     * @param location location in the film list.
     * Location 0 is closest to the vacuum layer.
     */
    public Film getFilm(int location) {
        return (Film)films.elementAt(location);
    }

}
```

A.9 Film.class

```
/*
 * Film.java
 *
 * Created on December 30, 2003, 11:25 AM
 */

/** Represents a layer in a multilayer mirror.
 * @author turley
 */
public class Film implements java.io.Serializable{

    /** Name of the material composing the layer. */
    public String name = "Layer";

    /** Real part of the index of refraction. */
    public FitParameter n;

    /** Imaginary part of the index of refraction */
    public FitParameter k;

    /** Thickness of the layer in Angstroms. */
    public FitParameter z;

    /** Creates a new instance of Layer */
    public Film() {
    }
```

```
    /** Constructor with initialization.
     * @param n real part of the index of refraction
     * @param k imaginary part of the index of refraction
     * @param name material description
     * @param thickness thickness of the layer in angstrongs
     */
    public Film(double n, double k, String name, double thickness) {
        this.n=new FitParameter(n);
        this.k=new FitParameter(k);
        this.name=name;
        z=new FitParameter(thickness);
    }

    public String toString() {
        StringBuffer buf=new StringBuffer(name+'\n');
        buf.append("n: "+n.toString()+'\n');
        buf.append("k: "+k.toString()+'\n');
        buf.append("z: "+z.toString());
        return buf.toString();
    }

}
```

A.10 index.class

```
/*
 * index.java
 *
 * Created on November 6, 2003, 2:29 PM
 */

/** The index of refraction for a material.
 * @author turley
 */
public class index extends Object {

    /** Index of refraction */
    public double n = 1;

    public double k = 0;

    /** Creates a new instance of index */
    public index() {
```

```
    }

    /** Creates a new instance of index with initialization */
    public index(double xn, double xk){
            n=xn;
            k=xk;
    }
    /** Returns the index information converted to a string. */
    public String toString(){
        /** return the index as a String */
        return "("+n+","+k+")";
    }

}
```

A.11 FitParameter.class

```
/*
 * FitParameter.java
 *
 * Created on December 30, 2003, 11:33 AM
 */

/** A quantity that will be fit with a (double) value,
 * possibility of being fit or constant, and with
 * possible constraints.
 * @author turley
 */
public class FitParameter implements java.io.Serializable{

    /** the initial or constant value for this parameter during
     * the fit
     */
    public double value = 0;

    /** true is this parameter will keep a constant value during
     * the fit
     */
    public boolean fixed = true;

    /** true if this parameter will be constrained to stay between
     * min and max during the fit.  This slows down the fits and
     * can lead to unexpected results.
```

```
     */
    public boolean constrained = false;

    /** minimum possible value for this parameter in fit.  It
     * only has meaning if constrained is true.
     */
    public double min = 0;

    /** maximum possible value for this parameter in fit.  It
     * only has meaning if constrained is true.
     */
    public double max = 0;

    /** Creates a new instance of FitParameter */
    public FitParameter() {
    }

    /** Convert parameter to a string. */
    public FitParameter(double val) {
        value=val;
    }

    /** Convert parameter to a string. */
    public String toString() {
        StringBuffer buf=new StringBuffer(Double.toString(value));
        if(fixed) buf.append(", fixed");
        if(constrained) buf.append(", min="+min+", max="+max);
        return buf.toString();
    }

}
```

A.12 MirrorFunc.class

```
/*
 * ReflFunc.java
 *
 * Created on December 5, 2003, 3:45 PM
 */
// package byu.xrd;
import hep.aida.IAnnotation; import hep.aida.IFunction;

import java.util.Vector;
```

```
/** Computes reflectance of a thin film stack for fitting
applications.
 * This version doesn't allow for model definition with a GUI yet.
 * @author turley
 * @version 0.1
 */
public class MirrorFunc implements IFunction {

    private final int dim = 1;
    // data will be a funciton of angle only for now
    /** This are the names of parameters for fitting.
     * These can be altered programatically through the
     * method or with a GUI using the method.
     */
    // Use Single 100 /AA Mo film on Si substrate for this
    // Fit Mo constant and thickness
    private String spar[];
    private double par[];
    private layer MoLayer;
    private String tit = "Reflection";
    private Vector stack;
    private refl mirror;

    /** Creates a new instance of MirrorFunction
     * It has not been tested, and probably doesn't work
     *
     * @throws java.io.IOException
     */
    public MirrorFunc() throws java.io.IOException {
        /* If this constructor is called, the code may break.
         * It is only here for historical purposes
         * (a copy of the ReflFunc constructor).
         */
        spar = new String[3];
        spar[0] = "Mo n";
        spar[1] = "Mo k";
        spar[2] = "Mo z";
        stack = new Vector();
        index nSi = new index();
        index nMo = new index();
        index nVac = new index(1, 0);
```

```
        final double wavelength = 304;
        try {
            nSi = rindex.nk("Si", wavelength);
            nMo = rindex.nk("Al", wavelength);
        } catch (WavelengthException e) {
            javax.swing.JOptionPane.showMessageDialog(null,
                "Bad wavelength in MirrorFunc", "Wavelength Error",
                javax.swing.JOptionPane.ERROR_MESSAGE);
            throw (e);
        } catch (java.io.IOException e) {
            javax.swing.JOptionPane.showMessageDialog(null,
                "Error reading IMD data file in MirrorFunc",
                    "IO Error",
                        javax.swing.JOptionPane.ERROR_MESSAGE);
            throw(e);
        }
        // Save Mo index as default for generating test data
        par = new double[3];
        par[0] = nMo.n;
        par[1] = nMo.k;
        par[2] = 100.;
        mirror = new refl(wavelength);
        stack.add(new layer(nSi, 0, "Si"));
        MoLayer = new layer(nMo, par[2], "Al");
        stack.add(MoLayer);
        stack.add(new layer(nVac, 0, "Vacuum"));
    }

    /** Creates a new instance of ReflFunction
     *
     * @param m Mirror describing the stack
     */
    public MirrorFunc(Mirror m) {
        spar = new String[m.numFilms()*3-2];
        par = new double[m.numFilms()*3-2];
        // Get substrate parameters (no thickness)
        spar[0] = m.substrate.name+" n";
        spar[1] = m.substrate.name+" k";
        par[0] = m.substrate.n.value;
        par[1] = m.substrate.k.value;
        stack = new Vector();
        stack.add(new layer(new index(par[0], par[1]),
```

```
            0,m.substrate.name));
        // Films above substrate
        for(int i=0; i<m.numFilms()-2; i++){
            Film f = m.getFilm(i);
            spar[3*i+2] = f.name+" n";
            spar[3*i+3] = f.name+" k";
            spar[3*i+4] = f.name+" z";
            par[3*i+2] = f.n.value;
            par[3*i+3] = f.k.value;
            par[3*i+4] = f.z.value;
            stack.add(new layer(new index(f.n.value, f.k.value),
                f.z.value, f.name));
        }
        // Vacuum layer
        int j=3*(m.numFilms()-2)+2;
        spar[j] = m.vacuum.name+" n";
        par[j++] = m.vacuum.n.value;
        spar[j] = m.vacuum.name+" k";
        par[j] = m.vacuum.k.value;
        stack.add(new layer(new index(m.vacuum.n.value,
            m.vacuum.k.value), 0, m.vacuum.name));
        //Verify, then set wavelength
        String lamStr=null;
        new LightDialog(null, true).show();
        mirror = new refl(nkLookup.wavelength, LightDialog.sPercent);
    }

    public IAnnotation annotation() {
        throw new UnsupportedOperationException();
    }

    public String codeletString() {
        // Hopefully, this is ignored, it is not general
        return "codelet:MyFunc:file://c:/Documents and
            Settings/Owner/.JAS3/classes/";
    }

    public int dimension() {
        /*
         * the number of independent variables
         */
        return dim;
```

```
}

public double[] gradient(double[] values) {
    // This is difficult to compute.  Do it numerically if at all
    throw new UnsupportedOperationException();
}

public int indexOfParameter(String str) {
    for (int i = 0; i < spar.length; i++) {
        if (str.equals(spar[i]))
            return i;
    }
    System.err.println("Parameter not found");
    System.err.println("Looking for <"+str+">");
    System.err.println("spar.length="+spar.length);
    for (int i=0; i<spar.length; i++){
        System.err.println("spar["+i+"]=<"+spar[i]+">");
    }
    System.err.println();
    throw new IllegalArgumentException(str + " not a parameter");
}

public boolean isEqual(IFunction iFunction) {
    throw new UnsupportedOperationException();
}

public int numberOfParameters() {
    return par.length;
}

public double parameter(String str) {
    return par[indexOfParameter(str)];
}

public String[] parameterNames() {
    return spar;
}

public double[] parameters() {
    // not implemented in TrialFunctions...
    return par;
}
```

```
public boolean providesGradient() {
    return false; // does not provide gradient
}

/** Set the parameter str to the value param
 * @see hep.aida.IFunction#setParameter(java.lang.String, double)
 */
public void setParameter(String str, double param)
    throws java.lang.IllegalArgumentException {
    // Not supported by TrialFunctions...
    int indx = indexOfParameter(str);
    par[indx] = param;
    if(indx<2){
        // This is a substrate parameter
        layer l=(layer)stack.get(0);
        if(indx==0){
            l.n.re=param; // n
        } else {
            l.n.im=param; // k
        }
    } else {
        // Regular film in stack (or vacuum)
        int film=(indx+1)/3;
        layer l=(layer)stack.get(film);
        int element=(indx+1)%3;
        switch(element){
            case 0:
                l.n.re=param;
                break;
            case 1:
                l.n.im=param;
                break;
            case 2:
                l.thick=param;
        }
    }
} // setParameter

public void setParameters(double[] values) {
    // Not implemented in TrialFunctions
    if (par.length != values.length)
```

