NIST SPECIAL PUBLICATION 400-83

U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology

Semiconductor Measurement Technology:

A Software Program for Aiding the Analysis of Ellipsometric Measurements, Simple Models

J. F. Marchiando

The National Institute of Standards and Technology[1] was established by an act of Congress on March 3, 1901. The Institute's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Institute conducts research to assure international competitiveness and leadership of U.S. industry, science and technology. NIST work involves development and transfer of measurements, standards and related science and technology, in support of continually improving U.S. productivity, product quality and reliability, innovation and underlying science and engineering. The Institute's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, the National Computer Systems Laboratory, and the Institute for Materials Science and Engineering.

The National Measurement Laboratory

Provides the national system of physical and chemical measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; provides advisory and research services to other Government agencies; conducts physical and chemical research; develops, produces, and distributes Standard Reference Materials; provides calibration services; and manages the National Standard Reference Data System. The Laboratory consists of the following centers:

- Basic Standards[2]
- Radiation Research
- Chemical Physics
- Analytical Chemistry

The National Engineering Laboratory

Provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

- Computing and Applied Mathematics
- Electronics and Electrical Engineering[2]
- Manufacturing Engineering
- Building Technology
- Fire Research
- Chemical Engineering[3]

The National Computer Systems Laboratory

Conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Laboratory consists of the following divisions:

- Information Systems Engineering
- Systems and Software Technology
- Computer Security
- Systems and Network Architecture
- Advanced Systems

The Institute for Materials Science and Engineering

Conducts research and provides measurements, data, standards, reference materials, quantitative understanding and other technical information fundamental to the processing, structure, properties and performance of materials; addresses the scientific basis for new advanced materials technologies; plans research around cross-cutting scientific themes such as nondestructive evaluation and phase diagram development; oversees Institute-wide technical programs in nuclear reactor radiation research and nondestructive evaluation; and broadly disseminates generic technical information resulting from its programs. The Institute consists of the following divisions:

- Ceramics
- Fracture and Deformation[3]
- Polymers
- Metallurgy
- Reactor Radiation

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Gaithersburg, MD 20899.
[2]Some divisions within the center are located at Boulder, CO 80303.
[3] Located at Boulder, CO, with some elements at Gaithersburg, MD.

Semiconductor Measurement Technology:

A Software Program for Aiding the Analysis of Ellipsometric Measurements, Simple Models

J. F. Marchiando

Semiconductor Electronics Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

July 1989

NOTE: As of 23 August 1988, the National Bureau of Standards (NBS) became the National Institute of Standards and Technology (NIST) when President Reagan signed into law the Omnibus Trade and Competitiveness Act.

U.S. DEPARTMENT OF COMMERCE, Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, Raymond G. Kammer, Acting Director

Library of Congress Catalog Card Number: 89–600743
National Institute of Standards and Technology Special Publication 400-83
Natl. Inst. Stand. Technol. Spec. Publ. 400-83, 252 pages (July 1989)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1989

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402–9325

Table of Contents

Semiconductor Measurement Technology:

A Software Program for Aiding the Analysis of Ellipsometric Measurements, Simple Models

J. F. Marchiando
Semiconductor Electronics Division
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

MAIN1 is a software program for aiding the analysis of ellipsometric measurements. MAIN1 consists of a suite of routines written in FORTRAN that are used to invert the standard reflection ellipsometric equations for simple systems. Here, a system is said to be simple if the solid material sample may be adequately characterized by models which assume at least the following: (1) materials are nonmagnetic; (2) samples exhibit depth-dependent optical properties, such as one with layered or laminar structure atop a substrate that behaves like a semi-infinite half-space; (3) layers are flat and of uniform thickness; and (4) the dielectric function within each layer/substrate is isotropic, homogeneous, local, and linear. Each layer is characterized in part by a thickness (z), while the optical properties for a given material and wavelength are expressed in terms of a refractive index (n) and extinction coefficient (k). The ellipsometric equations are formulated as a standard damped nonlinear least-squares problem and then solved by an iterative method when possible. Estimates of the uncertainties associated with assigning numerical values to the model parameters are calculated as well.

Key words: ellipsometry; FORTRAN; measurement; modeling; program; sensitivity; software; uncertainty.

1. Introduction

In general, when linearly polarized light is incident on a flat surface, it becomes elliptically polarized upon reflection. Ellipsometry involves measuring this induced change in polarization. The fundamental problem is then to understand those properties which characterize the material medium and are able to induce this measured change in polarization. MAIN1 is a program for aiding the analysis of ellipsometric measurements. MAIN1 consists of a suite of routines written in FORTRAN* that are used to invert a standard set of reflection ellipsometry equations. The systems under consideration here involve those containing flat, thin, solid films atop a substrate. The program has been used in modeling systems as basic as those involving thermally grown oxide films atop silicon and as complicated as those involving SIMOX (Separation by IMplanted OXygen).

A system is said to be simple if the solid material sample may be adequately characterized by models which assume at least the following: (1) materials are nonmagnetic; (2) samples exhibit depth-dependent optical properties, such as one with layered or laminar structure atop a substrate that behaves like a semi-infinite half-space; (3) layers are flat and of uniform thickness; and (4) the dielectric function within each layer/substrate is isotropic, homogeneous, local, and linear. Such approximations are well known and documented [1–7].

In general, the model assumes three kinds or types of parameters: each layer is characterized in part by a thickness (z), while the optical properties of the media (ambient, layers, substrate) are expressed in terms of a scalar refractive index (n) and extinction coefficient (k) at some given wavelength of light. With only three model parameters to characterize each layer, a sample may be readily assembled layer by layer atop the

---

* Disclaimer: Certain commerical equipment, instruments, materials, or software products are identified by name in this document in order to adequately specify the experimental procedure or software development. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials, equipment, or software products identified are necessarily the best available for the intended purpose.

substrate. Since the natural parameters of the formulation include the thickness and dielectric function, i.e., the model parameters themselves, it is straightforward to analyze distinct samples collectively. Thus, the program is said to have a multiple-sample capability. Due to the method of indexing the model parameters used in the program, it is possible for two distinct layers (media, thickness) to have some model parameter in common, e.g., distinct media but common thickness or vice versa. Also, it is possible for two distinct samples to have model parameters in common, e.g., common layer (media, thickness). Such cases may arise when analyzing samples that differ in thickness resulting from some etch-back process. Thus, the program is said to allow coupling between samples and/or layers.

Since the model parameters contain the optical properties themselves, it is important to note that they involve no further dependences. No provision is made to incorporate distinct physical models into the expression of the dielectric function, e.g., the effective medium approximation.

The phenomenological models and methods of solution are classical. The ellipsometric equations are formulated as a standard nonlinear, damped least-squares problem which is solved by an iterative method appropriate for linear systems. Following the convergence of the iterations to a *good* solution, i.e., within the resolution of the measurement by the instrumentation, simple estimates of the uncertainties associated with ascribing numerical values to the model parameters are calculated as well.

Regarding the uniqueness of the calculated solution which is found by minimizing a residual in the least-squares sense, it is important to realize that multiple or pseudo solutions may exist, as well as correlation among the modeling parameters. Consequently, while the program does serve as a useful tool in finding solutions, the responsibility of assessing the consistency and appropriateness of the results remains entirely dependent on the user. The collection of modules or subroutines is designed to run in batch mode, as opposed to interactive mode, and is suitable for calculations involving only the physical models mentioned earlier.

Although no graphic displays are provided while running the program, provision is made for selecting specific sets of data for output. These data are written to a file in a format amenable to the generation of graphics later. Thus, graphics capability is dependent

upon the resource library available at the local computing site. Actually, any one of several software graphics packages would be more than adequate. But for convenience, as well as for completeness, this document does refer to the NCAR software graphics package [27,28]. This is mentioned briefly in section 4.1.3. The section also discusses the use of the LINPACK library [17], which is public domain software.

This report presents a brief overview of the theory and methods of solution and provides a brief manual of operations or guide for the user to the options and capabilities of the software program. The model describing the scattering of light is presented in section 2. The methods of inverting the ellipsometric equations and estimating the uncertainties associated with knowing the modeling parameters are discussed in section 3. Section 4 presents the manual of operations for the software package. This includes an overview of considerations important in using the software package. A brief outline of the internal organization of the subroutines and calling sequences is provided for orientational purposes. This outline includes a short listing of the parameters which specify upper bounds for the necessary allocations of space for the work-storage arrays. Presented next are the free-field formats which are used when entering the required sets of input data. These include the tabular lists of model parameters, the specification or characterization of the samples, and the collective sets of measurements of the ellipsometric angles. Finally, information is given regarding the command options that are available to the user in exercising operational control over the program, i.e., constraints of procedures during analyses. A selected set of examples involving sample data input and program output is presented in section 5. The listing of the software source files and routines are presented in section 6.

The source code is written predominantly in standard FORTRAN-77. Minor compiler extentions are associated with the DO-loop constructions and formats for OPEN/CLOSE statements; such are appropriate for a VAX computer. The source code contains some in-line comments for the user's convenience. The more expert user may want to modify routines while taking advantage of the interior data structure of the program, whereas the less expert user may simply want to operate the package as presented. The program is available upon request by sending the author a floppy disk or a small nine-track magnetic tape.

2. The Forward Problem

Any discussion regarding the description of the scattering of light from a medium must ultimately incorporate some formulation involving Maxwell's equations [1-7]. Here, a brief discussion is presented regarding the reflection and refraction of a time-harmonic monochromatic plane electromagnetic wave obliquely incident on a flat plane surface of a stratified medium exhibiting depth-dependent optical properties. Within those regions or neighborhoods where the optical properties of the material medium are continuous, isotropic, linear, local, and absent of free charges and surface currents, Maxwell's equations in centimeter-gram-second (cgs) units are

$$\begin{aligned} \nabla \times \mathbf{E} &= -\frac{1}{c}\dot{\mathbf{B}} \\ &= i\frac{\omega}{c}\mu\mathbf{H}, \end{aligned} \tag{1}$$

$$\begin{aligned} \nabla \times \mathbf{H} &= \frac{1}{c}\dot{\mathbf{D}} + \frac{4\pi}{c}\mathbf{J} \\ &= -i\frac{\omega}{c}\left(\epsilon + i\frac{4\pi\sigma}{\omega}\right)\mathbf{E} \\ &= -i\frac{\omega}{c}\varepsilon\mathbf{E}, \end{aligned} \tag{2}$$

$$\nabla \cdot \mathbf{D} = 0, \tag{3}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{4}$$

with the constitutive relationships

$$\mathbf{B} = \mu\mathbf{H}, \qquad \mathbf{D} = \epsilon\mathbf{E}, \qquad \mathbf{J} = \sigma\mathbf{E},$$

and the complex dielectric function ε is defined by

$$\varepsilon = \epsilon + i\frac{4\pi\sigma}{\omega} = (n + ik)^2, \tag{5}$$

where $\mathbf{E}$ is the electric field vector, $\mathbf{H}$ is the magnetic field vector, $\mathbf{D}$ is the electric displacement, $\mathbf{B}$ is the magnetic induction, $\mathbf{J}$ is the induced conduction current, μ is the magnetic permeability, σ is the specific or optical conductivity, ϵ is the dielectric

permittivity, n is the index of refraction, k is the extinction coefficient, ω is the angular frequency of the incident light, c is the vacuum speed of light, and where the dot denotes differentiation with respect to time, and the time dependence is assumed to be of the form $e^{-i\omega t}$, which is distinct from the engineering or Nebraska convention that assumes $e^{+i\omega t}$ and $\varepsilon = (n - ik)^2$.

For convenience, the material medium is assumed to occupy the semi-infinite positive half-space ($z > 0$), with the flat surface being the plane ($z = 0$). For the isotropic homogeneous medium with depth-dependent (z) optical properties, there is translational invariance within the (x, y) plane. Thus, the z-axis is aligned in a direction normal to the surface and planes of stratification. A lossless isotropic homogeneous medium like air or vacuum is assumed to occupy the so-called ambient region ($z < 0$). For this configuration of geometry and restricted class of problems, the optical properties are assumed to be of the form

$$\epsilon = \epsilon(\omega, z), \qquad \sigma = \sigma(\omega, z), \qquad \mu = \mu(\omega, z) = 1, \qquad z > 0,$$

$$\epsilon = \epsilon_o(\omega), \qquad \sigma = 0, \qquad \mu = 1, \qquad z < 0,$$

where allowance is made for dispersive nonmagnetic materials. Then, it is convenient to express eqs (1) and (2) in Cartesian coordinates, i.e.,

$$\nabla_y E_z - \nabla_z E_y = i\left(\frac{\omega}{c}\right)\mu H_x, \tag{6}$$

$$\nabla_z E_x - \nabla_x E_z = i\left(\frac{\omega}{c}\right)\mu H_y, \tag{7}$$

$$\nabla_x E_y - \nabla_y E_x = i\left(\frac{\omega}{c}\right)\mu H_z, \tag{8}$$

$$\nabla_y H_z - \nabla_z H_y = -i\left(\frac{\omega}{c}\right)\varepsilon E_x, \tag{9}$$

$$\nabla_z H_x - \nabla_x H_z = -i\left(\frac{\omega}{c}\right)\varepsilon E_y, \tag{10}$$

$$\nabla_x H_y - \nabla_y H_x = -i\left(\frac{\omega}{c}\right)\varepsilon E_z. \tag{11}$$

A source time-harmonic monochromatic plane wave is assumed to be incident on the surface of the medium ($z = 0$), from the ambient region ($z < 0$), in a direction of angle ϕ with respect to the surface normal. The plane of incidence is taken to be the (z, x)

plane, and the direction cosines of propagation both being positive. Any arbitrarily polarized wave may be resolved into two waves. With this geometry, it is natural to choose the two waves to be the orthogonal set involving linearly polarized waves: one which maintains its electric vector polarized perpendicular to the plane of incidence, i.e., transverse electric, TE, or s–polarization; and the other which maintains its magnetic vector perpendicular to the plane of incidence, i.e., transverse magnetic, TM, or p–polarization. This set uncouples the system of Maxwell's equations, and each polarization (s, p) may be treated individually.

The next problem under consideration in modeling the medium is to determine or otherwise calculate the resulting reflected and/or refracted fields which may be subjected to measurement with an ellipsometer. In the following subsections, further discussion is presented regarding the calculation of the reflected and/or refracted waves for each polarization. Here, the problem for each polarization involves a second-order ordinary differential equation (ODE) complete with boundary conditions.

It is convenient to recognize that both ODEs admit plane wave solutions if the dielectric function within each layer is a constant. The profile of the stratified structure of the dielectric function is approximated by a stepped profile of isotropic, homogeneous, uniformily thick layers or laminae. Then, the problem of solving Maxwell's equations is reduced to one of matching analytic solutions at boundaries between adjacent layers. This involves solving a set of recurrence relations and is the form of the forward problem that is actually solved here. The Jacobian is determined in a like manner. Finally, regarding the quantities that are measured by the ellipsometer and characterize the phase shift induced upon reflection, the ellipsometric angles (Δ, ψ) are defined in terms of the reflected fields.

2.1 Transverse Electric Mode or S–Polarization

2.1.1 Reflection Coefficient

Due to the uncoupling within the system of Maxwell's equations, there is one solution or polarization where the incident, refracted, and reflected fields may be characterized by electric-field vectors which are aligned in a direction perpendicular to the plane of incidence, i.e., along the y–direction: $(E_z = E_x = 0)$ and $(E_y \neq 0)$. This is called the transverse electric mode or s–polarization. From eq (7), $H_y = 0$; from eq (11), $H_x = H_x(z, x)$;

and from eq (9), $H_z = H_z(z,x)$; so there are no dependences on y. Substituting eqs (6) and (8) into eq (10) yields

$$\nabla_z\left(-\frac{c}{i\mu\omega}\nabla_z E_y\right) - \nabla_x\left(\frac{c}{i\mu\omega}\nabla_x E_y\right) = -i\frac{\omega}{c}\varepsilon E_y$$

or

$$\left[\nabla_z^2 + \nabla_x^2 - (\nabla_z \ln\mu)\,\nabla_z + \frac{\omega^2}{c^2}\varepsilon\mu\right] E_y = 0.$$

Assuming a solution by the method of separable variables, let $E_y = \mathcal{Z}(z)\mathcal{X}(x)$ and $k_o = \omega/c$, then

$$\frac{1}{\mathcal{Z}}\left[\nabla_z^2 - (\nabla_z \ln\mu)\,\nabla_z + k_o^2\varepsilon\mu\right]\mathcal{Z} \;=\; -\frac{1}{\mathcal{X}}\nabla_x^2\mathcal{X} \;=\; \kappa_a^2$$

where κ_a is an arbitrary constant and real because of translation invariance within the (x,y) plane. One may write

$$\kappa_a = k_o\alpha,$$

so that

$$\mathcal{X} = e^{ixk_o\alpha} \tag{12}$$

where in the ambient region,

$$\alpha = \sqrt{\varepsilon_a}\sin\phi = \text{constant} \geq 0, \tag{13}$$

with ε_a being the dielectric function of the ambient region, and ϕ being the angle of incidence of the light, i.e., $0 \leq \phi \leq \frac{\pi}{2}$. Here, $(\phi = 0)$ involves a direction normal to the surface.

From the differential representation of Maxwell's equations across a surface of discontinuity in piecewise continuous media, the tangential fields, E_y and H_x, are continuous across boundaries between adjacent layers. Consequently, eq (12) becomes valid within each spatial region, i.e., ambient, layers, and substrate. Then, letting

$$\eta = \sqrt{\varepsilon\mu - \alpha^2}, \qquad \text{where} \qquad 0 \leq \arg(\eta) \leq \pi, \tag{14}$$

the above ODE in z may be written as

$$\left[\nabla_z^2 + (\nabla_z \ln\mu)\,\nabla_z + k_o^2\eta^2\right]\mathcal{Z} = 0.$$

Within any layer of the media, the optical properties are assumed to be homogeneous, so the middle gradient term vanishes. The ODE finally becomes

$$\left[\nabla_z^2 + k_o^2\eta^2\right]\mathcal{Z} = 0, \tag{15}$$

and admits solutions involving the well-known set of damped plane waves,

$$\mathcal{Z} = E^{(+)}e^{izk_o\eta} + E^{(-)}e^{-izk_o\eta}.$$

From eq (6),

$$\nabla_z E_y = -i\frac{\omega}{c}\mu H_x = -ik_o\mu H_x,$$

one finds

$$-H_x = \frac{\eta}{\mu}\left[E^{(+)}e^{izk_o\eta} - E^{(-)}e^{-izk_o\eta}\right]\mathcal{X}.$$

It follows from the continuity of the tangential fields that the logarithmic derivative of the solution or admittance function

$$\mathcal{Y} = -\frac{H_x}{E_y} = \frac{\eta}{\mu}\left[\frac{E^{(+)}e^{izk_o\eta} - E^{(-)}e^{-izk_o\eta}}{E^{(+)}e^{izk_o\eta} + E^{(-)}e^{-izk_o\eta}}\right]$$
$$= \frac{\eta}{\mu}\left[\frac{1 - Re^{-i2zk_o\eta}}{1 + Re^{-i2zk_o\eta}}\right]$$

is a continuous function of z alone, where R is the reflection coefficient local to the layer or spatial region. It also follows that coefficient $E^{(-)}$ vanishes within the substrate region because its basis function diverges for large z. This allows one to evaluate the admittance at the boundary of the substrate,

$$\mathcal{Y}_s = \frac{\eta_s}{\mu_s} = \frac{\sqrt{\varepsilon_s\mu_s - \varepsilon_a \sin^2\phi}}{\mu_s} \tag{16}$$

where the subscript s denotes the substrate region. This will be called the innermost boundary condition for the admittance.

The last part of the problem is the calculation of the reflection coefficient local to the ambient region, R_a, where the subscript denotes the ambient region. Since the optical properties are assumed to be known, the method of solution reduces to that of matching the boundary conditions across each layer that separates the substrate from the ambient region. A reflection coefficient local to each region needs to be determined. This

is accomplished by considering a single layer alone. For convenience, consider the layer (L) adjacent to the substrate, which has complex dielectric function ϵ_L and thickness z_L. Let a local coordinate system be defined on the layer and let it be oriented so that the positive z-axis extends from left-to-right. Let layer (L) occupy the spatial region ($0 \le z \le z_L$), while the substrate is located within the unbounded region ($z > z_L$). Then, from eq (14),

$$\eta_L = \sqrt{\epsilon_L \mu_L - \epsilon_a \sin^2 \phi}$$

and due to continuity, the boundary condition on the right-hand-side (rhs) of the layer (L) is of the form,

$$\begin{aligned} \mathcal{Y}_L(\text{rhs}) \equiv \mathcal{Y}_L(z_L) &= \mathcal{Y}_s \\ &= \frac{\eta_L}{\mu_L} \left[\frac{1 - R_L e^{-i2z_L k_o \eta_L}}{1 + R_L e^{-i2z_L k_o \eta_L}} \right] \end{aligned}$$

so that

$$R_L = e^{i2z_L k_o \eta_L} \left[\frac{(\eta_L/\mu_L) - \mathcal{Y}_L(\text{rhs})}{(\eta_L/\mu_L) + \mathcal{Y}_L(\text{rhs})} \right]$$

and

$$\mathcal{Y}_L(\text{lhs}) \equiv \mathcal{Y}_L(z=0) = \frac{\eta_L}{\mu_L} \left[\frac{1 - R_L}{1 + R_L} \right].$$

The dependences on the model parameters and (rhs) boundary condition may be expressed more explicitly, by letting

$$\gamma_L = e^{i2z_L k_o \eta_L}, \tag{17}$$

and substituting to find

$$\begin{aligned} \mathcal{Y}_L(\text{lhs}) &= \frac{\eta_L}{\mu_L} \left[\frac{(\eta_L/\mu_L) + \mathcal{Y}_L(\text{rhs}) - \gamma_L \left[(\eta_L/\mu_L) - \mathcal{Y}_L(\text{rhs}) \right]}{(\eta_L/\mu_L) + \mathcal{Y}_L(\text{rhs}) + \gamma_L \left[(\eta_L/\mu_L) - \mathcal{Y}_L(\text{rhs}) \right]} \right] \\ &= \frac{\eta_L}{\mu_L} \left[\frac{(1-\gamma_L)(\eta_L/\mu_L) + (1+\gamma_L)\mathcal{Y}_L(\text{rhs})}{(1+\gamma_L)(\eta_L/\mu_L) + (1-\gamma_L)\mathcal{Y}_L(\text{rhs})} \right]. \end{aligned} \tag{18}$$

This equation is applied recursively, matching the boundary conditions and moving toward the left across each and every layer in succession, so that the admittance function

may be evaluated at the outer surface which borders the ambient region, i.e., $\mathcal{Y}(z=0)$. The reflection coefficient in the ambient region, R_a, is found from $\mathcal{Y}(0)$ and is given by

$$R_a = \left[\frac{(\eta_a/\mu_a) - \mathcal{Y}(0)}{(\eta_a/\mu_a) + \mathcal{Y}(0)}\right], \tag{19}$$

where again, all of the μ's can be set equal to unity, if the material is nonmagnetic.

2.1.2 Partial Derivatives

The forward scattering problem assumes that all of the parameters which characterize the sample are known, as well as the source field which is incident upon the surface of the sample. The forward problem involves calculating the reflected and refracted fields resulting from this incident field. But reflection ellipsometry provides measurements involving only two numbers, i.e., the relative amplitude attenuation and the relative phase shift of the light that is induced upon reflection for a given angle of incidence ϕ and wavelength λ. This is discussed later in section 2.3. It is necessary to invert the ellipsometric equations and to determine best estimates to the model parameters that give rise to the reflected fields that manifest the measured phase shifts and amplitude attenuations. This is the inverse problem that is discussed later in section 3. During the inversion process that involves finding improvements to the values of the model parameters, it becomes necessary to estimate the effect that each model parameter induces in the calculated phase shifts or reflected fields. This requires partial derivatives of the fields with respect to the model parameters [8,9]. Collectively, these derivatives will be called the Jacobian.

Since the model parameters include refractive indices and extinction coefficients for each layer and substrate, as well as layer thicknesses, each layer is characterized by (z, n, k) and substrate by (n, k). Here, the subscript labeling of the layer or substrate region and the incident wavelength of light has been suppressed. For example, for a sample containing m_L layers measured by ellipsometry involving m_λ wavelengths, there would generally be $m_L + 2(m_L + 1)m_\lambda$ model parameters. However, only three types of partial derivatives need to be considered because the reflection coefficient involves a composite function of such basic parameters as thicknesses via $(i2zk_o)$, complex dielectric functions $(\varepsilon = (n+ik)^2)$, and the angle of incidence via $(-\varepsilon_a \sin^2\phi)$. Recalling eq (19) and suppressing references to the μ's, it follows that one may readily express the partial derivatives

of the reflection coefficient associated with these three basic parameters. They are of the following general operator form:

$$\frac{\partial R_a}{\partial\left(i2zk_o\right)} \equiv R_a' = -\frac{2\eta_a \mathcal{Y}'(0)}{\left[\eta_a + \mathcal{Y}(0)\right]^2} \tag{20}$$

when the prime is used for convenience as an operator to imply a partial derivative with respect to $(i2zk_o)$,

$$\frac{\partial R_a}{\partial \varepsilon} \equiv R_a' = -\frac{2\eta_a \mathcal{Y}'(0)}{\left[\eta_a + \mathcal{Y}(0)\right]^2} \tag{21}$$

when the prime is used to imply a partial derivative with respect to ε, and

$$\frac{\partial R_a}{\partial\left(-\varepsilon_a \sin^2\phi\right)} \equiv R_a' = \frac{2\left[\eta_a' \mathcal{Y}(0) - \eta_a \mathcal{Y}'(0)\right]}{\left[\eta_a + \mathcal{Y}(0)\right]^2} \tag{22}$$

when the prime is used to imply a partial derivative with respect to $(-\varepsilon_a \sin^2\phi)$. These need to be calculated for each model parameter undergoing variation.

The derivatives of the admittance function evaluated at the surface may be understood and calculated in a straightforward manner, by considering the effect of a variation in the model parameters within the substrate region. From eq (16),

$$\mathcal{Y}_s = \eta_s = \sqrt{\varepsilon_s - \varepsilon_a \sin^2\phi},$$

so

$$\mathcal{Y}_s' = \frac{\partial \mathcal{Y}_s}{\partial \varepsilon_s} = \frac{\partial \mathcal{Y}_s}{\partial\left(-\varepsilon_a \sin^2\phi\right)} = \frac{1}{2\eta_s} = \frac{1}{2\mathcal{Y}_s}. \tag{23}$$

The effect of this derivative on the admittance must then be matched as a boundary condition across each layer that lies between the substrate and the ambient region. This gives rise to four cases to consider during the matching process, depending upon whether the partial derivative operates on a function whose parameter is exterior or interior to the local region of interest. These four cases follow directly from considering the boundary conditions satisfied by any given layer, i.e., eq (18) for the layer (L) adjacent to the substrate,

$$\mathcal{Y}_L(\text{lhs}) = \eta_L \left[\frac{\left(1-\gamma_L\right)\eta_L + \left(1+\gamma_L\right)\mathcal{Y}_L(\text{rhs})}{\left(1+\gamma_L\right)\eta_L + \left(1-\gamma_L\right)\mathcal{Y}_L(\text{rhs})}\right].$$

Case 1 considers any partial derivative taken with respect to (ε, z) that operates exterior to the local region. Here, only the (rhs) boundary condition is affected. Using the chain rule for partial derivatives on the above equation, it follows that

$$\mathcal{Y}'_L(\text{lhs}) = \frac{4\gamma_L \eta_L^2 \mathcal{Y}'_L(\text{rhs})}{\left[(1+\gamma_L)\eta_L + (1-\gamma_L)\mathcal{Y}_L(\text{rhs})\right]^2}. \tag{24}$$

This equation provides the key relationship for transporting derivatives involving matched boundary conditions across any given layer. In general, if the layers of a sample are indexed consecutively, where the top layer (1) is adjacent to the ambient region, and where layer (L) is not the top layer, then

$$\mathcal{Y}_{L-1}(\text{rhs}) = \mathcal{Y}_L(\text{lhs}) \qquad \text{implies} \qquad \mathcal{Y}'_{L-1}(\text{rhs}) = \mathcal{Y}'_L(\text{lhs}). \tag{25}$$

So, eqs (24) and (25) are applied repeatedly until the outer boundary at $(z=0)$ is reached; i.e., determining $\mathcal{Y}'(0)$, then eqs (20) or (21) may be evaluated as appropriate.

Case 2 considers variations in the complex dielectric function (ε_L) interior to the layer (L). This leaves $\mathcal{Y}_L(\text{rhs})$ unaffected. Again, using the chain rule for the partial derivatives, and taking advantage of the identities,

$$\eta'_L = \frac{\partial \eta_L}{\partial \varepsilon_L} = \frac{1}{2\eta_L}$$

and

$$\gamma'_L = \frac{\partial \gamma_L}{\partial \varepsilon_L} = \frac{\partial \gamma_L}{\partial \eta_L}\,\frac{\partial \eta_L}{\partial \varepsilon_L} = \left(\frac{i z_L k_o}{\eta_L}\right)\gamma_L,$$

one finds that the variation may be written as

$$\mathcal{Y}'_L(\text{lhs}) = \frac{C_1^{(2,\text{TE})}}{\left[(1+\gamma_L)\eta_L + (1-\gamma_L)\mathcal{Y}_L(\text{rhs})\right]} - \frac{\left[(1-\gamma_L)\eta_L + (1+\gamma_L)\mathcal{Y}_L(\text{rhs})\right]}{\left[(1+\gamma_L)\eta_L + (1-\gamma_L)\mathcal{Y}_L(\text{rhs})\right]^2}\, C_2^{(2,\text{TE})} \tag{26}$$

where

$$C_1^{(2,\text{TE})} = (1-\gamma_L) + \frac{(1+\gamma_L)}{2}\frac{\mathcal{Y}_L(\text{rhs})}{\eta_L} + i z_L k_o \gamma_L \left[\mathcal{Y}_L(\text{rhs}) - \eta_L\right]$$

and

$$C_2^{(2,\text{TE})} = \frac{(1+\gamma_L)}{2} + i z_L k_o \gamma_L \left[\eta_L - \mathcal{Y}_L(\text{rhs})\right].$$

This boundary condition is then matched across every layer positioned between layer (L) and the ambient, by applying case 1 and eq (24) accordingly.

Case 3 considers variations in the layer's thickness via $(i2z_L k_o)$ interior to the local region. Taking the necessary partial derivatives and utilizing the fact that

$$\gamma'_L = \frac{\partial \gamma_L}{\partial (i2z_L k_o)} = \eta_L \gamma_L ,$$

the variation may be expressed as

$$\mathcal{Y}'_L(\text{lhs}) = -\frac{2\gamma_L \eta_L^2 \ [\eta_L - \mathcal{Y}_L(\text{rhs})] \ [\eta_L + \mathcal{Y}_L(\text{rhs})]}{[(1+\gamma_L)\eta_L + (1-\gamma_L)\mathcal{Y}_L(\text{rhs})]^2}. \tag{27}$$

Again, this boundary condition is transported to the outer surface, by applying case 1 and eq (24) as necessary.

Case 4 considers variations in the angle of incidence via $(-\varepsilon_a \sin^2\phi)$ exterior and interior to the local region under consideration. Recalling that

$$\eta'_L = \frac{\partial \eta_L}{\partial \left(-\varepsilon_a \sin^2\phi\right)} = \frac{1}{2\eta_L}$$

and

$$\gamma'_L = \frac{\partial \gamma_L}{\partial \eta_L}\, \eta'_L = \left(\frac{iz_L k_o}{\eta_L}\right) \gamma_L ,$$

the variation may be expressed as

$$\begin{aligned} \mathcal{Y}'_L(\text{lhs}) = {} & \frac{C_1^{(4,\text{TE})}}{[(1+\gamma_L)\eta_L + (1-\gamma_L)\mathcal{Y}_L(\text{rhs})]} \\ & - \frac{[(1-\gamma_L)\eta_L + (1+\gamma_L)\mathcal{Y}_L(\text{rhs})]}{[(1+\gamma_L)\eta_L + (1-\gamma_L)\mathcal{Y}_L(\text{rhs})]^2}\, C_2^{(4,\text{TE})} \end{aligned} \tag{28}$$

where

$$\begin{aligned} C_1^{(4,\text{TE})} = {} & iz_L k_o \gamma_L \left[\mathcal{Y}_L(\text{rhs}) - \eta_L\right] + (1-\gamma_L) + \frac{(1+\gamma_L)}{2}\frac{\mathcal{Y}_L(\text{rhs})}{\eta_L} \\ & + (1+\gamma_L)\eta_L \mathcal{Y}'_L(\text{rhs}) \end{aligned}$$

and

$$C_2^{(4,\text{TE})} = iz_L k_o \gamma_L \left[\eta_L - \mathcal{Y}_L(\text{rhs})\right] + \frac{(1+\gamma_L)}{2} + (1-\gamma_L)\eta_L \mathcal{Y}'_L(\text{rhs}).$$

This case is distinct from the others by requiring both $\mathcal{Y}_L(\text{rhs})$ and $\mathcal{Y}'_L(\text{rhs})$ to be in the right-hand-side of the equation as may be seen from the coefficients $C^{(4)}$. Consequently, eq (28) must be used recursively, starting with the boundary conditions for the substrate region, i.e., eq (23).

Following the evaluations of the necessary partial derivatives in eqs (20) to (22), $\mathcal{Y}'(0)$ and R'_a, it is necessary to express the derivatives in terms of the angle of incidence ϕ and the model parameters (z, n, k) for the appropriate layer or substrate, i.e.,

$$\frac{\partial R_a}{\partial z} = (i2k_o)\,\frac{\partial R_a}{\partial (i2zk_o)}, \tag{29}$$

$$\frac{\partial R_a}{\partial n} = 2\,(n+ik)\,\frac{\partial R_a}{\partial \varepsilon}, \tag{30}$$

$$\frac{\partial R_a}{\partial k} = 2i\,(n+ik)\,\frac{\partial R_a}{\partial \varepsilon}, \tag{31}$$

and

$$\frac{\partial R_a}{\partial \phi} = -2\varepsilon_a \sin\phi\cos\phi\;\frac{\partial R_a}{\partial\left(-\varepsilon_a \sin^2\phi\right)}. \tag{32}$$

These expressions determine the Jacobian of the reflected fields. These are then used in section 2.3, where the measured ellipsometric angles (Δ, ψ) and the associated partial derivatives are defined in terms of the reflected fields.

2.2 Transverse Magnetic Mode or *P*–Polarization

2.2.1 Reflection Coefficient

Proceeding in like manner of presentation of the TE mode, the uncoupling of Maxwell's equations provides a solution or polarization where the incident, refracted, and reflected fields may be characterized by electric field vectors which are aligned in a direction parallel to the plane of incidence, i.e., the (z, x) plane. Here, $(H_z = H_x = 0)$, and $(H_y \neq 0)$. This is called the transverse magnetic mode or p–polarization. From eq (10), $E_y = 0$; from eq (6), $E_z = E_z(z, x)$; from eq (8), $E_x = E_x(z, x)$; so there are no dependences on y. Substituting eqs (9) and (11) into eq (7) yields

$$\nabla_x\left(\frac{1}{ik_o\varepsilon}\nabla_x H_y\right) - \nabla_z\left(\frac{-1}{ik_o\varepsilon}\nabla_z H_y\right) = i\mu k_o H_y$$

or

$$\left[\nabla_z^2 + \nabla_z^2 - (\nabla_z \ln \varepsilon)\,\nabla_z + k_o^2 \varepsilon \mu\right] H_y = 0.$$

Assuming a solution by the method of separable variables, let $H_y = \mathcal{Z}(z)\, e^{izk_o\alpha}$. Then, restricting the domain to a single layer or substrate region where the optical properties are homogeneous and uniform, the gradient term vanishes, so that the ODE is of the form of eq (15), which has solutions involving a set of damped plane waves, i.e.,

$$\mathcal{Z} = H^{(+)} e^{izk_o\eta} + H^{(-)} e^{-izk_o\eta}.$$

Again, the tangential component of the magnetic field vector is continuous across boundaries between adjacent layers involving no interfacial currents. Accordingly, the method of solution follows from eq (9),

$$\nabla_z H_y = ik_o \varepsilon E_x$$

so

$$E_x = \frac{\eta}{\varepsilon}\left[H^{(+)} e^{izk_o\eta} - H^{(-)} e^{-izk_o\eta}\right] \mathcal{X}.$$

Since the tangential components of the field vectors are continuous across boundaries, the logarithmic derivative of the solution or impedance function

$$\mathcal{Y} = \frac{E_x}{H_y} = \frac{\eta}{\varepsilon}\left[\frac{H^{(+)} e^{izk_o\eta} - H^{(-)} e^{-izk_o\eta}}{H^{(+)} e^{izk_o\eta} - H^{(-)} e^{-izk_o\eta}}\right]$$
$$= \frac{\eta}{\varepsilon}\left[\frac{1 - Re^{-i2zk_o\eta}}{1 + Re^{-i2zk_o\eta}}\right]$$

is a continuous function of z alone. The innermost boundary condition for the impedance function is

$$\mathcal{Y}_s = \frac{\eta_s}{\varepsilon_s} = \frac{\sqrt{\varepsilon_s \mu_s - \varepsilon_a \sin^2\phi}}{\varepsilon_s} \tag{33}$$

where the subscript s denotes the substrate region. Starting with the substrate region, the boundary conditions of the impedance function may be matched to the adjacent layer (L),

$$\mathcal{Y}_L(\text{rhs}) = \mathcal{Y}_L(z_L) = \mathcal{Y}_s$$
$$= \frac{\eta_L}{\varepsilon_L}\left[\frac{1 - R_L e^{-i2z_L k_o \eta_L}}{1 + R_L e^{-i2z_L k_o \eta_L}}\right]$$

so that

$$R_L = e^{i2z_L k_o \eta_L}\left[\frac{(\eta_L/\varepsilon_L) - \mathcal{Y}_L(\text{rhs})}{(\eta_L/\varepsilon_L) + \mathcal{Y}_L(\text{rhs})}\right]$$

and

$$\mathcal{Y}_L(\text{lhs}) = \mathcal{Y}_L(z=0) = \frac{\eta_L}{\varepsilon_L}\left[\frac{1-R_L}{1+R_L}\right],$$

or upon substitution with eq (17),

$$\begin{aligned}\mathcal{Y}_L(\text{lhs}) &= \frac{\eta_L}{\varepsilon_L}\left[\frac{(\eta_L/\varepsilon_L)+\mathcal{Y}_L(\text{rhs})-\gamma_L\left[(\eta_L/\varepsilon_L)-\mathcal{Y}_L(\text{rhs})\right]}{(\eta_L/\varepsilon_L)+\mathcal{Y}_L(\text{rhs})+\gamma_L\left[(\eta_L/\varepsilon_L)-\mathcal{Y}_L(\text{rhs})\right]}\right] \\ &= \frac{\eta_L}{\varepsilon_L}\left[\frac{(1-\gamma_L)(\eta_L/\varepsilon_L)+(1+\gamma_L)\mathcal{Y}_L(\text{rhs})}{(1+\gamma_L)(\eta_L/\varepsilon_L)+(1-\gamma_L)\mathcal{Y}_L(\text{rhs})}\right] \\ &= \frac{\eta_L}{\varepsilon_L}\left[\frac{(1-\gamma_L)\eta_L+(1+\gamma_L)\varepsilon_L\mathcal{Y}_L(\text{rhs})}{(1+\gamma_L)\eta_L+(1-\gamma_L)\varepsilon_L\mathcal{Y}_L(\text{rhs})}\right]. \end{aligned} \tag{34}$$

The reflection coefficient in the ambient region, R_a, is found from the impedance function evaluated at the outer surface ($z=0$) and is given by

$$R_a = \left[\frac{(\eta_a/\varepsilon_a)-\mathcal{Y}(0)}{(\eta_a/\varepsilon_a)+\mathcal{Y}(0)}\right]. \tag{35}$$

2.2.2 Partial Derivatives

The discussion presented here parallels section 2.1.2 of the TE mode. Recalling eq (35), it follows that the operator equations regarding partial derivatives are of the form:

$$\frac{\partial R_a}{\partial(i2zk_o)} \equiv R_a' = -\frac{2(\eta_a/\varepsilon_a)\mathcal{Y}'(0)}{\left[(\eta_a/\varepsilon_a)+\mathcal{Y}(0)\right]^2} \tag{36}$$

when the prime is used to imply a partial derivative with respect to $(2izk_o)$,

$$\frac{\partial R_a}{\partial\varepsilon} \equiv R_a' = -\frac{2(\eta_a/\varepsilon_a)\mathcal{Y}'(0)}{\left[(\eta_a/\varepsilon_a)+\mathcal{Y}(0)\right]^2} \tag{37}$$

when the prime is used to imply a partial derivative with respect to ε, and

$$\frac{\partial R_a}{\partial\left(-\varepsilon_a\sin^2\phi\right)} \equiv R_a' = \frac{2\left[(\eta_a/\varepsilon_a)'\mathcal{Y}(0)-(\eta_a/\varepsilon_a)\mathcal{Y}'(0)\right]}{\left[(\eta_a/\varepsilon_a)+\mathcal{Y}(0)\right]^2} \tag{38}$$

when the prime is used to imply a partial derivative with respect to $(-\varepsilon_a\sin^2\phi)$. These need to be calculated for each model parameter undergoing variation.