```
            throw new IllegalArgumentException();
        par = values;
        layer l=(layer)stack.get(0); // substrate
        int ival=0;
        l.n.re=values[ival++];
        l.n.im=values[ival++];
        for(int i=1; i<stack.size()-1; i++){
            // film layers
            l=(layer)stack.get(i);
            l.n.re=values[ival++];
            l.n.im=values[ival++];
            l.thick=values[ival++];
        }
        l=(layer)stack.get(stack.size()-1); // vacuum
        l.n.re=values[ival++];
        l.n.im=values[ival];
    }

    public void setTitle(String title) {
        tit = title;
    }

    public String title() {
        return tit;
    }

    public double value(double[] values) {
        double theta = values[0];
        double r = 0;
        // Assume stack has been updated with setParameter calls
        if(OffsetDialog.offset){
            theta = theta - OffsetDialog.delta;
        }
        if(theta<60){
            r=mirror.reflectance(theta,stack);
        }
        else{
            r=mirror.refl_trans(theta,stack);
        }
        return r;
    }
```

```
    public String variableName(int param) {
        if (param > 0)
            throw new IllegalArgumentException();
        return "theta (deg)";
    }

    public String[] variableNames() {
        String[] snam = { "theta (deg)" };
        return snam;
    }

    public String toString() {
        StringBuffer strBuf = new StringBuffer("Stack:\n");
        for (int i = 0; i < stack.size(); i++) {
            strBuf.append(((layer) stack.elementAt(i))
                .toString() + "\n");
        }
        return strBuf.toString();
    }

}
```

A.13 layer.class

```
/*
 * layer.java
 *
 * Created on November 13, 2003, 6:34 AM
 */

/**
 *
 * @author  Owner
 */
public class layer {

    /** complex index of refraction */
    public Complex n;

    /** thickness ion Angstroms */
    public double thick;

    /** layer composition */
```

```
    private String description;

    /** Creates a new instance of layer */
    public layer() {
        n=new Complex();
        thick=0;
        description="";
    }

    /** Creates a new instance of layer
     * @param nk index of refraction
     * @param thickness layer thickness in Angstroms
     * @param description layer composition
     */
    public layer(index nk, double thickness,
        java.lang.String name) {
        n=new Complex(nk.n,nk.k);
        thick=thickness;
        description=name;
    }

    /** returns a string describing this layer */
    public String toString() {
        return "<"+description+": "+n+", "+thick+">";
    }

}
```

A.14 refl.class

```
/*
 * refl.java
 *
 * Created on November 7, 2003, 5:25 PM
 */

/** Compute reflectance of a stack of thin films.
 *
 * I anticipate this class will be used in several different ways.
 * For now, assume I'll be fitting data at a fixed wavelength.
 * At one level, I'll need to calculate refl efficiently
 * for multiple angles with index data and layer thicknesses
 * fixed.  I will make future calls with different thicknesses
```

```
 * and index data. These programs do not yet include roughness.
 * It would be simple to add this later.
 *
 * Since our layers tend to be of varying thicknesses, and usually
 * not very thick, I have not included a multilayer capability
 * allowing a stack of several identical layers to be added at once.
 *
 * The thickness of layers in the stack and the wavelength can be
 * in arbitrary units, as long as they are consistent.
 * @author turley
 */
import java.util.Vector; import java.util.ListIterator; import
java.lang.Math;

public class refl {

    private double lambda = 1;
    private static double k = 6.28;
    private double sFrac=0.9;
    private double pFrac=0.1;

    //this method will test our new way (Matrix formulation)
    //of computing R and T
    public static void main(String[] args){
        final double wavelength = 304;
        final double sPercent = 100;//assume s polarization
        //This uses our old method to compute R for comparison
        refl rfl = new refl(wavelength,sPercent);
        Vector stack = new Vector();
        //This mirror is a two layer stack of materials with the
        //following constants
        final double Ns = .929265;
        final double Ks = .0090661;
        final double N1 = .9513;
        final double K1 = .0055939;
        final double N2 = .67786;
        final double K2 =  .27993;
        final double Nv = 1;
        final double Kv = 0;
        final double theta = 75;
        //h is the thickness of both layers
        final double h = 100;
```

```
stack.add(new layer(new index(Ns,Ks),0,"Substrate"));
stack.add(new layer(new index(N2,K2),h,"layer2"));
stack.add(new layer(new index(N1,K1),h,"layer1"));
stack.add(new layer(new index(Nv,Kv),0,"Vacuum"));
//Compute the old way
double r = rfl.reflectance(theta,stack);
System.out.println("r= "+ r);
//Compute the new way
double rnew = rfl.refl_trans(theta,stack);
System.out.println("rnew="+rnew);

//Compute the new way by hand (test version)
Complex n0= new Complex(Ns,Ks);
Complex n1= new Complex(N2,K2);
Complex n2= new Complex(N1,K1);
Complex n3= new Complex(Nv,Kv);
//System.out.println("n0="+n0+" n1="+n1+" n2="+n2+" n3="+n3);

//See the kz method
Complex kz3 = kz(n3,wavelength,theta);
Complex kz2 = kz(n2,wavelength,theta);
Complex kz1 = kz(n1,wavelength,theta);
Complex kz0 = kz(n0,wavelength,theta);
//System.out.println("kz0="+kz0+" kz1="+kz1+" kz2="+kz2+
    " kz3="+kz3);

//The F's are the reflection coefficients (for s polarization)
//F=(k2-k1)/(k1+k2)
Complex F10 = (kz1.minus(kz0)).over(kz0.plus(kz1));
Complex F01 = (kz0.minus(kz1)).over(kz1.plus(kz0));
Complex F21 = (kz2.minus(kz1)).over(kz1.plus(kz2));
Complex F12 = (kz1.minus(kz2)).over(kz2.plus(kz1));
Complex F32 = (kz3.minus(kz2)).over(kz2.plus(kz3));
Complex F23 = (kz2.minus(kz3)).over(kz3.plus(kz2));
//System.out.println("F10="+F10+" F01="+F01+" F21="+F21
    +" F12="+F12+" F32="+F32+" F23="+F23);

//The G's are the transmission coefficients
//(for s polarization)
//G=2*k2/(k1+k2)
Complex G01 = new Complex(2,0).times(kz1).over(kz0.plus(kz1));
Complex G10 = new Complex(2,0).times(kz0).over(kz1.plus(kz0));
```

```
        Complex G12 = new Complex(2,0).times(kz2).over(kz1.plus(kz2));
        Complex G21 = new Complex(2,0).times(kz1).over(kz2.plus(kz1));
        Complex G23 = new Complex(2,0).times(kz3).over(kz2.plus(kz3));
        Complex G32 = new Complex(2,0).times(kz2).over(kz3.plus(kz2));
        //System.out.println("G10="+G10+" G01="+G01+" G21="+G21+
            " G12="+G12+" G32="+G32+" G23="+G23);

        //The C's are the phase and amplitude modulations of the
        //electric field while propagating through the half
        //multilayer.
        Complex C0 = new Complex(1,0);
        Complex C1 = (new Complex(0,h/2).times(kz1)).exp();
        Complex C2 = (new Complex(0,h/2).times(kz2)).exp();
        Complex C3 = (new Complex(0,0).times(kz3)).exp();
        //System.out.println("C0="+C0);
        //System.out.println("C1="+C1);
        //System.out.println("C2="+C2);
        //System.out.println("C3="+C3);

        //See method Am()
        Matrix A0 = Am(G10,G01,F10,F01,C0,C1);
        //System.out.println("A0="+A0);
        Matrix A1 = Am(G21,G12,F21,F12,C1,C2);
        //System.out.println("A1"+A1);
        Matrix A2 = Am(G32,G23,F32,F23,C2,C3);
        //System.out.println("A2="+A2);
        Matrix B = A2.times(A1).times(A0);
        //System.out.println("B="+B);
        //System.out.println("A0="+A0);
        //System.out.println("A1="+A1);
        //System.out.println("A2="+A2);
        //System.out.println("B="+B);

        Complex R = (B.A12).over(B.A22);
        Complex T = new Complex(1,0).over(B.A22);
        //System.out.println("R="+R+" T="+T);
    }

    //Computes the z component of the k vector for a specified n,
    //wavelength, and angle(in degrees).
    private static Complex kz(Complex n,double lambda,double theta){
        Complex k = new Complex(2*Math.PI/lambda,0);
```

```
        Complex kx = k.times(Math.cos(theta*Math.PI/180));
        Complex kz =
            (new Complex(4 * Math.PI * Math.PI / (
                lambda * lambda), 0)
                .times(sqr(n))
                .minus(sqr(kx)))
                .sqrt();
        return kz;
    }

    private static Matrix Am(Complex gpm,Complex gmp,
        Complex fpm,Complex fmp,Complex cm,Complex cp){
        Complex A11 = (gpm.times(cm).times(cp)).minus(
            fpm.times(fmp.times(cm.times(cp))).over(gmp));
        Complex A12 = fpm.times(cp).over(gmp.times(cm));
        Complex A21 = new Complex(-1,0).times(fmp).times(
            cm).over(gmp.times(cp));
        Complex A22 = new Complex(1,0).over(cm.times(
            cp).times(gmp));
        return new Matrix(A11,A12,A21,A22);
    }

    /** Creates a new instance of refl */
    public refl() {
    }

    /** Creates a new instance of the class with the
    specified wavelength */
    public refl(double wavelength, double sPercent) {
        setwavelength(wavelength);
        sFrac=sPercent/100;
        pFrac=1-sFrac;
    }

    /** Creates a new instance of the class with the
    specified wavelength */
    public refl(double wavelength) {
        setwavelength(wavelength);
    }

    /** Produces a string representation of the
    state of the class. */
```

```
public String toString() {
    return Double.toString(lambda);
}

/** Set the wavelength in Angstroms. */
public void setwavelength(double wavelength) {
    lambda=wavelength;
    k=2.0*Math.PI/lambda;
}