These derivatives of the impedance function may be understood by considering a variation in the model parameter within the substrate region. From eq (33), one finds that

$$\frac{\partial \mathcal{Y}_s}{\partial \varepsilon_s} = \mathcal{Y}_s' = \left(\frac{\eta_s}{\varepsilon_s}\right)' = \frac{1}{2\eta_s \varepsilon_s} - \frac{\eta_s}{\varepsilon_s^2} = \frac{\varepsilon_s - 2\eta_s^2}{2\eta_s \varepsilon_s^2} \tag{39}$$

and

$$\frac{\partial \mathcal{Y}_s}{\partial \left(-\varepsilon_a \sin^2 \phi\right)} = \mathcal{Y}_s' = \frac{1}{2\eta_s \varepsilon_s}. \tag{40}$$

The effect of each derivative may be viewed as a variation on the impedance function that must be matched as a boundary condition across each layer that lies between the substrate and ambient region. Again, four cases need to be considered, depending upon whether the partial derivative operates on a function whose parameter is exterior or interior to the local region of interest. Consider the adjacent layer (L).

Case 1 considers partial derivatives with respect to (ε, z) operating exterior to the local region. Again, only the (rhs) boundary condition is affected. Operating upon eq (34) yields

$$\mathcal{Y}_L'(\text{lhs}) = \frac{4\gamma_L \eta_L^2 \mathcal{Y}_L'(\text{rhs})}{\left[(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L \mathcal{Y}_L(\text{rhs})\right]^2}. \tag{41}$$

The above equation is applied repeatedly until the outer boundary at $(z = 0)$ is reached; i.e., $\mathcal{Y}'(0)$, so that eq (36) or (37) may be evaluated where appropriate.

Case 2 considers variations in the complex dielectric function (ε_L) interior to layer (L). This leaves $\mathcal{Y}_L(\text{rhs})$ unaffected. The partial derivative may be expressed rather simply in unreduced form,

$$\begin{aligned}\mathcal{Y}_L'(\text{lhs}) = {} & \left(\frac{\varepsilon_L - 2\eta_L^2}{2\eta_L \varepsilon_L^2}\right)\left[\frac{(1-\gamma_L)\eta_L + (1+\gamma_L)\varepsilon_L \mathcal{Y}_L(\text{rhs})}{(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L \mathcal{Y}_L(\text{rhs})}\right] \\ & + \left(\frac{\eta_L}{\varepsilon_L}\right)\frac{C_1^{(2,\text{TM})}}{\left[(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L \mathcal{Y}_L(\text{rhs})\right]} \\ & - \left(\frac{\eta_L}{\varepsilon_L}\right)\frac{\left[(1-\gamma_L)\eta_L + (1+\gamma_L)\varepsilon_L \mathcal{Y}_L(\text{rhs})\right]}{\left[(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L \mathcal{Y}_L(\text{rhs})\right]^2} C_2^{(2,\text{TM})}\end{aligned} \tag{42}$$

where

$$C_1^{(2,\text{TM})} = \left(\frac{1-\gamma_L}{2\eta_L}\right) + (1+\gamma_L)\mathcal{Y}_L(\text{rhs}) + iz_L k_o \gamma_L \left[\left(\frac{\eta_L}{\varepsilon_L}\right)\mathcal{Y}_L(\text{rhs}) - 1\right]$$

and

$$C_2^{(2,\mathrm{TM})} = \left(\frac{1+\gamma_L}{2\eta_L}\right) + (1-\gamma_L)\,\mathcal{Y}_L(\mathrm{rhs}) + iz_L k_o \gamma_L \left[1 - \left(\frac{\eta_L}{\varepsilon_L}\right)\mathcal{Y}_L(\mathrm{rhs})\right].$$

This boundary condition may then be matched across any layer positioned between layer (L) and the ambient, by repeated use of case 1 and eq (41) as appropriate.

Case 3 considers variations in the layer's thickness via $(i2z_L k_o)$ interior to the local region. The partial derivative is of the form

$$\mathcal{Y}'_L(\mathrm{lhs}) = -\left(\frac{\eta_L}{\varepsilon_L}\right) \frac{2\gamma_L\eta_L\ [\eta_L - \varepsilon_L\mathcal{Y}_L(\mathrm{rhs})]\ [\eta_L + \varepsilon_L\mathcal{Y}_L(\mathrm{rhs})]}{[(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L\mathcal{Y}_L(\mathrm{rhs})]^2}. \tag{43}$$

Again, this boundary condition is transported to the outer surface, by applying case 1 and eq (41) as necessary.

Case 4 considers variations in the angle of incidence via $(-\varepsilon_a \sin^2\phi)$ exterior and interior to the local region under consideration. The partial derivative may be expressed as

$$\begin{aligned}\mathcal{Y}'_L(\mathrm{lhs}) = {} & \left(\frac{1}{2\eta_L\varepsilon_L}\right)\left[\frac{(1-\gamma_L)\eta_L + (1+\gamma_L)\varepsilon_L\mathcal{Y}_L(\mathrm{rhs})}{(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L\mathcal{Y}_L(\mathrm{rhs})}\right] \\ & + \left(\frac{\eta_L}{\varepsilon_L}\right)\frac{C_1^{(4,\mathrm{TM})}}{[(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L\mathcal{Y}_L(\mathrm{rhs})]} \\ & - \left(\frac{\eta_L}{\varepsilon_L}\right)\frac{[(1-\gamma_L)\eta_L + (1+\gamma_L)\varepsilon_L\mathcal{Y}_L(\mathrm{rhs})]}{[(1+\gamma_L)\eta_L + (1-\gamma_L)\varepsilon_L\mathcal{Y}_L(\mathrm{rhs})]^2}\ C_2^{(4,\mathrm{TM})}\end{aligned} \tag{44}$$

where

$$C_1^{(4,\mathrm{TM})} = \left(\frac{1-\gamma_L}{2\eta_L}\right) + (1+\gamma_L)\,\varepsilon_L\mathcal{Y}'_L(\mathrm{rhs}) + iz_L k_o\gamma_L\left[\left(\frac{\eta_L}{\varepsilon_L}\right)\mathcal{Y}_L(\mathrm{rhs}) - 1\right]$$

and

$$C_2^{(4,\mathrm{TM})} = \left(\frac{1+\gamma_L}{2\eta_L}\right) + (1-\gamma_L)\,\varepsilon_L\mathcal{Y}'_L(\mathrm{rhs}) + iz_L k_o\gamma_L\left[1 - \left(\frac{\eta_L}{\varepsilon_L}\right)\mathcal{Y}_L(\mathrm{rhs})\right].$$

Again, case 4 is distinct from the other three by requiring both, $\mathcal{Y}_L(\mathrm{rhs})$ and $\mathcal{Y}'_L(\mathrm{rhs})$, as may be seen from the coefficients $C^{(4,\mathrm{TM})}$. Consequently, eq (44) must be used recursively, starting with the boundary condition of the substrate region, i.e., eq (40).

The partial derivatives with respect to the model parameters (z, n, k), for each appropriate layer, substrate, and wavelength are required later, and are of the same form as that presented in eqs (29) to (32).

2.3 Ellipsometric Angles (Δ, ψ)

In sections 2.1 and 2.2, a physical model was proposed for a restricted class of media, and the forward electromagnetic problem was solved where the reflection coefficients local to the ambient region are complex amplitudes, i.e., from eq (19)

$$R_{a,s} = |R_{a,s}| \exp(i\delta_s), \tag{45}$$

and from eq (35),

$$R_{a,p} = |R_{a,p}| \exp(i\delta_p), \tag{46}$$

where subscripts (s,p) refer to polarization, and δ refers to the phase. The ellipsometric angles (Δ, ψ) are defined in terms of the complex variable ρ which is formed by the ratio of the reflection coefficients,

$$\rho = \frac{R_{a,p}}{R_{a,s}} = \left|\frac{R_{a,p}}{R_{a,s}}\right| \exp\{i(\delta_p - \delta_s)\} = \tan\psi \ \exp(i\Delta), \tag{47}$$

where the explicit relationship of the phases involves

$$\Delta = \delta_p - \delta_s, \qquad \text{such that} \qquad 0 \leq \Delta < 2\pi, \tag{48}$$

while the modulus involves

$$\tan\psi = \left|\frac{R_{a,p}}{R_{a,s}}\right| = |\rho|, \qquad \text{where} \qquad 0 \leq \psi \leq \frac{\pi}{2},$$

so

$$\psi = \tan^{-1}|\rho|. \tag{49}$$

Further, since the ellipsometric angles refer exclusively to the reflected fields in the ambient region, it is convenient to omit further subscriptive references to the ambient.

It is important to draw attention to two subtleties arising from the choice of convention used in defining the ellipsometric angles. First, the R_s was defined in terms of the electric field, whereas R_p used the magnetic field. If electric fields are used in the definitions exclusively, then $R_{p,E} = -R_{p,H}$, so that only the Δ is affected, i.e.,

$$\Delta_{\text{E}} = \Delta_{\text{H}} + \pi \pmod{2\pi}. \tag{50}$$

Second, the time dependence here was chosen from the *physics* convention, as opposed to the *engineering* or Nebraska convention. The complex amplitudes found with one convention are related to that with the other convention by simple complex conjugation. This affects only the phase or sign of Δ, i.e.,

$$\Delta_{\text{engineering}} = 2\pi - \Delta_{\text{physics}} \pmod{2\pi}. \tag{51}$$

Now that the ellipsometric angles have been defined, it will be necessary to formulate the Jacobian or partial derivatives of the ellipsometric angles in terms of the reflected fields and their partial derivatives. These are important to the general formulation of the inverse problem, which involves inverting the ellipsometric equations discussed later in section 3.

For convenience, let all further primes used within this section refer to an implied partial derivative with respect to some model parameter, e.g., $(z_L, n_L, k_L, n_s, k_s, \phi)$, where the subscripts refer to some layer or substrate region. Since the ellipsometric angles involve phase and modulus information of the complex variable ρ, it will be necessary to develop relationships which express partial derivatives of the modulus and phase of a complex variable in terms of the partial derivatives of the real and imaginary parts of that same complex variable.

This may be accomplished in the usual manner by considering the notation involving a single complex variable, denoted by R, which has modulus r, phase δ, and real and imaginary parts, x and y, respectively, i.e.,

$$R = re^{i\delta} = x + iy \qquad \text{where} \qquad r = |R| = \sqrt{x^2 + y^2}.$$

Then,

$$\frac{R}{R^*} = e^{i2\delta} \qquad \text{and} \qquad i2\delta = \ln\left(R/R^*\right),$$

so that by forming

$$(R^*R)\left[\frac{R'}{R} - \frac{R^{*\prime}}{R^*}\right] = R^*R' - RR^{*\prime}$$

$$r^2\left[\ln\left(\frac{R}{R^*}\right)\right]' = (x - iy)(x' + iy') - (x + iy)(x' - iy')$$

$$2ir^2\delta' = 2i\left(xy' - yx'\right),$$

one finds that

$$\delta' = \frac{xy' - yx'}{x^2 + y^2}. \tag{52}$$

Proceeding in like manner,

$$\left(r^2\right)' = \left(x^2 + y^2\right)' = \left(R^* R\right)'$$

$$2rr' = 2\left(xx' + yy'\right),$$

one finds that

$$r' = \frac{xx' + yy'}{\sqrt{x^2 + y^2}}. \tag{53}$$

Here, eqs (52) and (53) applies to each polarization individually; i.e., one finds $(\delta_s', \delta_p', r_s', r_p')$.

Now, consider the definitions of the ellipsometric angles. From eq (49) and using the trigonometric identity involving derivatives of arctangents, it follows that

$$\psi' = \frac{|\rho|'}{1 + |\rho|^2} = \frac{\left(r_p / r_s\right)'}{\left[1 + \left(r_p / r_s\right)^2\right]}$$

$$= \frac{r_s^2}{r_s^2 + r_p^2}\left[\frac{r_p'}{r_s} - \frac{r_p r_s'}{r_s^2}\right]$$

$$= \frac{r_s r_p' - r_p r_s'}{r_s^2 + r_p^2}, \tag{54}$$

where the right-hand-side of this expression is used in conjunction with eq (53) for each polarization.

Proceeding in like manner with eq (48), it follows that

$$\Delta' = \delta_p' - \delta_s', \tag{55}$$

where the right-hand-side of this expression is used in conjunction with eq (52) for each polarization.

Eqs (52) to (55) provide the basic set of equations necessary for evaluating the various partial derivatives of the ellipsometric angles in terms of the reflection coefficients and their partial derivatives.

3. The Inverse Problem

3.1 Formulation of the Least-Squares Problem

In order to characterize the layered structure of the sample, it is necessary to invert the standard ellipsometric equations, i.e., eqs (48) and (49). These equations describe how the material optical properties of the sample induce a phase-shift in the reflected light that is measured by the ellipsometer. Because of the nonlinearity of the equations, it is usually not possible to find simple analytic expressions which will invert the equations. One common approach to performing such inversions is to formulate them as nonlinear least-squares problems [10–15]. Here, one considers a sequence of forward problems, where each increment of the sequence involves three distinct steps. The steps include: starting with a good estimate of values of the model parameters, determining the deviations between the experiment and the model, and then updating the model parameters with *better* values. This sequence is repeated until the magnitude of the corrections become sufficiently *small*.

The ellipsometric equations are of the form:

$$\Delta = \Delta(\lambda, \phi, \mathbf{b}) \tag{56}$$

$$\psi = \psi(\lambda, \phi, \mathbf{b}) \tag{57}$$

where λ is the vacuum wavelength of the incident light; ϕ is the angle of incidence; $\mathbf{b}$ is an array where the components specify the model parameters, i.e., layer thicknesses, indices of refraction and extinction coefficients of the layers and substrates, e.g., $(z_L, \hat{n_L^\lambda}, \hat{k_L^\lambda}, \hat{n_s^\lambda}, \hat{k_s^\lambda})$, where the subscripts refer to a layer or substrate region; and where the superscripts refer to the wavelength dependence of the optical properties. The standard procedure involves minimizing some non-negative scalar *error* expression containing the deviations between experiment and model of Δ and ψ in the least-squares sense, e.g.,

$$G(\mathbf{b}) = \frac{1}{2M} \sum_{i=1}^{M} \left[\left(\Delta_i^{\mathrm{e}} - \Delta_i^{\mathrm{m}}\right)^2 + \left(\psi_i^{\mathrm{e}} - \psi_i^{\mathrm{m}}\right)^2 \right] \tag{58}$$

$$= \sum_{i=1}^{2M} g_i^2 \; = \mathbf{g}^{\mathrm{T}}\mathbf{g} = |\mathbf{g}|^2 \tag{59}$$

where superscripts (e,m) refer to experiment and model, respectively, M denotes the number of measurements of (Δ, ψ) from experiment, $\mathbf{g}$ is a column array of the deviations, i.e.,

$$g_{2i-1} = \left(\Delta_i^{\mathrm{e}} - \Delta_i^{\mathrm{m}}\right) / \sqrt{2M}, \tag{60}$$

$$g_{2i} = \left(\psi_i^{\mathrm{e}} - \psi_i^{\mathrm{m}}\right) / \sqrt{2M}, \tag{61}$$

where $1 \leq i \leq M$, and the superscript T denotes transposition. Here, the *error* expression assumes equal weighting factors or uncertainties in the measurements of Δ and ψ. This is a simplification because it is well known that the measurement uncertainty in Δ and ψ may differ by more than a factor of two [26].

The third step of the sequence is concerned with the procedure for obtaining *better* numerical values for the model parameters, i.e., the Newton step. An estimate of the necessary correction may be realized by linearizing the functional representation of the model and solving the resulting matrix equation, $\mathbf{g} = \mathbf{J}\mathbf{v}$, where $\mathbf{v}$ is a column array (Newton step) for improving the model parameters that were selected to undergo variation, i.e., $v_j \sim \delta b_j$, and $\mathbf{J}$ is the sparse matrix involving the Jacobian which is not necessarily square, i.e., $J_{ij} \sim \left(1/\sqrt{2M}\right)\left(\partial \Delta_i / \partial b_j\right)$.

Such matrix equations are common to optimization problems and involve only linear algebra. It is well known that additional numerical stability may result when requesting the norm of $\mathbf{v}$ to be minimized as well. This may be accomplished by modifying the *error* expression to

$$G = (\mathbf{g} - \mathbf{J}\mathbf{v})^{\mathrm{T}}(\mathbf{g} - \mathbf{J}\mathbf{v}) + \kappa \mathbf{v}^{\mathrm{T}}\mathbf{v}, \tag{62}$$

where κ is a positive scalar parameter subjectively chosen between 0.01 and 1.0 for our calculations. Of course, the final solution $\mathbf{v}$ should be independent of κ, but here, κ simply moderates the rate of convergence.

It is also known that the columnar scaling of $\mathbf{J}$ affects the accuracy of the solution, as well as the effectiveness of κ. A simple choice for the scaling can be found by considering the diagonal elements from $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ and then defining the diagonal matrix, $\mathbf{S}$, where

$$S_{jj} = \sqrt{\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)_{jj}}\,. \tag{63}$$

Then letting

$$\tilde{\mathbf{J}} = \mathbf{J}\mathbf{S}^{-1}, \tag{64}$$

$$\tilde{\mathbf{v}} = \mathbf{S}\mathbf{v}, \tag{65}$$

$$\mathbf{r} = \mathbf{g} - \mathbf{J}\mathbf{v} = \mathbf{g} - \tilde{\mathbf{J}}\tilde{\mathbf{v}}, \tag{66}$$

a suitable *error* expression may be defined by

$$G = \left(\mathbf{g} - \tilde{\mathbf{J}}\tilde{\mathbf{v}}\right)^{\mathrm{T}}\left(\mathbf{g} - \tilde{\mathbf{J}}\tilde{\mathbf{v}}\right) + \kappa\tilde{\mathbf{v}}^{\mathrm{T}}\tilde{\mathbf{v}} \tag{67}$$

$$= \mathbf{r}^{\mathrm{T}}\mathbf{r} + \kappa\tilde{\mathbf{v}}^{\mathrm{T}}\tilde{\mathbf{v}}. \tag{68}$$

The criterion for critical points or relative minima is a vanishing variation, i.e., $\partial G/\partial\tilde{v}_j = 0$, which yields a set of equations that may be expressed as

$$\begin{bmatrix} \mathbf{1} & \tilde{\mathbf{J}} \\ \tilde{\mathbf{J}}^{\mathrm{T}} & -\kappa\mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \tilde{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{0} \end{bmatrix}, \tag{69}$$

and must then be solved for $\tilde{\mathbf{v}}$.

Because of the sparsity of the Jacobian, it is expedient to utilize an iterative method for solving the above matrix algebra problem. Algorithms [16,17] exist that specifically address this type of problem, one [18,19] of which utilizes a relatively stable Lanczos process (Krylov space decomposition) in formulating the method of steepest-descents. Essentially, the method requires each updating vector be orthogonal to the previous update vectors. Further details may be found elsewhere [18–22]. From this procedure, one finds $\tilde{\mathbf{v}}$, which leads to $\mathbf{v}$, which is then used to improve the estimate of the values for the model parameters, e.g., $b_j^{\mathrm{new}} = b_j^{\mathrm{old}} + v_j$.

Using this improved $\mathbf{b}$, the sequence is repeated again until either: $|\mathbf{g}|$ becomes sufficiently *small*, of the order of a few millidegrees, e.g., the resolution of the measurement, or until $|\tilde{\mathbf{v}}|$ becomes sufficiently *small* so that $|\mathbf{g}|$ suffers no further reduction regardless of magnitude. It is especially during this last case, that it becomes necessary to scan a grid of model parameters. Multiple pseudo-minima may be encountered, e.g., nonuniqueness. Often, this reveals either: (i) correlation which prevents model parameters from being resolved independently; i.e., the measurement data were not sufficiently functionally independent which thereby induces a functional dependence among the model parameters; or (ii) the inadequacy of the model in providing a sufficiently good physical

description of the process, which is likely whenever |g| greatly exceeds the resolution of the measurements.

Finally, it is important to realize that in the above outlined steps, the emphasis centers on searching for and ascribing *good* values to the parameters of a physical model which has already been specified. Only one specific model is assumed to have been applied to a given set of measurements. But often situations may occur or questions may arise where it is important to consider alternative models during the effort to further reduce the G, and these must be investigated as well. Also, it is possible that two or more distinct models may be found that reduce the deviations in |g| to comparable magnitudes. Then, the problem of characterizing the sample becomes one of comparing models which ought to lead toward a decision about selecting the *better* model. One heuristic approach has been to select that model which provides the smallest deviations, i.e., |g|, while utilizing the fewest modeling parameters and being consistent with physical reality. However, comparisons assume some criteria or ordering, and that requires a number. So the problem becomes one of reducing a model to a number. This reduction is certainly not simple and fully merits its own discussion, e.g., hypothesis testing and decision theory. But such a discussion is beyond the scope that is intended here, so that it is expedient to direct the reader to consider simple statistics and its use of the F–statistic in assessing the so-called *goodness-of-fit* test [23]. For further discussion on topics such as parameter estimation, hypothesis testing, significance testing, and other formulations involving decision processes, the reader is advised to consider the vast literature available in statistics.

3.2 Sensitivity Analysis of Model Parameters

In the discussion of the least-squares problem in section 3.1, the main emphasis was on obtaining accurate numerical values for the model parameters. It follows naturally that the next step should involve some assessment of the uncertainties associated with those values. Fortunately, for a restricted class of linearizable problems, one may ascribe estimates to these uncertainties by utilizing a formulism similar to that used in the least-squares problem [2, 10–15, 23–25].

Consider again the functional representation of the model and its expansion about the critical or fixed point solution, e.g., using variations out to first order, i.e.,

$$\Delta_i^e - \Delta_i^m = \sum_{j=1}^{N} \left.\frac{\partial \Delta_i}{\partial b_j}\right|_o \delta b_j + \left.\frac{\partial \Delta_i}{\partial \phi}\right|_o \delta\phi + \tilde{e}_{\Delta,i} \tag{70}$$

and

$$\psi_i^e - \psi_i^m = \sum_{j=1}^{N} \left.\frac{\partial \psi_i}{\partial b_j}\right|_o \delta b_j + \left.\frac{\partial \psi_i}{\partial \phi}\right|_o \delta\phi + \tilde{e}_{\psi,i}\ , \tag{71}$$

where N refers to the number of distinct model parameters, and $\tilde{e}_i$ refers to the uncertainty associated with individual measurements of (Δ, ψ) performed by the instrument. It may also be the case during the course of the analysis of finding the critical point solution discussed in section 3.1, that some of the model parameters will have had values and uncertainties assigned to them by some earlier experiment or measurement that is external to and distinct from those being analyzed here, e.g., values found or taken from the literature; these values are assumed as given and remained unchanged throughout the calculations. Consequently, the set of model parameters may be partitioned into two disjoint sets, those (δb_j) that remain unchanged (u_j) and those (δb_j) that are allowed to vary (v_j). This allows both of the above expansions to be combined and expressed as

$$g_i = J_{v,ij} v_j + J_{u,ij} u_j + J_{\phi,i}\tilde{\phi}_i + \tilde{e}_i, \tag{72}$$

or in matrix notation as

$$\mathbf{g} = \mathbf{J}_v \mathbf{v} + \mathbf{J}_u \mathbf{u} + \mathbf{J}_\phi \tilde{\Phi} + \tilde{\mathbf{e}}, \tag{73}$$

where g_i refers to the deviations similar to eqs (60) and (61) but without including the scaling factor of $(1/\sqrt{2M})$, J refers to the Jacobian or the array of partial derivatives that is appropriate to the partitioning and evaluated at the critical point, $\tilde{\phi}_i$ involves the uncertainty in the angle of incidence *and* the assumption of stochastic independence between each measurement, and the implied summation affects only the j index. The sensitivity analysis of the model parameters centers on the procedures of assigning values of uncertainty in knowing $\mathbf{v}$.

From the representation of the *error* expression G presented in eqs (62) or (67), it follows that the minimum value of G is attributed to residuals that are due to measurement uncertainties $\tilde{e}_i$ that are assumed to behave as random variables in a statistical

sense, i.e., being stochastically independent and identically distributed. So, when G is evaluated at the critical point solution, it is assumed to be of the form

$$G_o = |\mathbf{g}|^2 = |\mathbf{e}|^2 , \tag{74}$$

so that an expansion of G about the critical point may be shifted to zero and written as

$$\tilde{G} = G - G_o = \left|\mathbf{g} - \mathbf{J}_v\mathbf{v} - \mathbf{J}_u\mathbf{u} - \mathbf{J}_\phi\tilde{\mathbf{\Phi}}\right|^2 - |\mathbf{g}|^2 . \tag{75}$$

Before presenting a formal solution for $\mathbf{v}$, it is worthwhile to consider the fact that round-off error affects the numerical accuracy of a calculation, and this is often dependent on the scaling of the problem. Again, it is convenient to follow eqs (63) to (65) and scale the numerical problem by defining the diagonal matrix $\mathbf{S}$, where

$$\mathbf{S}_{jj} = \sqrt{\left(\mathbf{J}_v^{\mathrm{T}}\mathbf{J}_v\right)_{jj}} , \tag{76}$$

with

$$\tilde{\mathbf{v}} = \mathbf{S}\mathbf{v} \tag{77}$$

and

$$\tilde{\mathbf{J}}_v = \mathbf{J}_v\mathbf{S}^{-1}, \tag{78}$$

so that eq (75) may be written as a function of $\tilde{\mathbf{v}}$, i.e.,

$$\tilde{G} = \left|\mathbf{g} - \tilde{\mathbf{J}}_v\tilde{\mathbf{v}} - \mathbf{J}_u\mathbf{u} - \mathbf{J}_\phi\tilde{\mathbf{\Phi}}\right|^2 - |\mathbf{g}|^2 . \tag{79}$$

Recalling the criteria of a critical point, i.e., $(\partial\tilde{G}/\partial\tilde{v}_j) = 0$, one finds a system of equations for $\tilde{\mathbf{v}}$, i.e.,

$$\tilde{\mathbf{J}}_v^{\mathrm{T}}\left(\mathbf{g} - \tilde{\mathbf{J}}_v\tilde{\mathbf{v}} - \mathbf{J}_u\mathbf{u} - \mathbf{J}_\phi\tilde{\mathbf{\Phi}}\right) = 0. \tag{80}$$

Defining the square matrix

$$\mathbf{A} = \tilde{\mathbf{J}}_v^{\mathrm{T}}\tilde{\mathbf{J}}_v \tag{81}$$

and the nonsquare matrix

$$\mathbf{B} = \mathbf{S}^{-1}\mathbf{A}^{-1}\tilde{\mathbf{J}}_v^{\mathrm{T}}, \tag{82}$$

and substitutes eq (82) into eq (80), a formal solution of $\mathbf{v}$ may be written as

$$\mathbf{v} = \mathbf{B}\left(\mathbf{g} - \mathbf{J}_u\mathbf{u} - \mathbf{J}_\phi\tilde{\mathbf{\Phi}}\right). \tag{83}$$

Of course, this is contingent upon the rank or invertibility of $\mathbf{A}$. This equation involves a measure of both types of errors, i.e., random error ($\mathbf{g}$,$\tilde{\mathbf{\Phi}}$) and systematic error ($\mathbf{u}$). However, due to the assumption of stochastic independence, each type of error may be considered separately.

Consider first the contributions associated with the random error. From eq (83), the random component assumes

$$v_j = B_{ji}\left(g_i - J_{\phi,i}\tilde{\phi}_i\right). \tag{84}$$

The measurement uncertainties ($\tilde{e}_i$), as well as the deviations found from fitting the model to the data (g_i), are assumed to behave as statistical random variables, which are mutually stochastically independent and identically distributed. Of course, this assumes that Δ and ψ are uncorrelated. Then, each random variable involves the same probability distribution function (PDF), and the variance may be estimated from $\mathbf{g}$. Recall that $\mathbf{g}$ involves $2M$ measurements and N model parameters, and $\tilde{\mathbf{\Phi}}$ involves M model parameters. The PDF for g_i^2 is assumed to behave as a χ^2 distribution with $2M - N$ degrees of freedom, where the variance is estimated by

$$s_g^2 = s^2(g_i) = \frac{\mathbf{g}^{\mathrm{T}}\mathbf{g}}{2M - N}. \tag{85}$$

The PDF for $\tilde{\phi}_i^2$ is assumed to behave as a χ^2 distribution with $2M - M$ degrees of freedom, where the variance is estimated by

$$s_\phi^2 = s^2(\tilde{\phi}_i^2) = \frac{\tilde{\mathbf{\Phi}}^{\mathrm{T}}\tilde{\mathbf{\Phi}}}{2M - M}, \tag{86}$$

where the $\tilde{\phi}_i$ were estimated from a set of calibration data, which is external to the set of measurements of the ellipsometric angles, and not estimated from fitting the model to the measurement of (Δ,ψ). Often, $\tilde{\phi}_i$ may be of the order of a few millidegrees. Given the estimated variance of the statistical PDF for the random variables $(g_i,\tilde{\phi}_i)$, and given the linear relationship between the model parameters and the random variables, i.e., eq

(84), one may then proceed and estimate the variance of the PDF for the model parameters $\mathbf{v}$. The expectation values are of the form

$$\langle g_i g_l \rangle = s_g^2 \, \delta_{il}, \tag{87}$$

$$\langle \tilde{\phi}_i \tilde{\phi}_l \rangle = s_\phi^2 \, \delta_{il}, \tag{88}$$

$$\langle g_i \tilde{\phi}_l \rangle = 0, \tag{89}$$

where the bra-ket notation $\langle\rangle$ refers to an expectation value, and δ refers to the Kronecker delta. The variance of the model parameters $\mathbf{v}$ is then estimated by considering the expectation value of the following quadratic form,

$$v_j v_k = B_{ji} \left(g_i - J_{\phi,i} \tilde{\phi}_i \right) B_{kl} \left(g_l - J_{\phi,l} \tilde{\phi}_l \right). \tag{90}$$

This yields the form

$$\langle v_j v_k \rangle = B_{ji} B_{ki} \left(s_g^2 + s_\phi^2 J_{\phi,i}^2 \right), \tag{91}$$

or expressed in matrix notation as

$$\langle \mathbf{v}\mathbf{v}^{\mathrm{T}} \rangle = s_g^2 \mathbf{B}\mathbf{B}^{\mathrm{T}} + s_\phi^2 \mathbf{B}\mathbf{J}_\phi^2 \mathbf{B}^{\mathrm{T}}. \tag{92}$$

Further, it may be seen that the variance of that involving some linear combination of the model parameters, e.g.,

$$\mathcal{V} = \alpha_j v_j \tag{93}$$

where α_j are real scalar coefficients, is of the form

$$\langle \mathcal{V}^2 \rangle = \alpha_j \alpha_k \langle v_j v_k \rangle. \tag{94}$$

Also, it follows from eqs (78), (81), and (82), that

$$\begin{aligned}
\mathbf{B}\mathbf{B}^{\mathrm{T}} &= \mathbf{S}^{-1}\mathbf{A}^{-1}\tilde{\mathbf{J}}_v^{\mathrm{T}}\tilde{\mathbf{J}}_v\mathbf{A}^{-1}\mathbf{S}^{-1} \\
&= \mathbf{S}^{-1}\mathbf{A}^{-1}\mathbf{S}^{-1} \\
&= \mathbf{S}^{-1}\left[\mathbf{S}^{-1}\mathbf{J}_v^{\mathrm{T}}\mathbf{J}_v\mathbf{S}^{-1}\right]^{-1}\mathbf{S}^{-1} \\
&= \mathbf{S}^{-1}\left[\mathbf{S}\left(\mathbf{J}_v^{\mathrm{T}}\mathbf{J}_v\right)^{-1}\mathbf{S}\right]\mathbf{S}^{-1} \\
&= \left(\mathbf{J}_v^{\mathrm{T}}\mathbf{J}_v\right)^{-1},
\end{aligned} \tag{95}$$

which is well known.

Next, consider the contributions associated with systematic error. From eq (83), the systematic component assumes

$$v_j = B_{ji} J_{u,ik} u_k. \tag{96}$$

By applying absolute value signs and the triangle inequality, a strict upper bound for $|v_j|$ may be found and is given by

$$|v_j| \leq |B_{ji} J_{u,ik}||u_k|. \tag{97}$$

Assuming the total magnitude of uncertainty that is assigned to a model parameter involves a direct sum of both types of errors, then one may combine eqs (91) and (97) to yield the form

$$|v_j| \leq \sqrt{B_{ji}^2 \left(s_g^2 + s_\phi^2 J_{\phi,i}^2\right)} + |B_{ji} J_{u,ik}||u_k|, \tag{98}$$

where summation is implied on the (i,k) indices. And, the uncertainty associated with a linear combination of the model parameters from eq (93) is assumed to be given by

$$|\mathcal{V}| \leq \sqrt{\alpha_j \alpha_k B_{ji} B_{ki} \left(s_g^2 + s_\phi^2 J_{\phi,i}^2\right)} + |\alpha_j||B_{ji} J_{u,ik}||u_k|, \tag{99}$$

where summation is implied on the (i,j,k) indices.

4. Manual of Operations

In order to use MAIN1, it is necessary to: (1) specify the model parameters which characterize the layered structure of the sample, (2) collect the necessary measurement data of ellipsometric angles, (3) assign size allocations of arrays which hold these data, (4) assign formats and filenames which are provided for entering the input data and receiving the output data from calculations, and (5) understand the limitations and capabilities of the software package and its utility for implementing strategies to analyze the measurement data. The filename convention assumes a filename of up to six characters and an extension of up to three characters; the standard format is given by: filename.ext. This section presents a brief overview of these considerations.

4.1 Software Development Considerations

The main program, MAIN, performs a small set of functions. First, it calls a subroutine to open the necessary data files. Second, it requests a routine to read most of the input data file. Third, it allows the user to select an option from a menu or tabulated list of command options and then calls the appropriate subroutine. Following the return from the subroutine to the main program, the main program stops; no further command options are processed.

4.1.1 Input/Output Files

MAIN1 makes use of four input/output files. One file serves for entering the input data, while three files serve for collecting output data.

X.DAT is the input data file. It contains: the model parameters which characterize the sample, the configuration of the layered structure of the sample, the ellipsometric angles of the measurement data, and the command options that provide control of the program.

X.OUT is an output data file. It contains a collective list of: all the entered input data and the general proceedings generated during the course of a calculation, as well as any informative error messages.

X.SOUT is the solution output data file that collects the breakpoint information during any grid scan of the model parameters. This information is written onto disk at periodic intervals of 15 cpu-minutes and overwrites the information from previous breakpoints, so

that only the last or most recent breakpoint is retained. The procedure for restarting or resuming a previously interrupted calculation involves the simple task of appending the output of this file X.SOUT onto the end of file X.DAT, i.e., without any intervening blank lines, and then re-executing the job. When the job is re-executed, the program will attempt to read only one set of breakpoint information in file X.DAT before resuming calculations. Consequently, it is important to replace any earlier breakpoint data in X.DAT, as appropriate.

X.PLOT is the output file whose data are formatted in a manner that is intended to be amenable for later reading and plotting.

These files are opened initally by the subroutine FILEOP. Since breakpointing involves opening/closing files, other occurrences of the open/close statements may be found in subroutines: SCAN2, SCAN2G, SCAN3, SEAMAX. The integer logical units that are associated with these files are assigned by the block data statement located in BLKDAT.FOR. These logical units are passed throughout the interior of the program by the named common block statement located in IOUNIT.

Further, all output files are deleted at the start of the program, as may be seen from looking at files MAIN.FOR and FILEOP.FOR. Consequently, it is important to append any breakpoint information contained in X.SOUT to X.DAT, as appropriate, *before* the next execution of the program, lest that which was stored in X.SOUT be overwritten and lost.

4.1.2 Allocations of Array or Work Space

In general, the allocation size of the arrays used in the program may be determined or estimated from a set of eight constants. These constants are assigned their numerical values prior to compilation by parameter statements located in file DEFNIT. Consequently, if any changes are made to these assignments, it is usually necessary to re-compile and re-link most (if not all) of the subroutines in order to incorporate the said changes into the running program. The following list includes the eight basic constants.

nsampl, the total number of samples allowed for analysis, where the sample involves some finite number of films/layers atop a substrate.

nfilms, the total number of films/layers allowed atop the substrate of any given sample.

`nlmnts`, the total number of distinct indices of refraction and extinction coefficients that are allowed during analysis.

`nwaves`, the total number of distinct wavelengths of light that are allowed to be incident on the sample.

`nbient`, the total number of distinct ambients (air, vaccum, etc.) which are external to the layers/substrate of the sample.

`nrpeat`, the total number of allowed repeats of measurements (per wavelength) on a sample.

`nanglx`, the total number of allowed angles of incidence (per wavelength) of incident light.

`nanglm`, the total number of allowed angles of incidence (per wavelength) used during multiple-angle error analysis. This is discussed in section 4.3, but involves setting the command option=6. As may be seen in the main program, MAIN, this option has been suppressed; i.e., this constant is not currently used.

In some cases, the array allocations seem needlessly overestimated. The algorithm that estimates the array allocations from the eight constants assumes a worst-case scenario with the largest size possible being calculated. This occurs especially when sizing the arrays for the Jacobian. Consequently, it is convenient to assign some upper limit to these arrays. This is the purpose of the constant named `nnjaaa`. Further, a simple internal check-test is also provided to ensure that the indices remain within bounds of the dimensions of these arrays.

4.1.3 Library Software

During the course of analyses, it is often necessary to determine the sensitivity of dependence of the calculation upon the model parameters. As discussed earlier in section 3.2 from eqs (82) and (99), it is necessary to calculate the inverse of a square matrix, i.e., the matrix formed by the product of the Jacobian and its transpose. This is accomplished or performed by subroutines from the LINPACK library of mathematical software [17]. Reference calls to these subroutines may be found in files: CORLAT.FOR, SEAMA.FOR, and SEAMAX.FOR.

Finally, several of the subroutine calls occurring in subroutine POPLAT refer to the NCAR graphics library [27,28]. But again, this has been suppressed because it refers to command option=6, which is discussed in section 4.3.

4.2 Input Data Requirements

As mentioned in section 4.1.1, the program reads all of its input data from file X.DAT. From section 2, it follows that the file contains: the model parameters, the characterization of the layered structure of the samples, and the measurement data of ellipsometric angles. This section discusses a line by line construction of file X.DAT. An example of a partially completed input file is shown in section 4.2.4. The first line in the file is reserved for entering a command option. This is discussed later in section 4.3.

The main program uses subroutine INPDAT to read the data that are presented in the following subsections. These data are stored in arrays located in the named common areas that are in file FILMMM.

Note: The examples presented in this document are for the expressed single purpose of communicating the utility of the software package and are not to be construed as an endorsement of numerical values that are associated with optical properties of media at particular wavelengths.

4.2.1 Parameters

As discussed earlier in section 2, the forward problem involves at least two kinds of parameters, those which characterize the source, i.e., wavelength and angle of incidence, and those which characterize the material sample, i.e., indices of refraction and extinction coefficient at each wavelength for each distinct type of optical media and distinct layer thicknesses. In the following subsections, the parameters are grouped and presented in the order that they ought to appear in the input file. Each subsection presents one particular type of parameter and the general format appropriate for that type of input data. Each format is demonstrated by a worked example.