/** Compute reflectance of a stack at an angle
 * theta (in degrees).
 * This version assumes that the beam is 90% p
 * polarized and 10% s-pol.
 * @param theta angle (in degrees)
 * @param stack Vector of layer objects describing the stack.
 * Be sure to include the vacuum layer (with a thickness
 * of zero) and substrate (thickness is ignored).
 * The first layer in the stack should be the substrate.
 */
public double reflectance(double theta, Vector stack) {
    double kx; // constant wave number
    Vector klayer=new Vector(); // Complex
    Complex n1, n2, k1, k2;
    Complex fs, fp, C;
    Complex Rs=new Complex(0), Rp=new Complex(0);
    /*
     * Compute wave vector components
     */
    kx=k*Math.cos(theta*Math.PI/180.); // use in called methods
    for(int i=0; i<stack.size(); i++){
        // klayer(i)=sqrt((k*n[i])**2-kx**2)
        Complex n=((layer)(stack.get(i))).n;
        n=(sqr(n.times(k)).minus(sqr(kx))).sqrt();
        klayer.add(n);
    }
    for(int i=0; i<stack.size()-1; i++){
        // The Fresnel coeeficients are between layer i
        //and layer i+1
        k2=(Complex)klayer.get(i+1);
        k1=(Complex)klayer.get(i);
        n2=sqr(((layer)(stack.get(i+1))).n);
```

```
            n1=sqr(((layer)(stack.get(i))).n);
            // fs = (k2-k1)/(k1+k2)
            fs=k2.minus(k1).over(k1.plus(k2));
            // fp = (n1^2*k2-n2^2*k1)/(n2^2*k1+n1^2*k2)
            fp=sqr(n1).times(k2).minus(sqr(n2).times(k1));
            Complex den=sqr(n2).times(k1).plus(sqr(n1).times(k2));
            fp=fp.over(den);
            // Propagation constants
            // C is for propagation through layer i
            // C = exp(2*I*k2*t)
            C=(((new Complex(0,2)).times(k2)).times(((layer)(
                stack.get(i+1))).thick)).exp();
            //System.out.println("C="+C);
            // Rs = C*(fs+Rs)/(1+fs*Rs)
            Rs=(C.times(fs.plus(Rs))).over(((fs.times(Rs)).plus(1)));
            //System.out.println("Rs="+Rs);
            // Rp = C*(fp+Rp)/(1+fp*Rp)
            Rp=(C.times(fp.plus(Rp))).over(((fp.times(Rp)).plus(1)));
            //System.out.println("Rp="+Rp);
        } // for stack elements
        // average, assuming 90% s and 10% p
        return (sFrac*sqr(Rs.mag())+pFrac*sqr(Rp.mag()));
    }

    /**Computes the reflectance and transmission of a stack at
     * an angle theta(in degrees)
     * using matrices.
     * @param theta angle (in degrees)
     * @param stack Vector of layer objects describing the stack.
     * Be sure to include the vacuum layer (with a thickness
     * of zero) and substrate(be sure to include thickness if
     * you want transmission
     * data. If not, thickness=0).
     * The first layer in the stack should be the substrate.
     * Has been checked against the test program eariler in
     * this class and against the old way of computing reflectance
     * (i.e. the Parratt formula).
     * Returns t, but could return both r and t.
     */
    public double refl_trans(double theta,Vector stack){
        double kx;
        Vector klayer = new Vector();
```

```
Complex n1,n2,k1,k2;
Complex Fs21,Fs12,Gs21,Gs12;
Complex Fp21,Fp12,Gp21,Gp12;
Complex C1,C2;
Complex Rs=new Complex(0);Complex Rp=new Complex(0);
Complex Ts=new Complex(0);Complex Tp=new Complex(0);
Matrix As=new Matrix(1,0,0,1);
Matrix Ap=new Matrix(1,0,0,1);
Matrix Bs=new Matrix(1,0,0,1);
Matrix Bp=new Matrix(1,0,0,1);
double r; double t;

kx=k*Math.cos(theta*Math.PI/180);
for(int i=0; i<stack.size(); i++){
    // klayer(i)=sqrt((k*n[i])**2-kx**2)
    Complex n=((layer)(stack.get(i))).n;
    n=(sqr(n.times(k)).minus(sqr(kx))).sqrt();
    klayer.add(n);
}
for (int i=0;i<stack.size()-1;i++){
    k2=(Complex)klayer.get(i+1);
    k1=(Complex)klayer.get(i);
    n2=sqr(((layer)(stack.get(i+1))).n);
    n1=sqr(((layer)(stack.get(i))).n);

    Fs21 = (k2.minus(k1)).over(k1.plus(k2));
    //System.out.println("Fs21="+Fs21);
    Fs12 = (k1.minus(k2)).over(k2.plus(k1));
    //System.out.println("Fs12="+Fs12);
    Gs21 = new Complex(2,0).times(k1).over(k2.plus(k1));
    //System.out.println("Gs21="+Gs21);
    Gs12 = new Complex(2,0).times(k2).over(k1.plus(k2));
    //System.out.println("Gs12="+Gs12);

    Fp21 =
        (sqr(n1).times(k2).minus(sqr(n2).times(k1))).over(
            sqr(n2).times(k1).plus(sqr(n1).times(k2)));
    Fp12 =
        (sqr(n2).times(k1).minus(sqr(n1).times(k2))).over(
            sqr(n1).times(k2).plus(sqr(n2).times(k1)));
    Gp21 =
        new Complex(2, 0).times(
```

```
                (sqr(n2).times(k1)).over(
                    sqr(n1).times(k2).plus(sqr(n2).times(k1))));
        Gp12 =
            new Complex(2, 0).times(
                (sqr(n1).times(k2)).over(
                    sqr(n2).times(k1).plus(sqr(n1).times(k2))));

        C1 = ((new Complex(0,(((layer)(
            stack.get(i))).thick)/2)).times(k1)).exp();
        C2 = ((new Complex(0,(((layer)(
            stack.get(i+1))).thick)/2)).times(k2)).exp();
        //System.out.println("C1="+C1);
        //System.out.println("C2="+C2);

        As = Am(Gs21,Gs12,Fs21,Fs12,C1,C2);
        //System.out.println("As="+As);
        Bs = As.times(Bs);
        //System.out.println("Bs="+Bs);

        Ap = Am(Gp21,Gp12,Fp21,Fp12,C1,C2);
        Bp = Ap.times(Bp);
    }
    Rs = (Bs.A12).over(Bs.A22);
    Ts = new Complex(1, 0).over(Bs.A22);
    //System.out.println("Rs="+Rs);
    //System.out.println("Ts="+Ts);

    Rp = (Bp.A12).over(Bp.A22);
    Tp = new Complex(1, 0).over(Bp.A22);

    r = sFrac * sqr(Rs.mag()) + pFrac * sqr(Rp.mag());
    t = sFrac * sqr(Ts.mag()) + pFrac * sqr(Tp.mag());
    //System.out.println("r="+r+" t="+t);

    return t;
}


private double sqr(double x) {
    return x*x;
}
```

```
private static Complex sqr(Complex z) {
    return z.times(z);
}

/** A fast version of the reflectance class that avoids the use of
 * complex numbers and uses a list instead of a stack.  It is
 * also optimized in other internal ways.  It is a lot harder
 * to read, but runs in about 1/2 to 2/3 the time.  The arguments
 * are the same as reflectance.
 */
public double fast(double theta, Vector stack) {
    double kr[]=new double[stack.size()]; // real part of klayer
    double ki[]=new double[stack.size()]; // imag part of klayer
    double nr[]=new double[stack.size()];
    double ni[]=new double[stack.size()];
    double n2r[]=new double[stack.size()]; // Re(n^2)
    double n2i[]=new double[stack.size()]; // Im(n^2)
    double t[]=new double[stack.size()]; // thicknesses
    /*
     * Compute wave vector components
     */
    double kxsq=sqr(Math.cos(theta*Math.PI/180.));
    // use in called methods
    ListIterator slist=stack.listIterator(0);
    for(int i=0; i<stack.size(); i++){
        // klayer(i)=sqrt((k*n[i])**2-kx**2)
        layer l=(layer)slist.next();
        nr[i]=l.n.re;
        ni[i]=l.n.im;
        t[i]=l.thick;
        n2r[i]=sqr(l.n.re)-sqr(l.n.im);
        n2i[i]=2*l.n.re*l.n.im;
        double sqr=n2r[i]-kxsq;
        double sqi=n2i[i];
        double phihalf=Math.atan2(sqi, sqr)/2;
        double mag=k*Math.sqrt(Math.sqrt(sqr*sqr+sqi*sqi));
        kr[i]=mag*Math.cos(phihalf);
        ki[i]=mag*Math.sin(phihalf);
    }
    double Rsr=0, Rsi=0, Rpr=0, Rpi=0;
    for(int i=0; i<stack.size()-1; i++){
        // The Fresnel coeeficients are between
```