4.2.1.1 Wavelengths

The first group of parameters mentioned above involves wavelengths. Here, the first line of data is the number of distinct wavelengths that are associated with the measurements

collected on all of the samples. This positive integer is denoted by m_λ. This line of data is next followed by m_λ lines of data which contain the information about the wavelengths. Each of these lines contains two numbers, one index label and one wavelength. The wavelengths are expressed in units of nanometers. The labels are indexed consecutively from 1 to m_λ. The lines are ordered so that the index label is monotonically increasing while the wavelengths are monotonically decreasing.

An example of this format is shown below for a simple case, where the measurements involve only two distinct wavelengths, such as that resulting from the use of two different lasers and a collection of multiple-angle of incidence data on one or more samples. The format would then involve three lines of data, e.g.,

```
2                ! mwaves ' total number of distinct wavelengths.
 1     632.8      !   HeNe ' wavelength ' nanometers
 2     441.6      !   HeCd   laser line
```

This format may be expressed succinctly by the following form:

$$m_\lambda \ /(i, \lambda_i)$$

where:

m_λ is the total number of distinct wavelengths;

i is an integer that indexes the wavelengths consecutively in unit increments, i.e., $i = 1, 2, 3, \ldots, m_\lambda$; and

λ_i is the wavelength of the incident light that is measured in free-space and expressed in units of nanometers. It is required that the wavelengths be ordered monotonically decreasing with increasing index, i.e., $(\lambda_i > \lambda_{i+1})$. Wavelengths associated with lower energy are entered or listed first.

The above notation is a suggestive adaptation of the argument list for a READ statement in FORTRAN. The forward slash mark '/' delimits the first line of input data. The parentheses bound items that ought to appear on each subsequent line of data until the implied DO-loop has been satisfied. Such notation has been convenient in conveying the complicated formats that are required for the input entries as well as the associated data

structures. In general, this notation will be assumed throughout the remainder of the document, as appropriate, i.e., unless stated otherwise from context.

4.2.1.2 Optical Media

4.2.1.2.1 Ambient

The ambient region refers to that spatial region which lies external to the layered structure of the sample, i.e., the (layers/substrate) system. Distinct refractive indices may be due to dispersion or fluid that is inside some filled cell containing the submersed sample. Although the corrections for the window of the cell would be important, such considerations are beyond the scope of presentation here, and the reader is well advised to refer to the outside literature. The format for entering the indices of refraction for the ambient is of the same form as that discussed for entering wavelengths, i.e.,

$$m_a \ /(i,\ n_i)$$

where:

m_a is the total number of distinct refractive indices in the ambient region;

i is an integer that indexes the refractive indices consecutively in unit increments, i.e., $i = 1, 2, 3, \ldots, m_a$; and

n_i is the index of refraction of the ambient at one particular wavelength.

To demonstrate the use of the above format, consider a case where ellipsometric measurement data were collected on a sample that had been placed in at least two distinct ambients, e.g., air and vacuum. Further, let the incident wavelengths be those mentioned in the previous example. Then, the format could be the following:

```
3                ! mbient ' number of distinct ambients
 1     1.0       !        ' vacuum
 2     1.00027   !        ' air  at 632.8 nm, HeNe laser line
 3     1.00027   !        ' air  at 441.6 nm, HeCd
```

This involves four lines of input data. Apart from the comments that are appended to the above lines of data, the format does not associate optical properties with wavelength. This

will be accomplished later in section 4.2.2, where the construction of the sample's layered structure from the constituent model parameters is discussed.

4.2.1.2.2 Layers/Substrate

For each optical medium that contributes toward the construction of the layered structure of the sample, one requires the numerical value of the index of refraction and extinction coefficient for each wavelength of incident light. For convenience, the word 'element' or its mnemonic 'lmnt' is used to associate some form of indexing of the numerical values that are assigned to the optical properties at distinct wavelengths.

As discussed earlier in section 3, the numerical values assigned to these parameters may be selected to undergo variation (vary) or remain fixed (frozen) during the course of the analysis or calculations. An integer switch (froz/vary) is provided for each model parameter, i.e., refractive index or extinction coefficient. If the switch is set equal to 0, the numerical value is frozen. If the switch is set equal to 1 upon input, the numerical value may undergo variation. The switches remain unchanged during the course of calculation.

An uncertainty value is also required for each model parameter. The magnitude of this uncertainty may serve either of two purposes. If the numerical value of the parameter undergoes variation, then the uncertainty value sets the maximum stepsize allowed for changing the numerical value of the parameter between consecutive iterations of the calculation. If the numerical value of the parameter is selected to remain frozen, then the uncertainty value is used during the sensitivity analysis calculation to estimate the uncertainty values of other 'vary' model parameters. This is discussed further in section 3.2.

The format for entering the optical properties of the layers/substrate is:

$$m_{\text{lmnt}} \ /(i,\ n_i,\ \delta n_i,\ v_i)$$

where:

m_{lmnt} is the total number of distinct 'elements' involving refractive indices and extinction coefficients for each distinct type of optical media and wavelength;

i is an integer that indexes the line entries of data consecutively in unit increments, i.e., $i = 1, 2, 3, \ldots, m_{\text{lmnt}}$;

n_i is the numerical value that is assigned to either an index of refraction or extinction coefficient of some particular optical medium at some particular wavelength;

δn_i is the magnitude of uncertainty that is assigned to the numerical value of the model parameter n_i; and

v_i is the integer (froz/vary) switch with value of either 0 or 1, respectively.

To demonstrate this format, consider the well-known case involving a silicon substrate that supports a thermally grown oxide layer. Further, assume that the wavelengths associated with the measurements were those mentioned earlier from the example in section 4.2.1.1. The format could then be something like the following:

```
7                                   ! mlmnts ' (n,k | wavelength, element)
 1   3.882    0.002    0    !    n,nu,ivary ' Silicon ' at 632.8 nm, HeNe
 2   0.019    0.002    0    !    k,ku,ivary ' Silicon
 3   1.457    0.002    0    !    n,nu,ivary ' SiO2 amorphous glass
 4   0.0      0.0      1    !    k,ku,ivary ' SiO2    ' at   HeNe or HeCd
 5   4.753    0.002    0    !    n,nu,ivary ' Silicon ' at 441.6 nm, HeCd
 6   0.163    0.002    0    !    k,ku,ivary ' Silicon
 7   1.466    0.002    0    !    n,nu,ivary ' SiO2 amorphous glass
```

which involves only eight lines of input data. This example involves four sets of information. It contains the optical properties of crystalline silicon and amorphous silicon dioxide at two distinct wavelengths. This is a simple tabulation of numerical values; ordering is not important among the last seven line entries. Note that no reference to wavelength is indicated here; thus the specifications regarding the optical properties are still incomplete. The manner in which the optical properties are associated with the appropriate wavelength will be discussed later in section 4.2.2. (Again, that section discusses how the layered structure of the samples is constructed layer by layer from the constituent model parameters as a function of wavelength. That section will conclude the characterization of the sample.)

Lastly, the example shows that one model parameter was selected to have its numerical value undergo variation, i.e., the extinction coefficient for amorphous silicon dioxide at either wavelength. Note also that the uncertainty value was set to zero. This condition is check-tested upon input, so the user would be notified. (Setting the uncertainty value to

a nonzero value circumvents any notification.) The reason for the check-test is that the program uses the uncertainty value to limit the stepsize for updating the numerical value of the associated model parameter. With an allowed maximum stepsize of zero, the program would be unable to reduce the numerical value of the error expression, at least with respect to this model parameter. Being unable to justify further calculation, it would terminate. Such control is useful in other situations as well, e.g., limiting the calculation to a single evaluation of the residual or discerning relative contributions made to the uncertainty value of some 'vary' model parameter. This may be accomplished by assigning a sufficiently small number to the uncertainty value on input.

4.2.1.3 Layer Thickness

The program also requires the thicknesses of the layers that contribute to the layered structure of the samples. The unit of thickness is nanometers. Again, in similar fashion as that presented in the previous subsection, the format is of the form:

$$m_z \ /(i,\ z_i,\ \delta z_i,\ v_i)$$

where:

m_z is the total number of distinct thicknesses that contribute to the layered structure of the sample;

i is an integer that indexes the line entries consecutively in unit increments, i.e., $i = 1, 2, 3, \ldots, m_z$;

z_i is the thickness of a layer measured in nanometers;

δz_i is the magnitude of uncertainty in knowing the thickness z_i; and

v_i is the integer (froz/vary) switch with value 0 or 1, respectively.

To demonstrate this format, let the following example build upon information presented from previous examples. Consider a sample involving two layers atop a substrate of silicon, where one layer is atop the other layer. Let the layer adjacent to the substrate be a thermally grown oxide, and let the layer adjacent to the ambient region be a layer of silicon. Then, the layered structure is of the ordered form:

(ambient / silicon / oxide / silicon substrate).

Suppose further that the top layer of silicon is 50 nm thick, that the oxide layer is 100 nm thick, and that a reasonable initial estimate of the uncertainty is subjectively chosen to be 2 nm. This construct would require three lines of input data. An example of the format may be as the following:

```
2                                  ! mfilmm ' thicknesses
 1     50.0     2.0     0     !     z,zu,ivary '     top layer, Si
 2    100.0     2.0     1     !     z,zu,ivary ' bottom layer, SiO2
```

This tabulation would include the distinct thickness from all samples; thus ordering among thicknesses is not important here. Further, note that this format does not associate thickness with a sample or with ordering of layers on a sample. The ordering of the layers is discussed in the next section. And lastly, one may see from the above example that the oxide thickness has been judiciously chosen to undergo variation.

4.2.2 Sample Characterization or Construction

Since the basic constituent model parameters have been presented, it is now possible and necessary to discuss how: (1) the layered structure of each sample is constructed from the constituent model parameters, (2) the wavelengths are associated with the optical properties, and (3) the ordering of the wavelengths affects the ordering of the input of the measurement data. The last item is discussed in the next section, i.e., section 4.2.3.

First, it is important to discuss the meaning of the word 'sample.' A sample is defined as being that structure of the (layers/substrate) system that is subjected to ellipsometric measurement. Characterization of the sample refers to the index-labeling procedure that associates the spatial regions of the (ambient/layers/substrate) system with the appropriate model parameters at given wavelengths, so that the forward scattering problem is completely defined. Measurements on a sample may involve one or more wavelengths, as well as one or more ambients. For any given sample, the optical properties may change as a function of wavelength, i.e., exhibit dispersion, whereas layer thicknesses remain unchanged. Here, the representation of the optical media involves a distinct model parameter per distinct wavelength. A sample is constructed in a layer-by-layer fashion by specifying the optical properties for each spatial region for each incident wavelength. Admittedly,

this parameterization of optical media is by design suitable only for analyzing media involving very few wavelengths, very few ambients, but many-angle of incidence measurement data.

Recall, the inverse scattering problem is formulated as a least-squares problem. This involves a series of linear algebra problems. Consequently, the matrices associated with individual samples may be combined so that several samples may be analyzed together. Accordingly, the program is said to have a multiple-sample capability. The maximum number of samples that may be analyzed together is set by the parameter `nsampl`, as mentioned in section 4.1.2.

Recall again that the previous section discussed formats for inputting model parameters. Each model parameter has a distinct index label; each line entry following the first line is indexed. The actual ordering among the line entries within any given set of model parameters is relatively unimportant, i.e., apart from wavelength. Here, the index labels are used for specifying the configuration of the (ambient, wavelength, sample) system. The sample is characterized by an ordered set of integers that point into arrays that store the numerical values associated with the model parameters.

For convenience, the integers are ordered in a format compatible with the collection algorithm of the laboratory instrumentation. Here, measurements are collected by scanning the angles of incidence for each given wavelength. The ellipsometric measurement data are ordered into a data structure that follows the FORTRAN indexing convention for multiply-indexed arrays. That data structure is of the following form:

(*angles, repeats, ambients, wavelengths, samples*)

where:

samples indexes the set of distinct samples that were subjected to ellipsometric measurement;

wavelengths indexes the set of distinct wavelengths used during measurement on a given sample;

ambients indexes the set of distinct ambients used during measurement involving a given wavelength and sample;

repeats indexes the sets of distinct repeats of multiple-angle measurements performed on a system of given ambient, wavelength, and sample; and

angles indexes the sets of distinct angles of incidence used during measurement on a system of given repeat index, ambient, wavelength, and sample.

This form suggests that the samples are considered individually. First, the sample is specified; then, the incident wavelengths are specified; this is followed by the specification of the ambient, i.e., the value of its refractive index at the associated wavelength. Last, the angle of incidence is specified to completely define the forward problem. The *repeats* serve as convenient (or artificial) partitioning of the multiple-angle of incidence data, e.g., distinguishing data collected on different days.

To implement the above organization and thereby characterize the (ambient, wavelength, sample) system, the program reads a tabulated set of integers line by line. The first line of data specifies the number of samples; this is denoted by m_{samples}. This is followed by a series of lines that are grouped in a sample-by-sample manner.

For convenience, consider the lines that are associated with the j^{th} sample. The first line of data for this subset should specify: the index label, the number of layers, and the number of wavelengths. These are denoted, respectively, by: j_{sample}, $m_{z,j}$, and $m_{\lambda,j}$. The next line should specify the wavelength and number of ambients that have been involved with this sample and wavelength. The two integers are denoted, respectively, by: $\tilde{\lambda}_{i,j}$ and $m_{a,ij}$, where the first entry is the index label for the wavelength, and i is a local index of the ordered wavelengths, i.e., $i = 1, 2, 3, \ldots, m_{\lambda,j}$.

Now that the sample and wavelength have been indicated, consider next the structure of the layers/substrate. The layers/substrate are considered in succession, i.e., starting with the top layer or layer 1, which is adjacent to the ambient region. Each layer/substrate region is associated with a line of integer data. The integers include the index labels for the layer, refractive index, extinction coefficient, and thickness, in that order. This is indicated, respectively, by: l, $\tilde{n}_{l,ij}$, $\tilde{k}_{l,ij}$, and $\tilde{z}_{l,j}$, where the tilde refers to that implying index labels. The substrate assumes a layer index label of $m_{z,j}+1$, but no index label for thickness is required. This completes the specification of the layers/substrate system for the j^{th} sample.

The next set of $m_{a,ij}$ lines of data is used to specify the ambients and *repeats*, i.e., one line for each ambient. Each line should contain two integers; they are the index label of the ambient at associated wavelength and the number of repeats of the multiple-angle of incidence data. These are denoted, respectively, by: $\tilde{a}_{\ell,ij}$ and $r_{\ell,ij}$, where ℓ is a local index of the ambients, i.e., $\ell = 1, 2, 3, \ldots, m_{a,ij}$. Here, a *repeat* value of 1 is associated with one set of measurements, i.e., not two. This completes the specification of the (ambient/layers/substrate) system, as well as the implied organization of the collection of measurement data.

Regarding the organization for these specification data, the general format is given by the following form:

m_{samples}

1 $\cdots$

$\vdots$

$(j-1)_{\text{sample}}$ $\cdots$

$\vdots$

j_{sample} $m_{z,j}$ $m_{\lambda,j}$

$\tilde{\lambda}_{1,j}$ $\cdots$

$\vdots$

$\tilde{\lambda}_{i-1,j}$ $\cdots$

$\vdots$

$\tilde{\lambda}_{i,j}$ $m_{a,ij}$

1 $\tilde{n}_{1,ij}$ $\tilde{k}_{1,ij}$ $\tilde{z}_{1,j}$

$\vdots$ $\vdots$ $\vdots$ $\vdots$

l $\tilde{n}_{l,ij}$ $\tilde{k}_{l,ij}$ $\tilde{z}_{l,j}$

$\vdots$ $\vdots$ $\vdots$ $\vdots$

$m_{z,j}$ $\tilde{n}_{m_\Gamma ij}$ $\tilde{k}_{m_\Gamma ij}$ $\tilde{z}_{m_\Gamma j}$

$(m_{z,j}+1)$ $\tilde{n}_{s,ij}$ $\tilde{k}_{s,ij}$

$\tilde{a}_{1,ij}$ $r_{1,ij}$

$\vdots$ $\vdots$

$\tilde{a}_{l,ij}$ $r_{l,ij}$

$\vdots$ $\vdots$

$\tilde{a}_{m_a,ij}$ $r_{m_a,ij}$

$\tilde{\lambda}_{i+1,j}$ $\cdots$

$\vdots$

$\tilde{\lambda}_{m_\lambda,j}$ $\cdots$

$\vdots$

$(j+1)_{\text{sample}}$ $\cdots$

$\vdots$

m_{sample} $\cdots$

$\vdots$

where:

m_{samples} is the total number of distinct samples that are/were subject to ellipsometric measurement.

j_{sample} is an integer that indexes the samples consecutively in unit increments, i.e., $j = 1, 2, 3, \ldots, m_{\text{samples}}$.

$m_{z,j}$ is the total number of layers that lie atop the substrate of sample j. The top layer is adjacent to the ambient and is indexed locally as being equal to one. The bottom layer is adjacent to the substrate and is indexed locally as being equal to $m_{z,j}$.

$m_{\lambda,j}$ is the total number of distinct wavelengths that are used in the measurements involving sample j.

$\tilde{\lambda}_{i,j}$ is the integer index label of the appropriate wavelength that is associated with the i^{th} wavelength incident on sample j. Note that i is indexed locally, i.e., $i = 1, 2, 3, \ldots, m_{\lambda,j}$. Further, it is requested that the set of measurement data be ordered upon input so that the wavelengths be decreasing with increasing index, or equivalently, that the index label $\tilde{\lambda}$ be monotonically increasing with index, i.e., $(\tilde{\lambda}_{i,j} < \tilde{\lambda}_{i+1,j})$.

$m_{a,ij}$ is the total number of distinct ambients that involve the i^{th} wavelength incident on sample j.

l is an integer local index of the layers and substrate of the sample, i.e., $i = 1, 2, 3, \ldots, m_{z,j},\ m_{z,j}+1$. The layers are indexed consecutively; the layer adjacent to the ambient is labeled 1; the layer adjacent to the substrate is labeled $m_{z,j}$, and the substrate is labeled $m_{z,j}+1$.

$\tilde{n}_{l,ij}$ is the integer index label of the appropriate index of refraction that involves the l^{th} layer and the i^{th} wavelength incident on sample j.

$\tilde{k}_{l,ij}$ is the integer index label of the appropriate extinction coefficient that involves the l^{th} layer and the i^{th} wavelength incident on sample j.

$\tilde{z}_{l,ij}$ is the integer index label of the appropriate thickness of the l^{th} layer of sample j.

$\tilde{n}_{s,ij}$ is the integer index label of the appropriate index of refraction of the substrate that involves the i^{th} wavelength incident on sample j.

$\tilde{k}_{s,ij}$ is the integer index label of the appropriate extinction coefficient of the substrate that involves the i^{th} wavelength incident on sample j.

$\tilde{a}_{\ell,ij}$ is the integer index label of the index of refraction of the ℓ^{th} ambient that is used with the i^{th} wavelength incident on sample j. Here, ℓ is indexed locally, i.e., $(\ell = 1, 2, 3, \ldots, m_{a,ij})$. Furthermore, it is requested that the measurement data be ordered upon input so that $(\tilde{a}_{\ell,ij} < \tilde{a}_{\ell+1,ij})$.

$r_{\ell,ij}$ is the total number of sets of *repeated* measurements that involve the ℓ^{th} ambient with the i^{th} wavelength incident on sample j. Note that a *repeat* value of 1 relates to reading one set of data, i.e., not two. Each set involves a collection of multiple-angle measurements.

Again, the optical properties become associated with the appropriate wavelength via explicit use of index label. This simplifies any inherent coupling between distinct samples. Distinct samples may have one or more model parameters in common.

To demonstrate this format, consider again the example presented in the previous section that discussed layer thicknesses, i.e., section 4.2.1.3. There, the sample involved two layers, i.e., a top layer of silicon, a layer of amorphous silicon dioxide, and a substrate of silicon. Suppose that this sample is measured by ellipsometry using wavelengths from the two aforementioned laser lines. Assume further that sets of multiple-angle measurements were performed in air, as well as in vacuum. After performing these measurements, suppose that this sample is subjected to some etchant, which is able to remove the top layer and leave the oxide layer unaffected. Let this become the 'second' sample. Let this 'second' sample be subjected to a similiar set of ellipsometric measurements, but only in air. Note that the model parameters of the oxide layer and substrate are common to both samples. The format for this example could be as the following.

```
2                          ! msample ' number of samples
 1    2    2               !    isample, mfilm, mwave
 1    2                    !       iwave, mbient       ' (HeNe)
 1    1    2    1          !          n,k,z            ' Si,     50 nm
 2    3    4    2          !          n,k,z            ' SiO2,  100 nm
 3    1    2               !          n,k              ' Si,   substrate
 1    1                    !          imbient, mrpeat  ' vacuum
```

```
2   2                   !          imbient, mrpeat   ' air
2   1                   !       iwave, mbient        ' HeCd
1   5   6   1           !          n,k,z             ' Si,     50 nm
2   7   4   2           !          n,k,z             ' SiO2,   100 nm
3   5   6               !          n,k               ' Si,     substrate
3   1                   !          imbient, mrpeat   ' air
2   1   2               !    isample, mfilm, mwave
1   1                   !       iwave, mbient        ' HeNe
1   3   4   2           !          n,k,z             ' SiO2,   100 nm
2   1   2               !          n,k               ' Si,     substrate
2   1                   !          imbient, mrpeat   ' air
2   1                   !       iwave, mbient        ' HeCd
1   7   4   2           !          n,k,z             ' SiO2,   100 nm
2   5   6               !          n,k               ' Si,     substrate
3   1                   !          imbient, mrpeat   ' air
```

Here, the example includes a repeated set of measurements on the first sample in air using light from the HeNe laser.

4.2.3 Measurement Data (Δ, ψ)

The measurement data are organized in a manner similar to that discussed in the preceding section, i.e., section 4.2.2. There, the data structure assumed a form given by:

(*angles, repeats, ambients, wavelengths, samples*).

Again, the measurement data are entered into the data file line by line. Each line contains one measurement of (Δ, ψ), as well as the angle of incidence and associated uncertainties. Here, the format is of the following general form:

$$m_{\phi,\hat{r}\ell ij},\ \tilde{a}_{\ell,ij},\ \tilde{\lambda}_{ij}\ /(\alpha,\ \phi_\alpha,\ \delta\phi_\alpha,\ \Delta_\alpha,\ \delta\Delta_\alpha,\ \psi_\alpha,\ \delta\psi_\alpha)_{\hat{r}\ell ij},$$

where:

$m_{\phi,\hat{r}\ell ij}$ is the total number of multiple-angle measurements of ellipsometric angles (Δ, ψ) that were collected during the $\hat{r}^{\text{th}}$ repeat set, i.e., $(1 \leq \hat{r} \leq r_{\ell,ij})$, of data measurements on the system involving the ℓ^{th} ambient with the i^{th} wavelength incident on the j^{th} sample.

$\tilde{a}_{\ell,ij}$ is the integer index label of the index of refraction of the ℓ^{th} ambient associated with the i^{th} wavelength incident on the j^{th} sample. This is the same label mentioned in the previous section.

$\tilde{\lambda}_{ij}$ is the integer index label appropriate to the i^{th} wavelength incident on the j^{th} sample. This is the same label mentioned in the previous section.

α is an integer that indexes the multiple-angle measurements consecutively in unit increments, i.e., $\alpha = 1, 2, 3, \ldots, m_{\phi,\hat{r}ij}$.

ϕ_α is the angle of incidence measured in degrees, where a value of zero relates to that of normal incidence.

$\delta\phi_\alpha$ is the magnitude of uncertainty in the angle of incidence as determined from calibration of laboratory instrumentation.

Δ_α is measured in degrees, i.e., $(0 \leq \Delta < 360)$, assuming the Nebraska convention, i.e., $R_{p,H}$ and eq (51).

$\delta\Delta_\alpha$ is the magnitude of uncertainty associated with measuring Δ_α.

ψ_α is measured in degrees, i.e., $(0 \leq \psi \leq 90)$.

$\delta\psi_\alpha$ is the magnitude of uncertainty associated with measuring ψ_α.

To demonstrate this format, let the measurements be those found or generated by solving the forward scattering problem for the sample configurations presented in the previous example. Let the measurements involve only a few angles of incidence data. These measurements would be *exact*. Let the uncertainty associated with ϕ, Δ, and ψ be given by 0.01, 0.05, and 0.05 degrees, respectively. Although these values naturally depend upon the instrumentation, their magnitudes are chosen subjectively here for convenience. The format could then be as the following:

```
2   1   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! vacuum
1    65.0000    0.0100  173.5043    0.0500   37.0579    0.0500    ! HeNe
2    70.0000    0.0100  172.2664    0.0500   34.3707    0.0500
3   2   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
1    65.0000    0.0100  173.5101    0.0500   37.0527    0.0500    ! HeNe
2    70.0000    0.0100  172.2750    0.0500   34.3633    0.0500
3    75.0000    0.0100  170.0354    0.0500   30.1573    0.0500
2   2   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! repeat
1    67.5000    0.0100  172.9647    0.0500   35.8405    0.0500
2    72.5000    0.0100  171.3529    0.0500   32.5214    0.0500
4   3   2 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
1    67.5000    0.0100  286.8467    0.0500   30.2644    0.0500    ! HeCd
2    70.0000    0.0100  297.5219    0.0500   31.2799    0.0500
3    72.5000    0.0100  307.5732    0.0500   32.6322    0.0000
4    75.0000    0.0100  316.8687    0.0500   34.2225    0.0000
2   2   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
1    70.0000    0.0100   79.8092    0.0500   41.0455    0.0500    ! HeNe
2    73.0000    0.0100   68.2502    0.0500   41.0304    0.0500
```

```
2   3   2 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
1    70.0000     0.0100  211.2235     0.0500   77.3289     0.0500    ! HeCd
2    73.0000     0.0100  216.1556     0.0500   84.4928     0.0500
```

Incidentally, these data are retained in arrays that are located in the named common area, exprmt.

4.2.4 Combining

The input data file is composed of at least two kinds of information, those which define the system model and associated measurements, and those which define the options for processing these data. Thus far, only the first kind has been presented. The input data file is constructed, i.e., in part, by simply combining the previous formats in the order as presented. Such involves the wavelengths, the model parameters, the characterization of the samples, and the measurement data. No intervening blank lines are allowed.

Combining all the examples as presented thus far, and indexing the lines for convenience, i.e., the data file X.DAT should *not* actually contain such indexing, the combined format would be:

```
1  ...                                ! option ' reserved for some selected option
2   2                                 ! mwaves ' total number of distinct wavelengths.
3    1     632.8                      !   HeNe ' wavelength ' nanometers
4    2     441.6                      !   HeCd   laser line
5   3                                 ! mbient ' number of distinct ambients
6    1     1.0                        !        ' vacuum
7    2     1.00027                    !        ' air  at 632.8 nm, HeNe laser line
8    3     1.00027                    !        ' air  at 441.6 nm, HeCd
9   7                                 ! mlmnts ' (n,k | wavelength, element)
10   1   3.882    0.002    0         !    n,nu,ivary ' Silicon ' at 632.8 nm, HeNe
11   2   0.019    0.002    0         !    k,ku,ivary ' Silicon
12   3   1.457    0.002    0         !    n,nu,ivary ' SiO2 amorphous glass
13   4   0.0      0.0      0         !    k,ku,ivary ' SiO2    ' at   HeNe or HeCd
14   5   4.753    0.002    0         !    n,nu,ivary ' Silicon ' at 441.6 nm, HeCd
15   6   0.163    0.002    0         !    k,ku,ivary ' Silicon
16   7   1.466    0.002    0         !    n,nu,ivary ' SiO2 amorphous glass
17  2                                 ! mfilmm ' thicknesses
18   1     50.0     2.0     0        !    z,zu,ivary '    top layer, Si
19   2    100.0     2.0     1        !    z,zu,ivary ' bottom layer, SiO2
20  2                                 ! msample ' number of samples
21   1    2    2                     !    isample, mfilm, mwave
22   1    2                          !       iwave, mbient       ' (HeNe)
23   1    1    2    1                !         n,k,z             ' Si,     50 nm
24   2    3    4    2                !         n,k,z             ' SiO2,  100 nm
25   3    1    2                     !         n,k               ' Si,   substrate
```

```
 1    1                    !          imbient, mrpeat   ' vacuum
 2    2                    !          imbient, mrpeat   ' air
 2    1                    !       iwave, mbient        ' HeCd
 1    5    6    1          !          n,k,z             ' Si,     50 nm
 2    7    4    2          !          n,k,z             ' SiO2,  100 nm
 3    5    6               !          n,k               ' Si,   substrate
 3    1                    !          imbient, mrpeat   ' air
 2    1    2               !    isample, mfilm, mwave
 1    1                    !       iwave, mbient        ' HeNe
 1    3    4    2          !          n,k,z             ' SiO2,  100 nm
 2    1    2               !          n,k               ' Si,   substrate
 2    1                    !          imbient, mrpeat   ' air
 2    1                    !       iwave, mbient        ' HeCd
 1    7    4    2          !          n,k,z             ' SiO2,  100 nm
 2    5    6               !          n,k               ' Si,   substrate
 3    1                    !          imbient, mrpeat   ' air
  2   1   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! vacuum
  1    65.0000    0.0100  173.5043    0.0500   37.0579    0.0500    ! HeNe
  2    70.0000    0.0100  172.2664    0.0500   34.3707    0.0500
  3   2   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
  1    65.0000    0.0100  173.5101    0.0500   37.0527    0.0500    ! HeNe
  2    70.0000    0.0100  172.2750    0.0500   34.3633    0.0500
  3    75.0000    0.0100  170.0354    0.0500   30.1573    0.0500
  2   2   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! repeat
  1    67.5000    0.0100  172.9647    0.0500   35.8405    0.0500
  2    72.5000    0.0100  171.3529    0.0500   32.5214    0.0500
  4   3   2 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
  1    67.5000    0.0100  286.8467    0.0500   30.2644    0.0500    ! HeCd
  2    70.0000    0.0100  297.5219    0.0500   31.2799    0.0500
  3    72.5000    0.0100  307.5732    0.0500   32.6322    0.0000
  4    75.0000    0.0100  316.8687    0.0500   34.2225    0.0000
  2   2   1 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
  1    70.0000    0.0100   79.8092    0.0500   41.0455    0.0500    ! HeNe
  2    73.0000    0.0100   68.2502    0.0500   41.0304    0.0500
  2   3   2 ' mangl,imbien,iwave/(phi,delta,psi) (angle, d_angle)   ! air
  1    70.0000    0.0100  211.2235    0.0500   77.3289    0.0500    ! HeCd
  2    73.0000    0.0100  216.1556    0.0500   84.4928    0.0500
 ...              ! reserved for further/other option information
```

4.3 Command Options

The last type of input data that must be included in the file X.DAT is that associated with directing the use of the software package. To direct the execution of the software program package, a menu-driven decision-tree of command options is made available to the user. The first level of the tree involves a menu of five options. Here, one option is selected per execution of the program, and the selection of that option forms the first line of the input to the data file X.DAT, as mentioned previously in section 4.2. The program reads this line and the following lines shown in the previous subsection (4.2.4), i.e., model

parameters, sample characterization, and measurement data, where appropriate. After reading this input of data, the program branches to the appropriate subroutines. Upon completion from these routines (at level one), the main program stops. Again, as was mentioned earlier, the main program does not loop back to a previous condition where a request is made of the user to select another option at level one.

Before reading the first line of the input file, the program writes to the output file X.DAT the following menu of available options:

```
options:   1,  forward problems, plots, ...
           2,  search       (     vary)
           3,  search grid  (     vary)
           4,  search grid  (froz,vary)
           5,  sensitivity analysis
           6,  MAE plots ' uncertainty

Enter:   option '
```

Where the options (level one):

1, requests the program to perform one of several simple tasks, such as providing a set of tabulated output that is amenable for plotting, or initiating a series of calculations of the forward scattering problem. No iterative calculations are induced, i.e., no minimizations or inversions of the ellipsometric equations. This is discussed further in section 4.3.1.

2, requests the program to invert the ellipsometric equations by performing a series of unconstrained optimization calculations. The variation of numerical values of selected model parameters is generally unbounded. This is presented in section 4.3.2.

3, requests the program to invert the ellipsometric equations by performing a grid scan over a selected set of model parameters. A series of constrained optimization calculations is initiated at each point of the grid of model parameters. This is discussed further in section 4.3.3.

4, requests the program to invert the ellipsometric equations by performing a grid scan over a selected set of model parameters, where the grid scans are held local to the individual sample. Hence, the coupling between samples is suppressed. A

series of constrained optimization calculations is initiated at each point in the grid scan. This is discussed further in section 4.3.4.

5, requests the program to initiate a sensitivity analysis of selected model parameters. The uncertainty associated with assigning a numerical value to a model parameter is estimated from the covariance matrix elements and the relative contributions of uncertainty from the frozen model parameters. This is discussed further in section 4.3.5.

6, requests the program to determine which selection of multiple angles (of incidence) minimizes the calculated magnitude of uncertainty in the numerical value of some selected model parameter. This involves a grid scan over groups/sets of incident angles. The magnitude of uncertainty is calculated as a function of these groups/sets. This is discussed further in section 4.3.6.

These options provide the current range of utility of the program. Regarding the general nature of the options, it may be expected that option one is more useful towards the beginning of analysis with plots, option five is more useful towards the conclusion of analysis, while options two and three form the primary tools during analysis. Option four provides a measure of convenience with uncoupling samples. This not only reduces the sizes of the matrices, but also this tends to remove redundant calculations from the grid scan. Since the grid scans are constructed from a simple nesting of DO-loops, a reduction may be expected to occur whenever the grid scans involve model parameters which are not common to every sample. This reduction and savings can become significant.

These options are discussed further in the following subsections. Here, it is convenient to simply follow the decision-tree of options.

4.3.1 Forward Problems, Plots, ...

Upon selection of option one (at level one), the main program calls subroutine PLTDAT. The routine is partitioned to serve five distinct tasks. Hence, another menu of options is afforded to the user. Regarding the calling sequence of routines as initiated from the main program, this involves level two. Accordingly, the menu of options is:

```
PLTDAT,  forward scattering problems, plots, ...
         This menu of options is at level two.
```

```
options:   1,  plot (z,n,k)
           2,  plot deviations of fit,   g = experiment-model
           3,  model ---> experiment/measurement data,
               use model to simulate measurement data,
               format is suitable to reading as input,
               (angles,repeats,ambients,waves,samples)
           4,  use model at measurement/scan points,
               request another menu to format output,
               output " (delta,psi,intensity, R,dR, ...)
           5,  scan grid,  plot |g|,        1D or 2D

Enter:   option "
```

where the options (level two):

1, requests the program to output the profile of the index of refraction and extinction coefficient as a function of depth through the layered structure of the sample. The simple format is intended to be suitable for later plotting. The output is written to the plot file X.PLOT.

2, requests the program to calculate and tabulate the deviations between the measurement data and that calculated by the model. The deviations are ordered similar to that of the measurement data. The output is written to the plot file X.PLOT.

3, requests the program to use the model to simulate measurement data. The format is suitable for use in file X.DAT. The output is written to both files, X.OUT and X.PLOT.

4, requests the program to use the model to calculate some selected field quantity. The selections include: the ellipsometic angles, reflected field intensities, and matrix elements of the Jacobian. The output is written to file X.PLOT. To assist the selection process, two menus are provided. These are presented later in this section.

5, requests the program to initiate a grid scan of model parameters and evaluate the error expression |g| for each point on the grid. Here, the grid is either one or two dimensional; i.e., only one or two model parameters may undergo variation. Here, it is necessary to specify the domain of the grid. The format for this input is presented later in this subsection. The output is written to file X.PLOT.

As mentioned above, regarding options four and five, both present menus to assist the se-

lection process in the decision-tree of options. These menus are presented next.

For the case regarding option four (level two), the program responds by presenting the first of two menus. The first menu presents a list of selected field quantities that are available for formatted output.

```
PLTDAT,  select the format of the output table.
         This menu of options is at level three.

options:   1,  (i,ai,  delta,psi,  intensity      )
           2,  (i,ai,  |Rs|,|Rp|,  |Rs|**2,|Rp|**2)
           3,  (i,ai,  |Rs|,|Rp|,  intensity      )
           4,  (i,ai,  d/d parameter (delta, psi) )

Enter:    option '
```

where:

1, involves the ellipsometric angles and the intensity I, where $I = \frac{1}{2}(|R_s|^2 + |R_p|^2)$.

2, involves the magnitude and intensity of the field amplitude for each polarization.

3, involves the magnitude of the field amplitudes and the total intensity.

4, involves the matrix elements of the Jacobian. The partial derivative is specified by the (froz/vary) switch of the model parameter. Either zero or one model parameters may be selected as undergoing variation. For the case involving zero model parameters undergoing variation, the partial derivative is taken with respect to the angle of incidence. Here, the Jacobian matrix elements are expressed in the physics convention, i.e., *not* the engineering convention. Only that associated with Δ is affected, i.e., inducing a minus sign.

i, refers to the local indexing of the multiple angles of incidence, which involves one given configuration of the system, i.e., (*repeat, ambient, wavelength, sample*).

ai, refers to the angle of incidence measured in degrees.

Continuing the case regarding option four (level two), the second menu of options (level three) involves selecting the appropriate grid of incident angles used in the tabulation.

```
PLTDAT,   select the domain of incident angles.
          This menu of options is at level three.
```

```
options:   1,  at experiment/measurement points
           2,  grid scan, incident angles ° (1,89)

Enter:     option
```

where the options refer to the incident angles associated with

1, the measurement data of ellipsometric angles (Δ, ψ), and

2, a grid, indexed from 1 to 89, with stepsize of one (degree). Such is usually more than adequate for plotting the field as a function of incident angle.

Finally, consider the case involving option five (level two). Here, the domain of the grid needs to be specified. Since each dimension of the grid involves one of the distinct model parameters, it is necessary to specify: the model parameters, their bounds, and their stepsizes. The model parameters are selected via the (froz/vary) switch; only those indicated as *varying* contribute to the grid. Since the plots are either one or two dimensional, only one or two model parameters may have numerical values being indicated as varying. The format for reading this grid information follows the convention of DO-loop specification, and is given by the form:

$$i_p,\ p_1,\ p_2,\ p_3,$$

where i_p refers to the index label of the model parameter p, and p_j refers to numerical value of the model parameter associated with the initial value, final value, and stepsize. Further, it is required that the lines be ordered according to the input data requirements presented earlier in section 4.2, i.e., optical media first, then thicknesses.

4.3.2 Search (vary)

Upon selection of option two (level one), the main program calls subroutine ZOOM. This routine initiates and maintains the search for a minimum to the *error* expression as an unconstrained optimization problem. The (froz/vary) switch specifies the model parameters which have numerical values that are undergoing variation. At least one numerical value must undergo variation per measurement of (Δ, ψ), so that their associated rows in the Jacobian are not zero. Note that an iterative method requires an initial solution. Here, the initial value solution is given by the initial input of model parameters as presented in section 4.2. The problem is now completely defined; no further specifications are necessary;

no further menus need to be issued to the user.

Since the optimization algorithm is unconstrained, unphysical fixed-point solutions are possible and likely during analyses. Such has been discussed earlier in section 3.

The following presents a brief orientation regarding the internal organization of the routine ZOOM. The organization is that for setting up an iterative loop. It calls subroutine ASMBL to construct the Jacobian matrix. It calls subroutine CGNL to solve for the Newton step. It updates the numerical value of the selected model parameters and tests the rate of reduction of the *error* expression. The progress regarding the rate of reduction is written to the output file, X.OUT. If the rate of reduction is sufficiently small to merit no further expenditure, it returns; otherwise, it continues iterating. Such calculations usually involve short durations of time, breakpointing is not necessary, and so it was not incorporated into the routine.

Upon completion of the above tasks, the program reports its *best* fixed-point solution. The deviations between the measurement data and that of the model are written to the plot file, X.PLOT, using subroutine PLTDAT. Statistics regarding the deviations are provided by the routine STAT22. It reports the statistical means, standard deviations, and the correlation coefficient of the deviations g. Here, the mean involving Δ is defined by

$$\langle g_\Delta \rangle \equiv \frac{1}{M} \sum_{i=1}^{M} g_{\Delta,i},$$

the variance is defined by

$$\langle (g_\Delta - \langle g_\Delta \rangle)^2 \rangle \equiv \frac{1}{M} \sum_{i=1}^{M} (g_{\Delta,i} - \langle g_\Delta \rangle)^2 ,$$

where the square root estimates the standard deviation, and the covariance is defined by

$$\langle (g_\Delta - \langle g_\Delta \rangle)\,(g_\Psi - \langle g_\Psi \rangle) \rangle \equiv \frac{1}{M} \sum_{i=1}^{M} (g_{\Delta,i} - \langle g_\Delta \rangle)\,(g_{\Psi,i} - \langle g_\Psi \rangle) ,$$

and the correlation coefficient is defined by the ratio formed by the covariance divided by the product of standard deviations of $g_{\Delta,i}$ and $g_{\Psi,i}$.