```
//layer i and layer i+1
double a=kr[i+1]-kr[i];
double b=ki[i+1]-ki[i];
double c=kr[i+1]+kr[i];
double d=ki[i+1]+ki[i];
double den=c*c+d*d;
// fs=(k2-k1)/(k1+k2)
double fsr=(a*c+b*d)/den;
double fsi=(b*c-a*d)/den;
// fp = (n1*k2-n2*k1)/(n2*k1+n1*k2)
double n1k2r=n2r[i]*kr[i+1]-n2i[i]*ki[i+1];
double n1k2i=n2r[i]*ki[i+1]+n2i[i]*kr[i+1];
double n2k1r=n2r[i+1]*kr[i]-n2i[i+1]*ki[i];
double n2k1i=n2r[i+1]*ki[i]+n2i[i+1]*kr[i];
a=n1k2r-n2k1r;
b=n1k2i-n2k1i;
c=n2k1r+n1k2r;
d=n2k1i+n1k2i;
den=c*c+d*d;
double fpr=(a*c+b*d)/den;
double fpi=(b*c-a*d)/den;
// Propagation constants
// C is for propagation through layer i
// C = exp(2*I*k2*t)
double phir=-2*ki[i+1]*t[i+1];
double phii=2*kr[i+1]*t[i+1];
double mag=Math.exp(phir);
double Cr=mag*Math.cos(phii);
double Ci=mag*Math.sin(phii);
// Rs = C*(fs+Rs)/(1+fs*Rs)
a=fsr+Rsr;
b=fsi+Rsi;
c=1+fsr*Rsr-fsi*Rsi;
d=fsr*Rsi+fsi*Rsr;
den=c*c+d*d;
double rr=(a*c+b*d)/den;
double ri=(b*c-a*d)/den;
Rsr=Cr*rr-Ci*ri;
Rsi=Cr*ri+Ci*rr;
// Rp = C*(fp+Rp)/(1+fp*Rp)
a=fpr+Rpr;
b=fpi+Rpi;
```

```
            c=1+fpr*Rpr-fpi*Rpi;
            d=fpr*Rpi+fpi*Rpr;
            den=c*c+d*d;
            rr=(a*c+b*d)/den;
            ri=(b*c-a*d)/den;
            Rpr=Cr*rr-Ci*ri;
            Rpi=Cr*ri+Ci*rr;
        } // for stack elements
        // average, assuming 90% s and 10% p
        return sFrac*(Rsr*Rsr+Rsi*Rsi)+pFrac*(Rpr*Rpr+Rpi*Rpi);
    }

    // reflectance

}
```

A.15 ReflFit.class

```
import hep.aida.*; // in aida.jar import
hep.aida.ref.fitter.FitResult; // in freehep-hep.jar import
java.util.Random; import java.io.*; import
hep.aida.ref.histogram.DataPoint; // in freehep-hep.jar ? import
java.awt.Component; import javax.swing.JOptionPane;

/** Fit the reflectance of a multi-layer stack
 *
 * @author Steve Turley
 *
 */
/*
 * Use the following jar files:
 * JAS3/extensions/aida.jar
 * JAS3/extensions/aida-dev.jar
 * JAS3/extensions/freehep-hep.jar
 * JAS3/lib/freehep-base.jar
 * JAS3/lib/openide-lookup.jar
 * JAS3/lib/jas-plotter.jar
 */

public class ReflFit {

    private IAnalysisFactory af = IAnalysisFactory.create();
    private ITree tree = null;
```

```
private IDataPointSetFactory dpsf = null;
private IFunctionFactory funcF = null;
private IFitFactory fitF = null;
private IFitter fitter = null;
private IDataPointSet dataPointSet = null;
// The following should be unnecessary with real data
private Random r = new Random();
private final double mon = 0.9;
private final double mok = 0.02;
private final double moz = 100.;

/** ReflFit Constructor
 * Fitting routines using the JAS3 framework
 *
 */
public ReflFit() {
    /*
     * The following code implements persistant trees
     * stored on a disk
    try{
        tree= af.createTreeFactory().create(
            "c:/BYU/java/ReflFit.aida","XML",false, true);
    } catch (Exception e){
        System.out.println("IO Exception: "+e.getMessage());
    }
     */
    tree = af.createTreeFactory().create();
    dpsf = af.createDataPointSetFactory(tree);
    funcF = af.createFunctionFactory(tree);
    fitF = af.createFitFactory();
    // fitter = fitF.createFitter("Chi2","uncmin","noClone=true");
    fitter = fitF.createFitter("chi2", "minuit", "noClone=true");
    System.out.println(); // initialize prints unterminated line
}

/** Fit the data to a function
 *
 */
public FitResult fit() {
    /* Fitter returns no parameters before or after
     * fit (listParameterSettings)
     * fitter uses initial values from the function as
```

```
         * a starting point when
         * none are supplied in the fitter call.  Alternatively,
         * the following
         * line works to fit a specific parameter.  This seems
         *to be the case
         * even if the function has not yet been defined.
         * If an invalid parameter
         * is specified, it is ignored without error.
         */
        /* Bounded fits seem to work okay as well.  Again,
         * if the variable is
         * not found, it is ignored.
         */
        // Fit some fake data
        javax.swing.JOptionPane.showMessageDialog(
            null, "Bounds and Constraints not set");
        fitter.fitParameterSettings("Mo n").setBounds(
            mon*0.75, mon*1.25);
        fitter.fitParameterSettings("Mo k").setBounds(
            mok*0.75, mok*1.25);
        fitter.fitParameterSettings("Mo z").setBounds(
            moz*0.75, moz*1.25);

        ReflFunc rfl = new ReflFunc();
        rfl.setParameter("Mo n", mon * (r.nextDouble()/2 + 0.75));
        rfl.setParameter("Mo k", mok * (r.nextDouble()/2 + 0.75));
        rfl.setParameter("Mo z", moz * (r.nextDouble()/2 + 0.75));
        return (FitResult) (fitter.fit(dataPointSet, rfl));
    } // fit()

    /** Set n and k constraints for a film being fit
     *
     * @param f Film that has the constraints (or is fixed)
     */
    private void setNK(Film f){
        setParameter(f.n, f.name+" n");
        setParameter(f.k, f.name+" k");
    }

    /** Set n, k, and z constraints for a film being fit
     *
     * @param f Film that has constraints (or is fixed)
```

```
 */
private void setNKZ(Film f){
    setParameter(f.n, f.name+" n");
    setParameter(f.k, f.name+" k");
    setParameter(f.z, f.name+" z");
}

/** Set constraints (or fixed) for parameter p with
 * parameter string pStr
 *
 * @param p FitParameter for which constraints or fixed
 * property is to be set
 * @param pStr String used for ReflFit to describe parameter
 */
private void setParameter(FitParameter p, String pStr){
    if(p.fixed){
        fitter.fitParameterSettings(pStr).setFixed(true);
    } else {
        fitter.fitParameterSettings(pStr).setFixed(false);
        if(p.constrained){
            fitter.fitParameterSettings(pStr).removeBounds();
            fitter.fitParameterSettings(pStr).setBounds(
                p.min,p.max);
        } else {
            fitter.fitParameterSettings(pStr).removeBounds();
        }
    }
}

/** Fit the data to a function given a mirror stack
 *
 */
public FitResult fit(Mirror m) {
    setNK(m.substrate);
    for(int i=0; i<m.numFilms()-2; i++){
        setNKZ(m.getFilm(i));
    }
    setNK(m.vacuum);
    MirrorFunc rfl = new MirrorFunc(m);
    return (FitResult) (fitter.fit(dataPointSet, rfl));
} // fit()
```

```
/** Create a two dimensional IDataPointSet from random data
 *
 */
public void createData(File cfile){
    // Change the name and plot this second to set title plot
    // Set up my function to calculate fake data (I'll add noise)
    ReflFunc rfl = new ReflFunc();
    PrintStream pout=null;
    try{
        pout = new PrintStream(new FileOutputStream(cfile));
    } catch (FileNotFoundException e){
        e.printStackTrace();
    }
    rfl.setParameter("Mo n", mon);
    rfl.setParameter("Mo k", mok);
    rfl.setParameter("Mo z", moz);
    for (int i = 0; i < 70; i++) {
        double theta = i + 3;
        double[] thetas = { theta };
        final double noiseAmp = 0.01;
        double rval = rfl.value(thetas) + r.nextGaussian()
            * noiseAmp;
        pout.println(theta+" "+rval);
    }
}

/** Create a two dimensional IDataPointSet from data in file
 *
 */
public void data() {
    // Change the name and plot this second to set title plot
    dataPointSet =
        dpsf.create("dataPointSet",
            "two dimensional IDataPointSet", 2);
    dataPointSet.setTitle("My Title");
    // Use to label data in plots

    // Set up my function to calculate fake data (I'll add noise)
    ReflFunc rfl = new ReflFunc();
    rfl.setParameter("Mo n", mon);
    rfl.setParameter("Mo k", mok);
    rfl.setParameter("Mo z", moz);
```

```
        double[] theta = new double[70];
        double[] rval = new double[70];
        double[] error = new double[70];
        for (int i = 0; i < 70; i++) {
            theta[i] = i + 3;
            double[] thetas = { theta[i] };
            final double noiseAmp = 0.01;
            rval[i] = rfl.value(thetas) + r.nextGaussian() * noiseAmp;
            error[i] = noiseAmp;
        }
        dataPointSet.setCoordinate(0, theta, error);
        dataPointSet.setCoordinate(1, rval, error);
    } // data()

    /** Create a two dimensional IDataPointSet from data in file
     *
     */
    public void data(File file, double err, boolean absolute,
        boolean ref) {
        // Change the name and plot this second to set title plot
        dataPointSet =
            dpsf.create("dataPointSet", "two dimensional
                IDataPointSet", 2);
        dataPointSet.setTitle(file.getName());
        // Use to label data in plots

        // Set up my function to calculate fake data (I'll add noise)
        BufferedReader in = null;
        String line = null;
        // Open reader and get first line
        try {
            in = new BufferedReader(new FileReader(file));
            line = in.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
        double[] error = new double[2];
        error[0] = .01;
        error[1] = err; // use constant error for now
        while (line != null) {
            // Split into tab-delimited tokens
            if (line.charAt(0) == '#'){
```

```
            try {
                line = in.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            continue;
        }
        String[] stok = line.split("\\s+");
        // Uses one or more white space characters as delims
        // Assume all lines are okay with two strings in them
        int loc=0;
        if(stok[0].length()==0) loc++;
        // skip leading white space
        double lambda = Double.parseDouble(stok[loc++]);
        double rv = Double.parseDouble(stok[loc++]);
        double[] pair = new double[2];
        pair[0] = lambda;
        pair[1] = rv;
        if (!absolute) {
            error[1] = rv * err;
        }
        DataPoint dpt = new DataPoint(pair, error);
        dataPointSet.addPoint(dpt);
        try {
            line = in.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
} // data(file)