A correlation matrix, as defined by eq (81), is calculated by subroutine CORLAT. Being a symmetric matrix, the upper triangle is reported to the output file, X.OUT. The output file also reports the condition number of this matrix [17], as well as the diagonal elements

of the renormalization matrix as defined by eq (63). Such helps identify which model parameters may be correlated and aids the decision process regarding the selection of frozen model parameters during a series of calculations. Upon completing these tasks, the program stops.

4.3.3 Search Grid (vary)

Upon selection of option three (level one), the main program calls subroutine SCAN2 to invert the ellipsometric equations. It seeks to find a minimum to the *error* expression, as a constrained optimization problem, by initiating a scan over a grid of numerical values associated with a selected set of model parameters. Here, the (froz/vary) switch specifies the selection; the model parameters contributing to the grid are those whose numerical values are selected to undergo variation. Since the grid is constructed from a single block of nested DO-loops, each selected model parameter contributes one dimension to the grid, i.e., a hyper-cube. Note that efficient use of the grid occurs when the selected model parameters are common to all samples.

Again, each dimension of the grid involves four items: the model parameter, the lower bound, the upper bound, and the stepsize. To provide this information to the program, it is convenient to follow the convention of DO-loop specification and request/require that the input format to be of the form:

$$i_p,\ p_1,\ p_2,\ p_3,$$

where i_p refers to the index label associated with the model parameter p, and p_j refers to the numerical value of the model parameter associated with the initial value, final value, and stepsize. Also, it is required that the lines of data be ordered according to the input data requirements presented earlier in section 4.2, i.e., optical media first, then thicknesses.

For each point on the multidimensional grid, the calculation initiates a series of unconstrained optimization problems, except that the range of numerical values of the selected model parameters is restricted within the bounds of the grid. At the conclusion of the grid scan, when the program reports its *best* fixed-point solution, the program also reports any components of the solution which lie near the boundary of the selected grid. Solutions with components at grid boundaries are, of course, grid dependent, and as such, further calculation is usually necessary, i.e., after the grid is moved or redefined.

Since grid specification includes the stepsize, i.e., number of steps, it is possible for calculations to become very time-consuming. For this reason, the routine provides breakpoint information to the output file X.SOUT at intervals of 15 cpu-minutes. This is discussed in section 4.1.1. The mechanism for restarting a previously interrupted calculation is rather straightforward. One need merely append the contents of X.SOUT onto the end of the input data file X.DAT, *without* any intervening blank lines. Upon starting any grid scan calculation, the program will attempt reading a set of breakpoint information. If restarting is not intended by the user, the input data file should be absent of excess lines. The program does initiate some measure of fail-testing regarding the breakpoint information. Anything deemed irregular in the input file should inhibit the chances of erroneous restarts, but it is good practice to truncate the input data file with either an end-of-file condition or some other delimiter, e.g., a connected line of four equal signs.

Again, as presented in the previous subsection, i.e., section 4.3.2, the routine reports only a basic set of statistics regarding deviations of the fit between the measurement data and that of the model.

4.3.4 Search Grid (froz, vary)

Option four (level one) is one example of a specialty algorithm. It is tailored to a specific need of the user regarding the analysis of a particular class of problems and, as such, may find limited applicability with general problems, apart from providing an outline for the user to modify the program to address one's own particular needs.

Here, the program calls subroutine SCAN3 to invert the ellipsometric equations by constructing two nested blocks of distinct grid scans, where the inner-block grid assumes that the samples are uncoupled. The first/inner block of grids is similar to that mentioned earlier in section 4.3.3, the so-called *vary* grid. Each sample is scanned independently of other samples, so the coupling between samples is suppressed, while coupling within the sample is fully accounted. At each point of the grid and sample, the routine initiates a calculation of the constrained optimization problem.

The outer block of grids involves a selected subset of model parameters whose numerical values are considered frozen. For each point of the so-called *froz* grid, the program scans the appropriate *vary* grid of each sample, and the results of calculation are reported to the output file, X.OUT. Consequently, this can lead to substantial listings of output. Note,

the coupling between samples involving model parameters contributing to the *froz* grid is fully accounted.

Breakpointing capability is provided with the routine. Upon completion of the above tasks, a basic set of statistics regarding deviations of the fit between measurement and the model is provided. If coupling exists between samples regarding the *vary* grid of model parameters, the simple statistics that are generated at conclusion of calculation will generally be meaningless, because the samples were treated independently; i.e., distinct samples maintain distinct values associated with the same model parameter.

The capability provided by this option has been found toward generating surfaces of the *error* expression for purposes of later graphics. Note, it is a simple matter to dupe the program and induce an uncoupling among the samples during analysis by selecting one model parameter to compose the *froz* grid and set its lower/upper bound to the same numerical value.

To specify which model parameters contribute to the *froz* grid, one need merely set the (froz/vary) switch to an integer value of two and include the specification of its grid within the ordering of the input file as discussed earlier in section 4.3.3.

4.3.5 Sensitivity Analysis

Option five (level one) induces the program to initiate a sensitivity analysis regarding the model parameters whose numerical values were selected to undergo variation, as discussed in section 3.2. This calculation is performed by the subroutine SEAMA.

The output includes the following, where:

$\langle gg \rangle$, refers to the statistical estimate of the variance of the deviations involving both Δ and ψ as defined by eq (85). The standard deviation is included for convenience.

$\langle aa \rangle$, refers to the statistical estimate of the variance of the uncertainty in the angle of incidence associated with laboratory instrumentation during measurement, as defined by eq (86). The standard deviation is included as well.

$|BggB|_{jk}$, refers to covariance matrix elements involving the deviations as defined by the first term in eq (91).

$|BJaaJB|_{jk}$, refers to covariance matrix elements involving the uncertainty in the angles of incidence as defined by the second term in eq (91).

The so-called *total* uncertainty reported for the *vary* model parameter is defined by eq (98), which combines contributions from random and systematic errors. Here, the random component involves one of the diagonal elements from the covariance matrix. Also listed are the relative contributions of systematic errors entering into this sum.

4.3.6 MAE Plots of Uncertainties

Option six (level one) is another example of a specialty algorithm. This option serves the purpose of finding the *optimum* set of multiple angles of incidence which may induce the minimum magnitude of uncertainty associated with some selected model parameter. To accomplish this, the routine scans a grid of incident angles and calculates the magnitude of uncertainty associated with the appropriate model parameter. The formulation of the problem regarding uncertainties is that as reported in the literature [24], which is distinct from that presented in section 3.2. The algorithm combines contributions utilizing absolute values, and so it overestimates the calculated magnitudes. Further, as may be expected, any effects due to correlation here will naturally frustrate the exercise.

This option is delegated to the subroutine SEAMAX. It calls subroutine SEAM2 to construct a table involving the forward problem, i.e., providing predetermined magnitudes to the deviations g, while properly calculating/storing the elements of the Jacobian, for the entire grid of incident angles. As the multidimensional grid is scanned across this collection of multiple angles of incidence, subroutine SEAM3 constructs the forward problem for the appropriate set of incident angles. The associated magnitudes of uncertainty $|v_j|$ are calculated and written to the output file, X.SOUT. At conclusion of the scan, subroutine POPLAT is used to plot the data. Here, since the calculated magnitudes were found to vary by several orders of magnitude, it was convenient to request subroutine POPLAT to induce a logarithmic transformation onto the magnitudes, where magnitudes are mapped into the range from –5 to 4. For the case involving more that two multiple angles of incidence, the routine graphs the so-called *Density of States* profile associated with the magnitudes. The domain of the graph involves the angle of incidence.

The necessary input is read by subroutine INPDAT. No repeat data are allowed; i.e., repeat values are set equal to one. No measurement data are necessary in the input file, X.DAT. The first line of input, associated with this option, is the integer one. The input format for the next following lines is of the form:

$$h_{\phi,\hat{r}\ell ij},\ m_{\phi,\hat{r}\ell ij},\ \tilde{a}_{\ell,ij},\ \tilde{\lambda}_{ij},\ j$$

where $h_{\phi,\hat{r}\ell ij}$ refers to the integer increment or step of the DO-loop associated with the construction of the grid of incident angles, involving the $\hat{r}^{\text{th}}$ repeat, the ℓ^{th} ambient, and the i^{th} wavelength on the j^{th} sample. The unit step is measured in degrees, where a reasonable integer value is something like 2 degrees. The other symbols are the same as that referred to in section 4.2.3. Note that $(1 \leq m_{\phi,\hat{r}\ell ij} \leq \texttt{nanglm})$ as was mentioned earlier in section 4.1.2.

Again, subroutine SEAM2 sets up a tabulation, and subroutine SEAM3 uses it. This design removes redundant calculations of the forward problem, i.e., to hasten execution time, but at the expense of array space. The tabulation requires the use of large arrays, and the algorithms within SCANCC and SEAMX2 tend to overestimate the sizes necessary for said arrays. For these reasons and that due to limited applications of this option, it has been suppressed from the main program.

5. Worked Examples

The following subsections present worked examples of using the program. Each subsection presents one example involving one selection from among the set of available top level options. A brief orientation is given regarding the purpose of each calculation. Since the program outputs a journal listing of the input data, only the output file needs to be presented in the following subsections. Again, the output lines are shown as indexed for convenience. As mentioned earlier in section 4.1.1, this output file is named X.OUT.

5.1 Search (vary)

This option is discussed in section 4.3.2. The input data file X.DAT is similar to that presented in section 4.2.4, i.e., apart from some minor modifications as discussed below. This option is selected by setting the first line of data in the input file X.DAT to be the integer value '2.' This is shown on line 9 of the output file listing below. Regarding the output on line 31, the model parameter whose numerical value is selected to undergo variation is that associated with thickness of the layer of oxide atop the substrate. The numerical value of the model parameter is set to 110 nm, i.e., the initial value solution as entered in X.DAT.

Recall from section 4.2.3 that the measurement data are *exact* for the model parameters with numerical values as presented in section 4.2.4. Note that on line 19 of the example presented in section 4.2.4, the oxide thickness is 100 nm, i.e., the true solution. Hence, the initial value solution is displaced from the true solution by 10 nm. The purpose here is to show the rate of convergence of the program. The output below written on line 141 shows that 54 iterations were used to converge to the correct final solution shown on line 144. Attention is directed less toward optimality of iterations than with that necessary to follow convergence.

Again, the output file contains a journal listing of the input data, progress reports regarding convergence, the final numerical values of the *vary* model parameters, and a selection of associated statistics, as well as the amount of cpu-time that is used during calculation. The output file X.OUT is given below.

```
1  options:   1,  forward problems, plots, ...
2             2,  search       (     vary)
3             3,  search grid  (     vary)
```

```
4             4,  search grid  (froz,vary)
5             5,  sensitivity analysis
6             6,  MAE plots ' uncertainty
7
8  Enter:   option '
9           option ' 2
10
11 mwaves =   2, number of distinct wavelengths
12            1    632.8000
13            2    441.6000
14
15 mbient =   3, number of distinct ambient environments and waves.
16            1    1.000000
17            2    1.000270
18            3    1.000270
19
20 mlmnts =   7, number of distinct:   n+ik
21            1      3.8820      0.0020    0
22            2      0.0190      0.0020    0
23            3      1.4570      0.0020    0
24            4      0.0000      0.0000    0
25            5      4.7530      0.0020    0
26            6      0.1630      0.0020    0
27            7      1.4660      0.0020    0
28
29 mfilmm =   2, number of distinct film widths
30            1     50.0000      2.0000    0
31            2    110.0000      2.0000    1
32
33 msampl =   2, number of distinct samples
34            1   2   2          ' sample, mfilm, mwave
35            1   2              ' iwave, mbien
36            1   1   2   1      ' i,n,k,z
37            2   3   4   2      ' i,n,k,z
38            3   1   2          ' i,n,k
39            1   1              ' imbien, mrpeat
40            2   2              ' imbien, mrpeat
41            2   1              ' iwave, mbien
42            1   5   6   1      ' i,n,k,z
43            2   7   4   2      ' i,n,k,z
44            3   5   6          ' i,n,k
45            3   1              ' imbien, mrpeat
46
47            2   1   2          ' sample, mfilm, mwave
48            1   1              ' iwave, mbien
49            1   3   4   2      ' i,n,k,z
50            2   1   2          ' i,n,k
51            2   1              ' imbien, mrpeat
52            2   1              ' iwave, mbien
53            1   7   4   2      ' i,n,k,z
54            2   5   6          ' i,n,k
55            3   1              ' imbien, mrpeat
56
```

```
    2   1   1   1   1 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
    1      65.0000       0.0100     173.5043       0.0500      37.0579       0.0500
    2      70.0000       0.0100     172.2664       0.0500      34.3707       0.0500

    3   1   2   1
    1      65.0000       0.0100     173.5101       0.0500      37.0527       0.0500
    2      70.0000       0.0100     172.2750       0.0500      34.3633       0.0500
    3      75.0000       0.0100     170.0354       0.0500      30.1573       0.0500

    2   2   2   1
    1      67.5000       0.0100     172.9647       0.0500      35.8405       0.0500
    2      72.5000       0.0100     171.3529       0.0500      32.5214       0.0500

    4   1   3   2
    1      67.5000       0.0100     286.8467       0.0500      30.2644       0.0500
    2      70.0000       0.0100     297.5219       0.0500      31.2799       0.0500
    3      72.5000       0.0100     307.5732       0.0500      32.6322       0.0000
    4      75.0000       0.0100     316.8687       0.0500      34.2225       0.0000

    2   1   2   1   2 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
    1      70.0000       0.0100      79.8092       0.0500      41.0455       0.0500
    2      73.0000       0.0100      68.2502       0.0500      41.0304       0.0500

    2   1   3   2
    1      70.0000       0.0100     211.2235       0.0500      77.3289       0.0500
    2      73.0000       0.0100     216.1556       0.0500      84.4928       0.0500


zoom:   loop,    ratio of reduction,      |g|
                   (rel)       (total)
              0                           1.487E+01    (degrees)
              1   9.034E-01   9.034E-01   1.344E+01
              2   8.729E-01   7.886E-01   1.173E+01
              3   8.138E-01   6.418E-01   9.546E+00
              4   7.986E-01   5.125E-01   7.623E+00
              5   7.925E-01   4.061E-01   6.041E+00
              6   7.893E-01   3.206E-01   4.768E+00
              7   7.907E-01   2.535E-01   3.770E+00
              8   7.918E-01   2.007E-01   2.985E+00
              9   7.937E-01   1.593E-01   2.369E+00
             10   7.953E-01   1.267E-01   1.884E+00
             11   7.961E-01   1.008E-01   1.500E+00
             12   7.971E-01   8.038E-02   1.196E+00
             13   7.976E-01   6.411E-02   9.536E-01
             14   7.982E-01   5.117E-02   7.612E-01
             15   7.986E-01   4.087E-02   6.079E-01
             16   7.989E-01   3.265E-02   4.856E-01
             17   7.991E-01   2.609E-02   3.880E-01
             18   7.994E-01   2.085E-02   3.102E-01
             19   7.995E-01   1.667E-02   2.480E-01
             20   7.995E-01   1.333E-02   1.983E-01
             21   7.995E-01   1.066E-02   1.585E-01
             22   7.999E-01   8.525E-03   1.268E-01
```

```
            23   7.999E-01   6.819E-03   1.014E-01
            24   8.001E-01   5.456E-03   8.115E-02
            25   7.995E-01   4.362E-03   6.489E-02
            26   7.998E-01   3.489E-03   5.190E-02
            27   7.999E-01   2.791E-03   4.151E-02
            28   8.000E-01   2.233E-03   3.321E-02
            29   7.995E-01   1.785E-03   2.655E-02
            30   7.995E-01   1.427E-03   2.123E-02
            31   7.997E-01   1.141E-03   1.697E-02
            32   8.002E-01   9.131E-04   1.358E-02
            33   7.991E-01   7.297E-04   1.085E-02
            34   7.974E-01   5.819E-04   8.655E-03
            35   8.029E-01   4.672E-04   6.949E-03
            36   8.034E-01   3.753E-04   5.583E-03
            37   7.993E-01   3.000E-04   4.462E-03
            38   7.994E-01   2.398E-04   3.567E-03
            39   7.888E-01   1.892E-04   2.814E-03
            40   8.146E-01   1.541E-04   2.292E-03
            41   7.874E-01   1.213E-04   1.805E-03
            42   8.000E-01   9.706E-05   1.444E-03
            43   8.191E-01   7.950E-05   1.182E-03
            44   7.951E-01   6.320E-05   9.401E-04
            45   8.042E-01   5.083E-05   7.561E-04
            46   8.152E-01   4.143E-05   6.163E-04
            47   8.072E-01   3.344E-05   4.975E-04
            48   7.208E-01   2.411E-05   3.586E-04
            49   8.994E-01   2.168E-05   3.225E-04
            50   7.821E-01   1.696E-05   2.522E-04
            51   8.458E-01   1.434E-05   2.133E-04
            52   8.463E-01   1.214E-05   1.805E-04
            53   8.298E-01   1.007E-05   1.498E-04
            54   7.243E-01   7.294E-06   1.085E-04

model parameter value along the minimum:
   1)       100.0000      0.00000  for:   2,  (z   ), estimated uncertainty

initial |g| =  1.48745E+01    (degrees)
  final |g| =  1.08499E-04

-------------------------------------------------------------
Statistics of deviations ' experiment-model ' g

   where:    g ' column array of length ' 2M
     let:   () ' (psi or delta) ' (1 or 2)

        mean () = m() = <g()> = (1/M) sum: g()
    variance () = < [g( )-m( )]**2 >
 covariance     = < [g(1)-m(1)]*[g(2)-m(2)] >
       std dev = sqrt (variance)
 correlat coef = covariance / [std dev (psi) * std dev (delta)]

                     mean,     std dev     (degrees)
      psi:          0.000        0.000
```

```
     delta:          0.000        0.000
                     0.595 * correlation coefficient * <psi|delta>

J(T)*J:         (renormalized for correlation)
    1)    1.00000

Normalization coefficients:     sqrt [J(T)*J](i,i)
  8.35E-02

rcond=   1.000E+00,  condition number

elapsed cpu-time =  38  centi-seconds
                 +  16  seconds
```