/** Scale errors in data by factor
 *
 * @param factor double value by which to uniformly
 * scale errors in data
 */
public void scaleErrors(double factor){
    dataPointSet.scaleErrors(Math.sqrt(factor));
}

public void plot(FitResult result){
    IPlotter plotter =
```

```
            af.createPlotterFactory().create("ReflFit.java plot");
        plotter.createRegions();
        // Change styles to plot only points
        IPlotterStyle pstyle = plotter.region(0).style();
        IDataStyle dstyle = pstyle.dataStyle();
        dstyle.markerStyle().setColor("orange");
        dstyle.setParameter("showHistogramBars", "false");
        dstyle.setParameter("showErrorBars", "false");
        dstyle.setParameter("showDataPoints", "true");
        plotter.region(0).plot(dataPointSet);
        plotter.show();
        plotter.region(0).plot(result.fittedFunction());
        plotter.region(0).setTitle("Refl Fit");
    } // plot(result)

    /** Print fit quality
     * @param result FitResult returned from fit()
     * @param frame Component in which to display dialog with results
     * @return true if caller should update stack with fitted values
     */
    public boolean print(FitResult result, Component frame){
        double fitp[] = result.fittedParameters();
        double errs[] = result.errors();
        String[] names=result.fittedParameterNames();
        String[] message = new String[2 + fitp.length];
        message[0] = "\nChi2=" + result.quality();
        for (int i = 0; i < fitp.length; i++) {
            message[i + 1] = names[i]+": "+fitp[i];
            if(errs[i]!=0.0)
                message[i+1] = message[i+1]+ " +/- " + errs[i];
        }
        message[fitp.length+1]="\nUpdate stack with fit?";
        int status=JOptionPane.showConfirmDialog(frame, message,
            "Fit Results", JOptionPane.YES_NO_OPTION);
        return status==JOptionPane.YES_OPTION;
    } // print(result, frame)

    /** update stack with data from last fit
     * @param r FitResult returned by ReflFit.fit(Mirror)
     * @param m Mirror with stack information
     */
    public void updateStack(FitResult r, Mirror m){
```

```
        getNK(r, m.substrate);
        for(int i=0; i<m.numFilms()-2; i++){
            getNKZ(r, m.getFilm(i));
        }
        getNK(r, m.vacuum);
    } // updateStack

    private void getNK(FitResult r, Film f){
        String[] name=r.fittedParameterNames();
        f.n.value=r.fittedParameter(f.name+" n");
        f.k.value=r.fittedParameter(f.name+" k");
    }

    private void getNKZ(FitResult r, Film f){
        f.n.value=r.fittedParameter(f.name+" n");
        f.k.value=r.fittedParameter(f.name+" k");
        f.z.value=r.fittedParameter(f.name+" z");
    }

    /**
     * Print fit quality
     * @param result FitResult returned from fit()
     */
    public void print(FitResult result){
        System.out.println("\nChi2=" + result.quality());
        double fitp[] = result.fittedParameters();
        double errs[] = result.errors();
        for (int i = 0; i < fitp.length; i++) {
            System.out.println(fitp[i] + " +/- " + errs[i]);
        }
    } // print(result)

    public static void main(String[] argv) {
        ReflFit rfit = new ReflFit();
        rfit.data();
        FitResult result=rfit.fit();
        rfit.plot(result);
        rfit.print(result);
    }
}
```

A.16 Complex.class

```
/*
 * Complex.java
 *
 * Created on November 13, 2003, 5:36 AM
 */

/** Complex numbers class providing storage and arithmetic.
 *
 * Validated against Maple 11/13/03
 * @author Steve Turley
 */
public class Complex extends Object {

    /** Imaginary part */
    public double re = 0;

    /** Real part */
    public double im = 0;

    /** Creates a new instance of Complex */
    public Complex() {
        /* Since variables are already initiated, I really
         * don't have to do
         * anything here.  At some point, it might be
         * more efficient to
         * put explicit initialization here rather than in
         * the declarations
         */
    }

    /** Creates a new instance of Complex with real part x
     * and imaginary part y.
     */
    public Complex(double x, double y) {
        re=x; im=y;
    }

    /** Creates a new instance of complex with a real part of x and
     * an imaginary part of 0.  This is essentially a cast
     * from double to Complex.
     */
    public Complex(double x) {
```

```
    re=x;
}

/** Returns true is this number is equal to the argument. */
public boolean equals(Complex obj) {
    return (re==obj.re && im==obj.im);
}

/** converts this number to a string */
public String toString() {
    return "("+re+","+im+")";
}

/** returns this number plus the argument */
public Complex plus(Complex y) {
    return new Complex(re+y.re,im+y.im);
}

/** returns this number minus the argument */
public Complex minus(Complex y) {
    return new Complex(re-y.re, im-y.im);
}

/** returns the additive inverse of this number*/
public Complex neg() {
    return new Complex(-re,-im);
}

/** returns this number times the argument */
public Complex times(Complex y) {
    return new Complex(re*y.re-im*y.im,re*y.im+im*y.re);
}

/** returns this number divided by the argument */
public Complex over(Complex y) {
    double den=y.re*y.re+y.im*y.im;
    return new Complex((re*y.re+im*y.im)/den,(
        im*y.re-re*y.im)/den);
}

/** Returns the complex exponential of this number. */
public Complex exp() {
```

```
        double ex=Math.exp(re);
        return new Complex(ex*Math.cos(im),ex*Math.sin(im));
    }

    /** Returns the complex conjugate of this number. */
    public Complex conj() {
        return new Complex(re, -im);
    }

    /** Returns the absolute value (magnitude) of this number. */
    public double mag() {
        return Math.sqrt(re*re+im*im);
    }

    /** Multiply the complex number by its double argument and
     * return the result.
     * @param y double multiplied by the Complex number
     */
    public Complex times(double y) {
        return new Complex(re*y, im*y);
    }

    /** subtracts its argument form the complex number and
     * returns the result.
     * @param y subtracted from the complex number
     */
    public Complex minus(double y) {
        return new Complex(re-y, im);
    }

    /** Add a double to a Complex */
    public Complex plus(double x) {
        return new Complex(re+x, im);
    }

    /** Returns the (positive) square root. Validated with Maple. */
    public Complex sqrt() {
        double coef=mag();
        double phi=Math.atan2(im, re);
        coef=Math.sqrt(coef);
        phi=phi/2;
        return new Complex(coef*Math.cos(phi),coef*Math.sin(phi));
```

```
    }

}
```

A.17 Matrix.class

```
/*
 * Created on Apr 1, 2004
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;
 * Code and Comments
 */

/**
 * @author farnsworth
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;
 * Code and Comments
 */
public class Matrix extends Object{

    //components of the matrix
    public Complex A11 = new Complex(0);
    public Complex A12 = new Complex(0);
    public Complex A21 = new Complex(0);
    public Complex A22 = new Complex(0);

    //*Creates a new instance of Matrix
    public Matrix(){
    }

    public Matrix (int x11,int x12,int x21,int x22){
        A11=new Complex(x11);
        A12=new Complex(x12);
        A21=new Complex(x21);
        A22=new Complex(x22);
    }

    //Creates a new instance of Matrix with components B11,B12,B21,B22
    public Matrix(Complex B11,Complex B12,Complex B21,Complex B22){
        A11=B11;
```

```
        A12=B12;
        A21=B21;
        A22=B22;
    }

    //returns the Matrix as a string
    public String toString(){
        return "["+A11.re+" "+A12.re+"],["+A21.re+" "+A22.re+"] ,
             i["+A11.im+" "+A12.im+"],["+A21.im+" "+A22.im+"]";
    }

    //returns this matrix plus the matrix B
    public Matrix plus(Matrix B){
        return new Matrix(A11.plus(B.A11),A12.plus(B.A12),A21
            .plus(B.A21),A22.plus(B.A22));
    }

    //returns this matrix minus the matrix B
    public Matrix minus(Matrix B){
        return new Matrix(A11.minus(B.A11),A12.minus(B.A12),
            A21.minus(B.A21),A22.minus(B.A22));
    }

    //returns this matrix times the scalar y
    public Matrix multscalar(double y){
        Complex ycomplex = new Complex(y);
        return new Matrix(A11.times(ycomplex),A12.times(ycomplex),
            A21.times(ycomplex),A22.times(ycomplex));
    }

    //returns this matrix times the matrix B
    public Matrix times(Matrix B){
        return new Matrix(
            A11.times(B.A11).plus(A12.times(B.A21)),
            A11.times(B.A12).plus(A12.times(B.A22)),
            A21.times(B.A11).plus(A22.times(B.A21)),
            A21.times(B.A12).plus(A22.times(B.A22)));
    }

    //returns the determinant of this matrix
    public Complex det(){
        return A11.times(A22).minus(A12.times(A21));
```

```
    }

    //returns the inverse of this matrix
    public Matrix inv(){
        return new Matrix(
            A22.over(A11.times(A22).minus(A12.times(A21))),
            A12.neg().over(A11.times(A22).minus(A12.times(A21))),
            A21.neg().over(A11.times(A22).minus(A12.times(A21))),
            A11.over(A11.times(A22).minus(A12.times(A21))));
    }
}
```

Appendix B

AFM Tip Source Code

This is the source code used to simulate an AFM tip dragging over a surface. The program gaussianRandomness.m approximates the surface as an array of gaussian random numbers around y=0 of width 1 and uniformRandomness.m approximates the surface as an array of uniform random numbers between 1 and -1. SeveralRuns2.m averages the rms roughnesses and power spectral densities of N random surfaces. SemiCorrelated.m multiplies each point of a random surface by a gaussian of with a specified width, allowing the user to change the correlation length of the surface. SemiCorrelatedSeveralRuns.m averages the rms roughnesses and power spectral densities of N semi-correlated surfaces.

B.1 gaussianRandomness.m

```
%begin gaussian_randomness.m
%Generates a random surface (Gaussian randomness),
%creates another surface
%by tracing an AFM tip over it, finds the rms roughnesses
%and the power spectral
%densities of the two surfaces.
clear;close all; xfinal=50; dx=.5; x=0:dx:xfinal-dx;
s=randn(1,xfinal/dx); %actual surface
for j=1:xfinal/dx
    t=(x-j*dx).^2+2; %the tip (y=x^2)
    d=t-s;
    [dmin,imin]=min(d);
```

```
    xhit(j)=x(imin);
    yhit(j)=s(imin);
    xtip(j)=j*dx;
    ytip(j)=yhit(j)-(xhit(j)-j*dx).^2;%assuming tip is a parabola
end

%find the rms roughness of the real surface and the tip surface
p1=0;q1=0;p2=0;q2=0; for m=1:length(s)
    p1=p1+s(m);
    p2=p2+s(m)^2;
    q1=q1+ytip(m);
    q2=q2+ytip(m)^2;
end surfaceAvg=p1/length(s); tipAvg=q1/length(ytip);
surfaceRMS=sqrt(p2/length(s))-surfaceAvg
tipRMS=sqrt(q2/length(ytip))-tipAvg

subplot(2,1,1) plot(x,s,'r-',xtip,ytip,'b-') ylabel('nm')
a=sprintf('RMS roughness surface = %i nm, tip = %i nm',
    surfaceRMS,tipRMS); title(a)

%power spectral density
g=fft(s-surfaceAvg); P1=abs(g).^2; h=fft(ytip-tipAvg);
P2=abs(h).^2; dw=2*pi/xfinal; w=0:dw:2*pi/dx-dw; subplot(2,1,2)
plot(w,P1,'r-',w,P2,'b--') xlabel('\omega') ylabel('P(\omega)')
title('Power Spectral density') axis([0 max(w)/2,0 max(P2)]);
%end gaussian_randomness.m
```

B.2 uniformRandomness.m

```
%begin uniform_randomness.m
clear;close all;
%n=highest point, dx=spacing of points
xfinal=50; dx=.5;
%creates an array x
x=0:dx:xfinal-dx;
%surface
s=2.*(rand(1,xfinal/dx)-.5); for j=1:xfinal/dx
    %creates the tip and moves the tip over dx on
    %every repetition
    t=(x-j*dx).^2+2;
    %subtract the tip from the surface
    d=t-s;
```

```
    %find the minimum of the array
    [dmin,imin]=min(d);
    %this is where the tip hit the surface
    xhit(j)=x(imin);
    yhit(j)=s(imin);
    %this is the position of the tip of the tip
    xtip(j)=j*dx;
    %assuming tip is a parabola
    ytip(j)=yhit(j)-(xhit(j)-j*dx).^2;
end

%find the rms roughness of the real surface and the tip surface
p1=0;q1=0;p2=0;q2=0; for m=1:length(s)
    p1=p1+s(m);
    p2=p2+s(m)^2;
    q1=q1+ytip(m);
    q2=q2+ytip(m)^2;
end surfaceAvg=p1/length(s) tipAvg=q1/length(ytip)
surfaceRMS=sqrt(p2/length(s))-surfaceAvg
tipRMS=sqrt(q2/length(ytip))-tipAvg

subplot(2,1,1) plot(x,s,'r-',xtip,ytip,'b--') xlabel('nm')
ylabel('nm')
a=sprintf('RMS roughness surface = %i nm, tip = %i nm',
    surfaceRMS,tipRMS)
title(a)

%power spectral density
g=fft(s-surfaceAvg); P1=abs(g).^2; h=fft(ytip-tipAvg);
P2=abs(h).^2; dw=2*pi/xfinal; w=0:dw:2*pi/dx-dw; subplot(2,1,2)
plot(w,P1,'r-',w,P2,'b--') xlabel('\omega') ylabel('P(\omega)')
title('Power Spectral density') axis([0 max(w)/2,0 max(P1)]);
%end uniform_randomness.m
```

B.3 SeveralRuns2.m

```
%begin SeveralRuns2.m
%Same thing as Gaussian_randomness.m but averages RMS
%roughness and PSD functions over several runs.
clear;close all; xfinal=50; dx=.5; x=0:dx:xfinal-dx; N=4000;
C1=0;C2=0; sa=0;ta=0; for b=1:N
    s=randn(1,xfinal/dx); %actual surface
```

```
    for j=1:xfinal/dx
        t=(x-j*dx).^2+2; %the tip (y=x^2)
        d=t-s;
        [dmin,imin]=min(d);
        xhit(j)=x(imin);
        yhit(j)=s(imin);
        xtip(j)=j*dx;
        ytip(j)=yhit(j)-(xhit(j)-j*dx).^2;
        %assuming tip is a parabola
    end

    %find the rms roughness of the real surface and the tip surface
    p1=0;q1=0;p2=0;q2=0;
    for m=1:length(s)
        p1=p1+s(m);
        p2=p2+s(m)^2;
        q1=q1+ytip(m);
        q2=q2+ytip(m)^2;
    end
    surfaceAvg=p1/length(s);
    tipAvg=q1/length(ytip);
    surfaceRMS=sqrt(p2/length(s))-surfaceAvg;
    sa=sa+surfaceRMS;
    tipRMS=sqrt(q2/length(ytip))-tipAvg;
    ta=ta+tipRMS;

    %power spectral density
    g=fft(s-surfaceAvg);
    P1=abs(g).^2;
    C1=C1+P1;
    h=fft(ytip-tipAvg);
    P2=abs(h).^2;
    C2=C2+P2;
end

C1=C1./b;
C2=C2./b;

sa=sa/b;
ta=ta/b;

dw=2*pi/xfinal;
```

```
w=0:dw:2*pi/dx-dw;
plot(w,C1,'r-',w,C2,'b-')
xlabel('\omega')
ylabel('P(\omega)')
a=sprintf('PSD functions of surface and tip averaged over %i
    runs:\navg surface RMS = %i nm avg tip RMS = %i nm',N,sa,ta);
title(a)
axis([0 max(w)/2,0 max(C2)]);
%end SeveralRuns.m
```

B.4 SemiCorrelated.m

```
%begin SemiCorrelated.m
clear;close all;
xfinal=20; dx=.2;
x=0:dx:xfinal-dx;
s=rand(1,xfinal/dx);
width=2;
for o=1:length(x)
    gauss=1/(sqrt(2*pi)*width)*exp(-(x-(o-1)*dx).^2/(2*width^2));
    b=gauss.*s;
    surf(o)=b(o);
end
for j=1:xfinal/dx
    t=(x-j*dx).^2+2; %the tip (y=x^2)
    d=t-surf;
    [dmin,imin]=min(d);
    xhit(j)=x(imin);
    yhit(j)=surf(imin);
    xtip(j)=j*dx;
    ytip(j)=yhit(j)-(xhit(j)-j*dx).^2;
    %assuming tip is a parabola
end

p1=0;q1=0;p2=0;q2=0;
for m=1:length(surf)
    p1=p1+surf(m);
    p2=p2+surf(m)^2;
    q1=q1+ytip(m);
    q2=q2+ytip(m)^2;
end
surfaceAvg=p1/length(surf);
```

```
tipAvg=q1/length(ytip);
surfaceRMS=sqrt(p2/length(surf))-surfaceAvg;
tipRMS=sqrt(q2/length(ytip))-tipAvg;

subplot(2,1,1)
plot(x,surf,'r-',xtip,ytip,'b--')
axis([0 xfinal min(surf) max(ytip)])
xlabel('nm')
ylabel('nm')
a=sprintf('RMS roughness surface = %0.2g nm, tip =
    %0.2g nm',surfaceRMS,tipRMS);
title(a)

%power spectral density
g=fft(surf-surfaceAvg);
P1=abs(g).^2;
h=fft(ytip-tipAvg);
P2=abs(h).^2;
dw=2*pi/xfinal;
w=0:dw:2*pi/dx-dw;
subplot(2,1,2)
plot(w,P1,'r-',w,P2,'b--')
xlabel('\omega')
ylabel('P(\omega)')
title('Power Spectral density')
axis([0 max(w)/2,0 max(P2)]);
%end SemiCorrelated.m
```

B.5 SemiCorrelatedSeveralRuns.m

```
%begin SemiCorrelated_SeveralRuns.m
clear;close all;
xfinal=20; dx=.1;
x=0:dx:xfinal-dx;
N=input(' Number of runs to average over - ');
sR=0;tR=0;
P1avg=0;P2avg=0;
width=2;
for n=1:N
    s=rand(1,xfinal/dx);
    for o=1:length(x)
        gauss=1/(sqrt(2*pi)*width)*exp(-(x-(o-1)*dx).^2/
```

```
            (2*width^2));
        b=gauss.*s;
        surf(o)=b(o);
    end
    for j=1:xfinal/dx
        t=(x-j*dx).^2+2; %the tip (y=x^2)
        d=t-surf;
        [dmin,imin]=min(d);
        xhit(j)=x(imin);
        yhit(j)=surf(imin);
        xtip(j)=j*dx;
        ytip(j)=yhit(j)-(xhit(j)-j*dx).^2;
        %assuming tip is a parabola
    end

    p1=0;q1=0;p2=0;q2=0;
    for m=1:length(surf)
        p1=p1+surf(m);
        p2=p2+surf(m)^2;
        q1=q1+ytip(m);
        q2=q2+ytip(m)^2;
    end
    surfaceAvg=p1/length(surf);
    tipAvg=q1/length(ytip);
    surfaceRMS=sqrt(p2/length(surf))-surfaceAvg;
    tipRMS=sqrt(q2/length(ytip))-tipAvg;

    sR=sR+surfaceRMS;
    tR=tR+tipRMS;

    %power spectral density
    g=fft(surf-surfaceAvg);
    P1=abs(g).^2;
    h=fft(ytip-tipAvg);
    P2=abs(h).^2;

    P1avg=P1avg+P1;
    P2avg=P2avg+P2;
end

sR=sR/N;
tR=tR/N;
```

```
P1avg=P1avg./N;
P2avg=P2avg./N;

dw=2*pi/xfinal;
w=0:dw:2*pi/dx-dw;

plot(w,P1avg,'r-',w,P2avg,'b--')
xlabel('\omega')
ylabel('P(\omega)')
a=sprintf('\\sigma=%0.2g, N=%0.2g, avg RMS surface =
    %0.2g nm, tip = %0.2g nm',width,N,sR,tR);
title(a)
axis([0 max(w)/2,0 max(P1avg)]);
%end SemiCorrelated_SeveralRuns.m
```

Appendix C

Representative Fits

Here are some representative fits of reflectance and transmittance data.

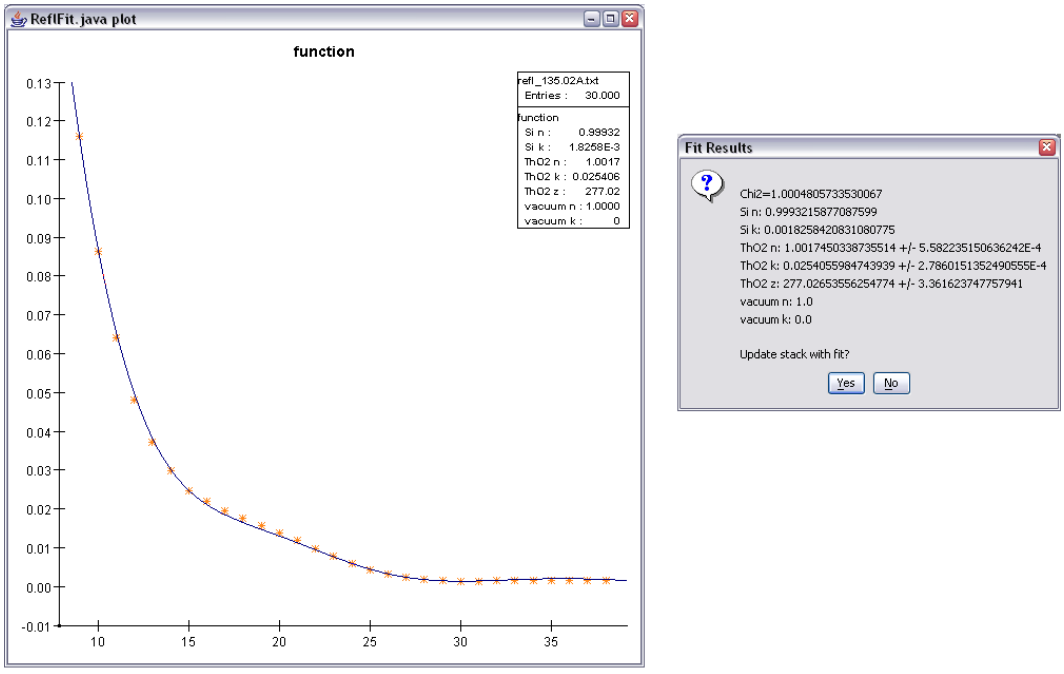

Figure C.1: Reflectance of ThO_2 on silicon at 135 Å. The thickness of the ThO_2 layer was constrained to be between 220 and 280 Å.

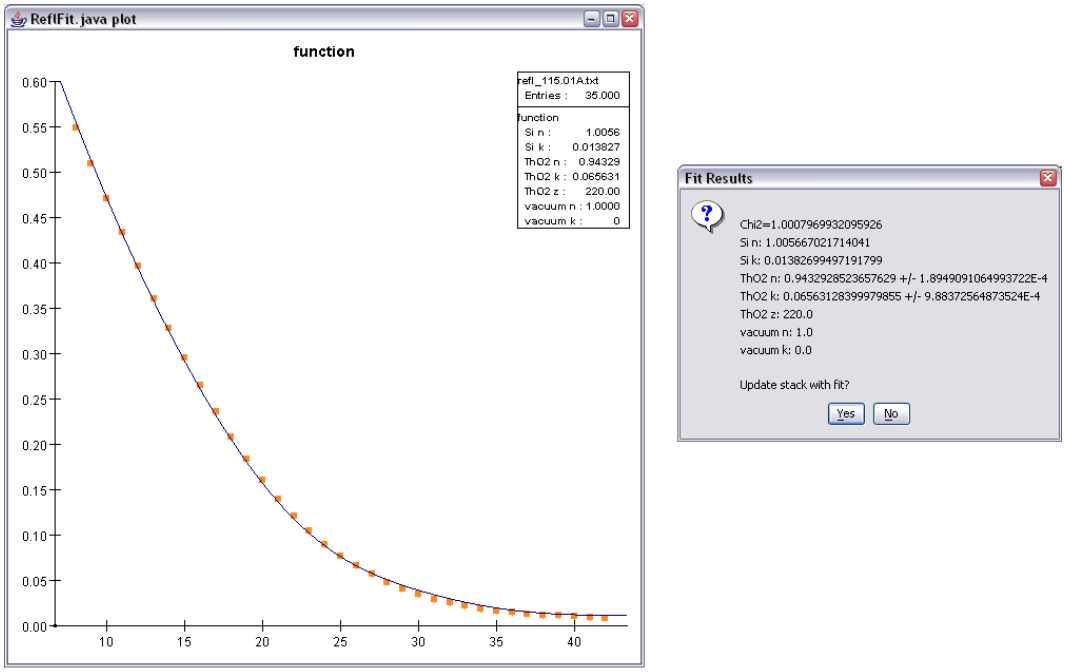

Figure C.2: Reflectance of ThO_2 on silicon at 115 Å. The thickness of the ThO_2 layer was fixed to be between 220 Å.

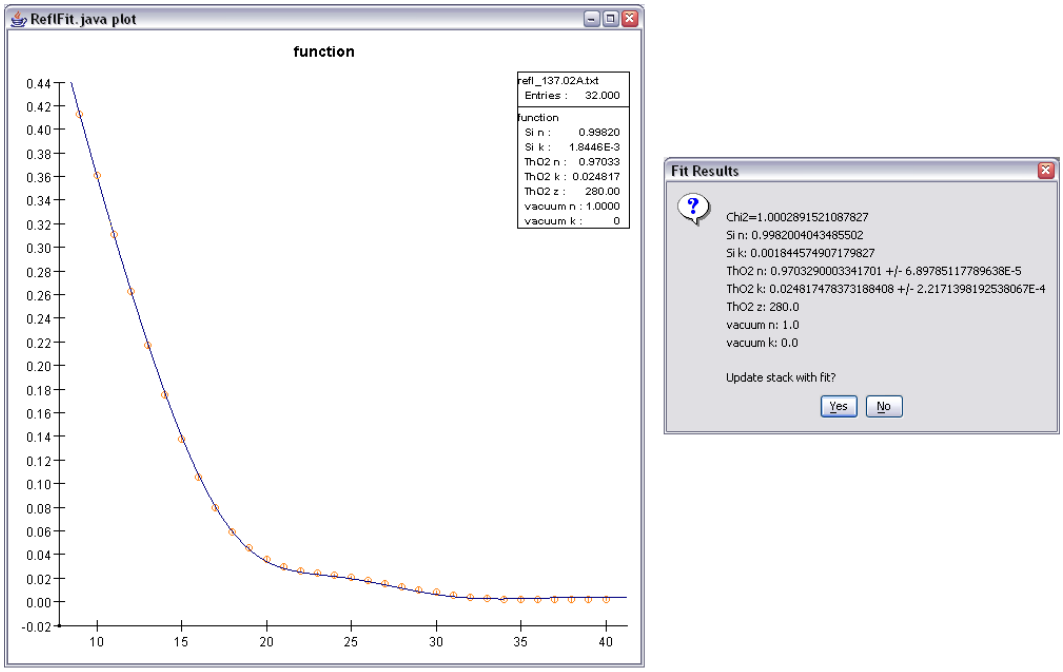

Figure C.3: Reflectance of ThO_2 on silicon at 137 Å. The thickness of the ThO_2 layer was fixed to be between 280 Å.

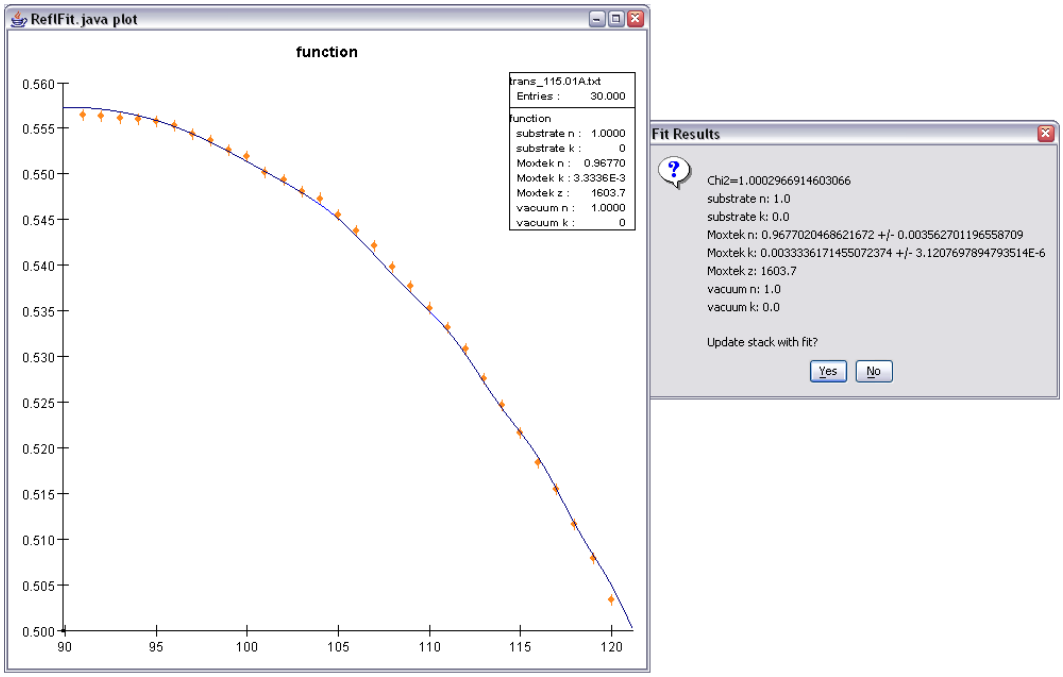

Figure C.4: Transmittance of polyimide at 115 Å. The thickness of the film was fixed to be 1603.7 Å.

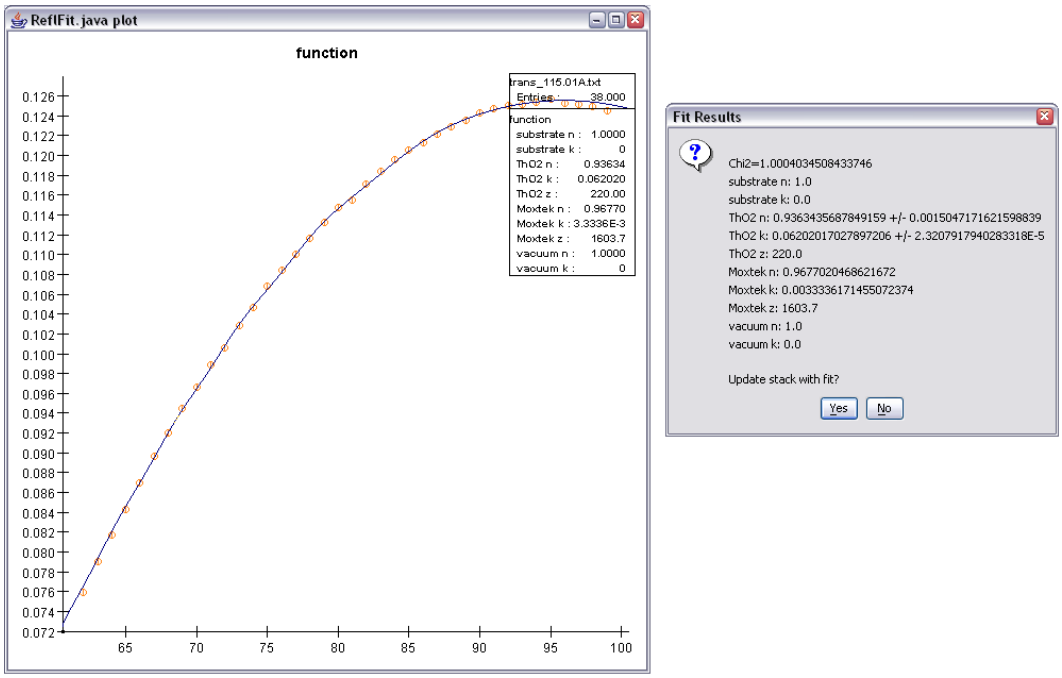

Figure C.5: Transmittance of ThO_2 on polyimide at 115 Å. The thickness of the ThO_2 layer was fixed to be 220 Å.

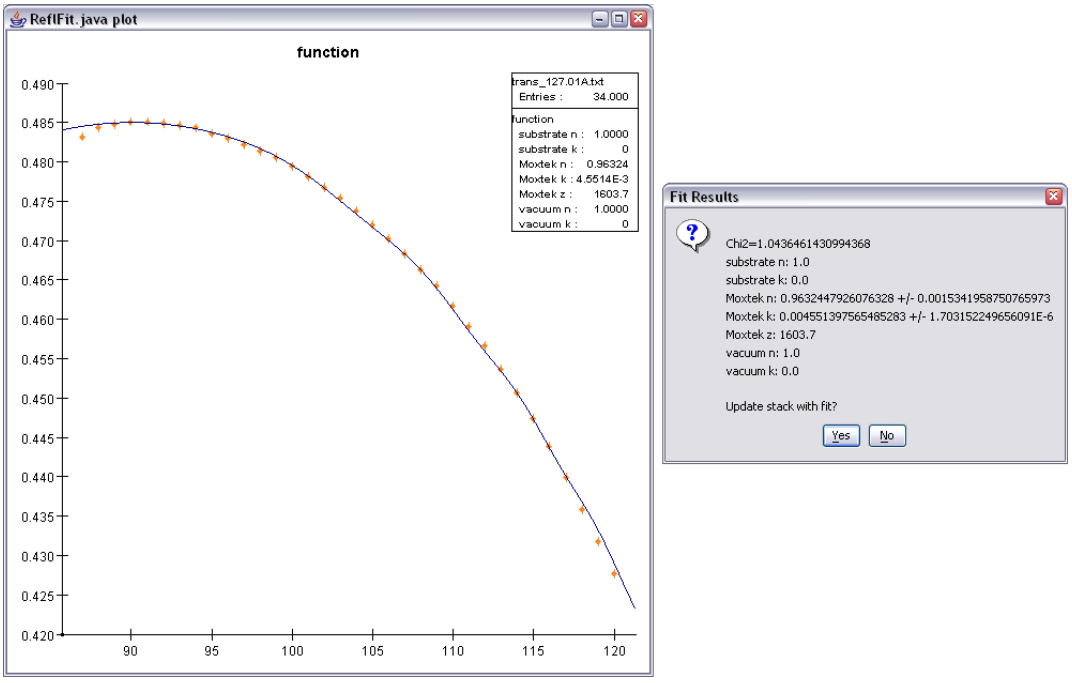

Figure C.6: Transmittance of polyimide at 127 Å. The thickness of the film was fixed to be 1603.7 Å.

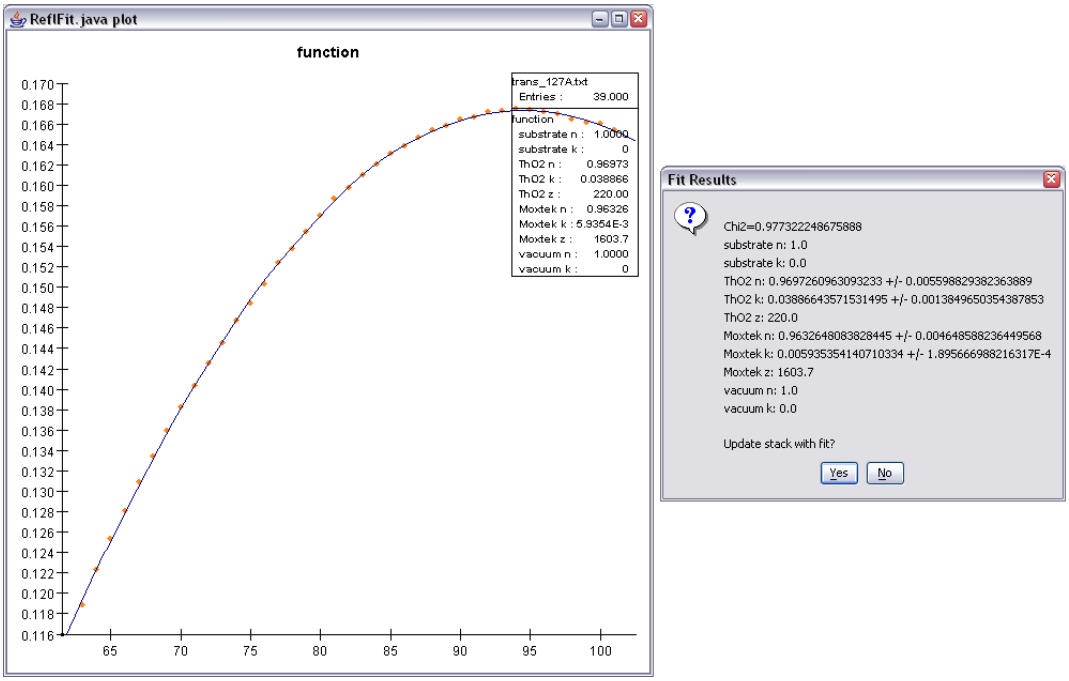

Figure C.7: Transmittance of ThO_2 on polyimide at 127 Å. The thickness of the ThO_2 layer was fixed to be 220 Å.

Bibliography

[1] Jedediah Edward Jensen Johnson, *Thorium-based Mirrors for High Reflectivity in the EUV*, Brigham Young University, Provo UT, 2004.

[2] http://www-cxro.lbl.gov/, March 13, 2005.

[3] David J. Griffiths, *Introduction to Electrodynamics, Third Edition*, Prentice Hall, Upper Saddle River, NJ, 1999.

[4] Eberhard Spiller, *Soft X-ray Optics*, SPIE Optical Engineering Press, Bellingham, WA, 1994.

[5] L. G. Parratt, Phys. Rev. **95**, 2 (1954).

[6] V. G. Kohn, Phys. Stat. Sol. (b) **187**, 61 (1995).

[7] F. James, *MINUIT, Function Minimization and Error Analysis*, CERN Program Library, Geneva, Switzerland, 1998.

[8] V. Holy, J. Kubena, and I. Ohlidal, Phys. Rev. B **47**, 23 (1993).

[9] L. Nevot, B. Pardo, and J. Corno, Revue Phys. Appl. **23**, (1988).

[10] P. Debye, Verh. D. Deutsch. Phys. Ges. **15**, 22 (1913).

[11] P. Croce and L. Nevot, J. De Physique Appliquee **11**, 5 (1976).

[12] http://www.moxtek.com/, March 13, 2005.

[13] http://www-als.lbl.gov/als/, March 13, 2005.