5.2 Search Grid (vary)

This option is discussed in section 4.3.3. The option is selected by entering an integer value of '3' on the first line in the input data file X.DAT. This is indicated on line 9 of the output file listing below. Again, only the oxide layer thickness is subjected to variation. Line 87 shows the necessary parameters governing the formation of the grid. Note that the grid values overstep the correct solution. For each grid point, the program initiates a series of iterations similar to that shown in the previous subsection. The solution is presented on line 96; it is the correct solution. The contents of the output file X.OUT is given below.

```
 1  options:   1,  forward problems, plots, ...
 2             2,  search       (     vary)
 3             3,  search grid  (     vary)
 4             4,  search grid  (froz,vary)
 5             5,  sensitivity analysis
 6             6,  MAE plots ' uncertainty
 7
 8  Enter:   option '
 9           option ' 3
10
11  mwaves =   2, number of distinct wavelengths
12             1    632.8000
13             2    441.6000
14
15  mbient =   3, number of distinct ambient environments and waves.
16             1    1.000000
17             2    1.000270
18             3    1.000270
19
20  mlmnts =   7, number of distinct:   n+ik
21             1      3.8820      0.0020    0
22             2      0.0190      0.0020    0
23             3      1.4570      0.0020    0
24             4      0.0000      0.0000    0
25             5      4.7530      0.0020    0
26             6      0.1630      0.0020    0
27             7      1.4660      0.0020    0
28
29  mfilmm =   2, number of distinct film widths
30             1     50.0000      2.0000    0
31             2    100.0000      2.0000    1
32
33  msampl =   2, number of distinct samples
34             1   2   2          ' sample, mfilm, mwave
35             1   2              ' iwave, mbien
36             1   1   2   1      ' i,n,k,z
37             2   3   4   2      ' i,n,k,z
```

```
38              3   1   2          ' i,n,k
39              1   1              ' imbien, mrpeat
40              2   2              ' imbien, mrpeat
41              2   1              ' iwave, mbien
42              1   5   6   1      ' i,n,k,z
43              2   7   4   2      ' i,n,k,z
44              3   5   6          ' i,n,k
45              3   1              ' imbien, mrpeat
46
47              2   1   2          ' sample, mfilm, mwave
48              1   1              ' iwave, mbien
49              1   3   4   2      ' i,n,k,z
50              2   1   2          ' i,n,k
51              2   1              ' imbien, mrpeat
52              2   1              ' iwave, mbien
53              1   7   4   2      ' i,n,k,z
54              2   5   6          ' i,n,k
55              3   1              ' imbien, mrpeat
56
57    2   1   1   1   1 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
58    1     65.0000       0.0100     173.5043       0.0500     37.0579       0.0500
59    2     70.0000       0.0100     172.2664       0.0500     34.3707       0.0500
60
61    3   1   2   1
62    1     65.0000       0.0100     173.5101       0.0500     37.0527       0.0500
63    2     70.0000       0.0100     172.2750       0.0500     34.3633       0.0500
64    3     75.0000       0.0100     170.0354       0.0500     30.1573       0.0500
65
66    2   2   2   1
67    1     67.5000       0.0100     172.9647       0.0500     35.8405       0.0500
68    2     72.5000       0.0100     171.3529       0.0500     32.5214       0.0500
69
70    4   1   3   2
71    1     67.5000       0.0100     286.8467       0.0500     30.2644       0.0500
72    2     70.0000       0.0100     297.5219       0.0500     31.2799       0.0500
73    3     72.5000       0.0100     307.5732       0.0500     32.6322       0.0000
74    4     75.0000       0.0100     316.8687       0.0500     34.2225       0.0000
75
76    2   1   2   1   2 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
77    1     70.0000       0.0100      79.8092       0.0500     41.0455       0.0500
78    2     73.0000       0.0100      68.2502       0.0500     41.0304       0.0500
79
80    2   1   3   2
81    1     70.0000       0.0100     211.2235       0.0500     77.3289       0.0500
82    2     73.0000       0.0100     216.1556       0.0500     84.4928       0.0500
83
84
85  Scan a grid of model parameters.
86  Grid info:     initial,       final,      increment
87       1)        95.0000      105.0000       10.0000    ' for:    2,  (z    )
88
89  Note:    NO attempt was made to restart.
90
```

```
 91  number of grid points scanned, kt  =          2
 92   population along the minimum, kts =          1
 93               norm of residual, |g| =   5.30715E-05   (degrees)
 94
 95  Model parameter value along the minimum:
 96     1)   1.000000E+02   ' for:    2,  (z   )
 97
 98  ------------------------------------------------------------
 99  Statistics of deviations ' experiment-model ' g
100
101     where:    g ' column array of length ' 2M
102       let:   () ' (psi or delta) ' (1 or 2)
103
104         mean () = m() = <g()> = (1/M) sum: g()
105     variance () = < [g( )-m( )]**2 >
106   covariance    = < [g(1)-m(1)]*[g(2)-m(2)] >
107         std dev = sqrt (variance)
108   correlat coef = covariance / [std dev (psi) * std dev (delta)]
109
110                       mean,     std dev     (degrees)
111        psi:          0.000        0.000
112      delta:          0.000        0.000
113                      0.274 ' correlation coefficient ' <psi|delta>
114
115  J(T)*J:        (renormalized for correlation)
116     1)    1.00000
117
118  Normalization coefficients:     sqrt [J(T)*J](i,i)
119    8.35E-02
120
121  rcond=   1.000E+00,  condition number
122
123  elapsed cpu-time =  15  centi-seconds
124                   +  32  seconds
```

5.3 Search Grid (froz,vary)

This option is discussed in section 4.3.4. To select this option, the first item entered on the first line of the input data file X.DAT ought to be the integer value '4.' This is indicated on line 9 in the listing presented below. Again, the thickness of the oxide layer is subjected to variation, a *vary* model parameter, i.e., line 31. Regarding line 24, it is seen that the integer (froz/vary) switch has been set to two, i.e., a grid involving a frozen model parameter. This model parameter is the extinction coefficient of the oxide layer. The associated grid parameters are shown on lines 87 and 88. Here, for convenience, the oxide extinction coefficient is set to consider only a single value, i.e., zero. The correct solution is presented on line 131. The output file is listed below.

```
 1  options:   1,  forward problems, plots, ...
 2             2,  search       (     vary)
 3             3,  search grid  (     vary)
 4             4,  search grid  (froz,vary)
 5             5,  sensitivity analysis
 6             6,  MAE plots ' uncertainty
 7
 8  Enter:   option '
 9           option ' 4
10
11  mwaves =   2, number of distinct wavelengths
12             1    632.8000
13             2    441.6000
14
15  mbient =   3, number of distinct ambient environments and waves.
16             1    1.000000
17             2    1.000270
18             3    1.000270
19
20  mlmnts =   7, number of distinct:   n+ik
21             1      3.8820      0.0020    0
22             2      0.0190      0.0020    0
23             3      1.4570      0.0020    0
24             4      0.0000      0.0000    2
25             5      4.7530      0.0020    0
26             6      0.1630      0.0020    0
27             7      1.4660      0.0020    0
28
29  mfilmm =   2, number of distinct film widths
30             1     50.0000      2.0000    0
31             2    100.0000      2.0000    1
32
33  msampl =   2, number of distinct samples
34             1   2   2         ' sample, mfilm, mwave
35             1   2             ' iwave, mbien
```

```
36              1   1   2   1      ' i,n,k,z
37              2   3   4   2      ' i,n,k,z
38              3   1   2          ' i,n,k
39              1   1              ' imbien, mrpeat
40              2   2              ' imbien, mrpeat
41              2   1              ' iwave, mbien
42              1   5   6   1      ' i,n,k,z
43              2   7   4   2      ' i,n,k,z
44              3   5   6          ' i,n,k
45              3   1              ' imbien, mrpeat
46
47              2   1   2          ' sample, mfilm, mwave
48              1   1              ' iwave, mbien
49              1   3   4   2      ' i,n,k,z
50              2   1   2          ' i,n,k
51              2   1              ' imbien, mrpeat
52              2   1              ' iwave, mbien
53              1   7   4   2      ' i,n,k,z
54              2   5   6          ' i,n,k
55              3   1              ' imbien, mrpeat
56
57    2   1   1   1   1  1 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
58    1     65.0000       0.0100     173.5043       0.0500     37.0579       0.0500
59    2     70.0000       0.0100     172.2664       0.0500     34.3707       0.0500
60
61    3   1   2   1
62    1     65.0000       0.0100     173.5101       0.0500     37.0527       0.0500
63    2     70.0000       0.0100     172.2750       0.0500     34.3633       0.0500
64    3     75.0000       0.0100     170.0354       0.0500     30.1573       0.0500
65
66    2   2   2   1
67    1     67.5000       0.0100     172.9647       0.0500     35.8405       0.0500
68    2     72.5000       0.0100     171.3529       0.0500     32.5214       0.0500
69
70    4   1   3   2
71    1     67.5000       0.0100     286.8467       0.0500     30.2644       0.0500
72    2     70.0000       0.0100     297.5219       0.0500     31.2799       0.0500
73    3     72.5000       0.0100     307.5732       0.0500     32.6322       0.0000
74    4     75.0000       0.0100     316.8687       0.0500     34.2225       0.0000
75
76    2   1   2   1   2 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
77    1     70.0000       0.0100      79.8092       0.0500     41.0455       0.0500
78    2     73.0000       0.0100      68.2502       0.0500     41.0304       0.0500
79
80    2   1   3   2
81    1     70.0000       0.0100     211.2235       0.0500     77.3289       0.0500
82    2     73.0000       0.0100     216.1556       0.0500     84.4928       0.0500
83
84
85  Scan a grid of model parameters.
86  Grid info:     initial,       final,      increment
87     1)           0.0000         0.0000        0.0000    ' for:    4,  (n+ik)
88     2)          95.0000       105.0000       10.0000    ' for:    2,  (z   )
```

```
 89
 90 Range of do-loops:   froz
 91    1)      1   grid points for:     4,  (n+ik)
 92
 93 Range of do-loops:   vary
 94    1)      2   grid points for:     2,  (z   )
 95
 96 Couplings between samples involve:   1   distinct model parameters
 97            2,   (z   )
 98
 99 Note:    The existence of coupling between samples
100          due to the  VARY  model parameters --
101 induces an unusual interpretation unto the residual  |g|,
102 because, while providing coupling,  they are NOT
103 necessarily of similar value on different samples.
104
105
106 Attempt to read breakpoint information
107
108 Note:   NO attempt was made to restart.
109
110 ======================================================================
111    #      1                   ' grid:   froz
112    1)     1    0.000000E+00   ' model parameter for:    4,  (n+ik)
113     ----------------------------------------------------------------
114         sample                ' grid:   vary
115            1    3.947890E-05   ' |g|
116                 1.000000E+02   ' for:    2,  (z   )
117            2    1.996443E-04   ' |g|
118                 1.000000E+02   ' for:    2,  (z   )
119                 1.084976E-04   ' |g| ' summed over samples
120
121 ======================================================================
122
123 number of FROZ grid points scanned, ktu  =          1
124       population along the minimum, ktum =          1
125                   norm of residual, |g|  =  1.084994E-04   (degrees)
126
127 Model parameter value along the minimum:    froz
128                 0.000000E+00   ' for:    4,  (n+ik)  ' boundary
129
130 Model parameter value along the minimum:    vary
131                 1.000000E+02   ' for:    2,  (z   )
132
133 Note that the minimum point is near a boundary.
134
135 Note:    The existence of coupling between samples
136          due to the  VARY  model parameters --
137 induces an unusual interpretation unto the residual  |g|,
138 because, while providing coupling,  they are NOT
139 necessarily of similar value on different samples.
140
141
```

```
142  ------------------------------------------------------------
143  Statistics of deviations ' experiment-model ' g
144
145     where:     g ' column array of length ' 2M
146       let:    () ' (psi or delta) ' (1 or 2)
147
148         mean () = m() = <g()> = (1/M) sum: g()
149     variance () = < [g( )-m( )]**2 >
150   covariance     = < [g(1)-m(1)]*[g(2)-m(2)] >
151         std dev = sqrt (variance)
152   correlat coef = covariance / [std dev (psi) * std dev (delta)]
153
154                        mean,      std dev      (degrees)
155        psi:          0.000         0.000
156      delta:          0.000         0.000
157                      0.595 ' correlation coefficient ' <psi|delta>
158
159  J(T)*J:         (renormalized for correlation)
160     1)    1.00000
161
162  Normalization coefficients:      sqrt [J(T)*J](i,i)
163    8.35E-02
164
165  rcond=    1.000E+00,  condition number
166
167  elapsed cpu-time =  29  centi-seconds
168                   +  42  seconds
```

5.4 Sensitivity Analysis

This option is discussed in section 4.3.5. To select this option, the first item on the first line in the input data file X.DAT ought to contain the integer value '5.' Regarding the output presented below, lines 30 and 31 reveal that both layer thicknesses are the only *vary* model parameters. Note that no iterations are performed by the program; it is assumed that the critical-point or fixed-point solution has already been found, and is given by those entered upon input. The total uncertainty associated with knowing the thicknesses of the the silicon and oxide layers is given, respectively, by lines 125 and 142, i.e., `4.69E-02` and `3.03E-01`. The output file is given by the following.

```
 1  options:   1,  forward problems, plots, ...
 2             2,  search       (     vary)
 3             3,  search grid  (     vary)
 4             4,  search grid  (froz,vary)
 5             5,  sensitivity analysis
 6             6,  MAE plots ' uncertainty
 7
 8  Enter:   option '
 9           option ' 5
10
11  mwaves =   2, number of distinct wavelengths
12             1    632.8000
13             2    441.6000
14
15  mbient =   3, number of distinct ambient environments and waves.
16             1    1.000000
17             2    1.000270
18             3    1.000270
19
20  mlmnts =   7, number of distinct:   n+ik
21             1      3.8820      0.0020    0
22             2      0.0190      0.0020    0
23             3      1.4570      0.0020    0
24             4      0.0000      0.0000    0
25             5      4.7530      0.0020    0
26             6      0.1630      0.0020    0
27             7      1.4660      0.0020    0
28
29  mfilmm =   2, number of distinct film widths
30             1     50.0000      2.0000    1
31             2    100.0000      2.0000    1
32
33  msampl =   2, number of distinct samples
34             1   2   2          ' sample, mfilm, mwave
35             1   2              ' iwave, mbien
36             1   1   2   1      ' i,n,k,z
37             2   3   4   2      ' i,n,k,z
```

```
38            3   1   2          ' i,n,k
39            1   1              ' imbien, mrpeat
40            2   2              ' imbien, mrpeat
41            2   1              ' iwave, mbien
42            1   5   6   1      ' i,n,k,z
43            2   7   4   2      ' i,n,k,z
44            3   5   6          ' i,n,k
45            3   1              ' imbien, mrpeat
46
47            2   1   2          ' sample, mfilm, mwave
48            1   1              ' iwave, mbien
49            1   3   4   2      ' i,n,k,z
50            2   1   2          ' i,n,k
51            2   1              ' imbien, mrpeat
52            2   1              ' iwave, mbien
53            1   7   4   2      ' i,n,k,z
54            2   5   6          ' i,n,k
55            3   1              ' imbien, mrpeat
56
57    2   1   1   1   1 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
58    1     65.0000       0.0100     173.5043       0.0500     37.0579       0.0500
59    2     70.0000       0.0100     172.2664       0.0500     34.3707       0.0500
60
61    3   1   2   1
62    1     65.0000       0.0100     173.5101       0.0500     37.0527       0.0500
63    2     70.0000       0.0100     172.2750       0.0500     34.3633       0.0500
64    3     75.0000       0.0100     170.0354       0.0500     30.1573       0.0500
65
66    2   2   2   1
67    1     67.5000       0.0100     172.9647       0.0500     35.8405       0.0500
68    2     72.5000       0.0100     171.3529       0.0500     32.5214       0.0500
69
70    4   1   3   2
71    1     67.5000       0.0100     286.8467       0.0500     30.2644       0.0500
72    2     70.0000       0.0100     297.5219       0.0500     31.2799       0.0500
73    3     72.5000       0.0100     307.5732       0.0500     32.6322       0.0000
74    4     75.0000       0.0100     316.8687       0.0500     34.2225       0.0000
75
76    2   1   2   1   2 ' mangl, repeat,ambient,wave,sample/ (phi,delta,psi)
77    1     70.0000       0.0100      79.8092       0.0500     41.0455       0.0500
78    2     73.0000       0.0100      68.2502       0.0500     41.0304       0.0500
79
80    2   1   3   2
81    1     70.0000       0.0100     211.2235       0.0500     77.3289       0.0500
82    2     73.0000       0.0100     216.1556       0.0500     84.4928       0.0500
83
84
85  ------------------------------------------------------------
86  Statistics of deviations ' experiment-model ' g
87
88     where:    g ' column array of length ' 2M
89       let:   () ' (psi or delta) ' (1 or 2)
90
```

```
        mean () = m() = <g()> = (1/M) sum: g()
    variance () = < [g( )-m( )]**2 >
 covariance     = < [g(1)-m(1)]*[g(2)-m(2)] >
       std dev = sqrt (variance)
 correlat coef = covariance / [std dev (psi) * std dev (delta)]

                    mean,     std dev     (degrees)
      psi:         0.000        0.000
    delta:         0.000        0.000
                   0.017 ' correlation coefficient ' <psi|delta>

J(T)*J:        (renormalized for correlation)
   1)    1.00000
   2)    0.08240   1.00000

Normalization coefficients:     sqrt [J(T)*J](i,i)
  7.85E-02  8.35E-02

rcond=   8.477E-01,  condition number
=========================================================================
<gg> '    3.414E-13   5.843E-07    (variance, std dev)
<aa> '    3.046E-08   1.745E-04    (variance, std dev)
-------------------------------------------------------------------------
Discern:    Uncertainty      in model parameters
  where:    |v| = sqrt( BB(T) <gg> + (BJ)(BJ)(T) <aa>)
                                  +  |BJ||u|,

            B ' [J(T)*J]**-1 *J(T),      (nonsquare J)

   vary      total     initial    parameter
-------------------------------------------------------------------------
             1    1   1.86E-12   2.36E-06     (j,i), |BggB|,  |BJaaJB|
                      1.36E-06   1.54E-03            |Bg|,    |BJa|
             - - - - - - - - - - - - - - - - - - - - - - - - - - -
     1)     4.69E-02   2.00E+00   5.000000E+01, for:   1, (z   )
            1.54E-03   1.36E-06   1.54E-03    random, |Bg|, |BJa|
                       4.54E-02    systematic ' total

                  1)   2.29E-03    systematic ' |BJu| '   1, (n+ik)
                  2)   9.22E-04    systematic ' |BJu| '   2, (n+ik)
                  3)   2.54E-03    systematic ' |BJu| '   3, (n+ik)
                  4)   0.00E+00    systematic ' |BJu| '   4, (n+ik)
                  5)   2.16E-02    systematic ' |BJu| '   5, (n+ik)
                  6)   8.66E-03    systematic ' |BJu| '   6, (n+ik)
                  7)   9.40E-03    systematic ' |BJu| '   7, (n+ik)
-------------------------------------------------------------------------
             2    1  -1.44E-13  -2.68E-07     (j,i), |BggB|,  |BJaaJB|
                      3.80E-07   5.18E-04            |Bg|,    |BJa|
             2    2   1.64E-12   3.03E-06     (j,i), |BggB|,  |BJaaJB|
                      1.28E-06   1.74E-03            |Bg|,    |BJa|
             - - - - - - - - - - - - - - - - - - - - - - - - - - -
     2)     3.03E-01   2.00E+00   1.000000E+02, for:   2, (z   )
            1.74E-03   1.28E-06   1.74E-03    random, |Bg|, |BJa|
```

```
144                        3.01E-01    systematic ' total
145
146                  1)    1.03E-04    systematic ' |BJu| '   1, (n+ik)
147                  2)    2.51E-05    systematic ' |BJu| '   2, (n+ik)
148                  3)    2.45E-04    systematic ' |BJu| '   3, (n+ik)
149                  4)    0.00E+00    systematic ' |BJu| '   4, (n+ik)
150                  5)    5.04E-03    systematic ' |BJu| '   5, (n+ik)
151                  6)    6.31E-03    systematic ' |BJu| '   6, (n+ik)
152                  7)    2.90E-01    systematic ' |BJu| '   7, (n+ik)
153 ================================================================
154
155 elapsed cpu-time =  62  centi-seconds
```

6. Listing of Software Source Files and Routines

This section presents a listing of the source files for MAIN1. For convenience, the files are partitioned into three classes. The first class involves files associated with storage arrays, the second class involves source programs that govern the scattering problem and hosts the user's options, and the third class involves general utilities that provide basic tasks for the source programs of class two.

6.1 Named COMMON and BLOCK DATA Statements

6.1.1 IOUNIT.

```
1  c     Named common of the logical unit assignments associated with
2  c     reading/writing of input/output files on the hardware disk.
3
4        common /iounit / in, iout, idat, isout, iplt
5
```

6.1.2 DEFNIT.

```
1  c     Each sample is characterized by the film/substrate:
2  c       geometry:    air / z(1) / ... / z(nfilms) / substrate
3  c     parameters:    three data (t,n,k) for each film,     may vary.
4  c                    two   data ( ,n,k) for the substrate, may vary.
5  c                    one   data ( ,n, ) for the air,   may not vary.
6
7  c     Special case for:   analyzing Standard Reference Materials
8  c*    parameter (nsampl=150) ! sample configurations * films/substrate
9  c*    parameter (nfilms=  4) ! distinct films on a sample
10 c*    parameter (nlmnts= 16) ! (n,k| w,lmnt) * Si, SiO2, etc.
11 c*    parameter (nwaves=  1) ! wavelengths of incident probe
12 c*    parameter (nbient=  1) ! ambient environment of samples
13 c*    parameter (nrpeat= 20) ! repeats of any given experiment
14 c*    parameter (nanglx=  2) ! angles at one wavelength to fit experiment
15 c*    parameter (nanglm=  2) ! multi-angle error analysis at one wavelength
16
17 c     Usual case:
18       parameter (nsampl=  8) ! sample configurations * films/substrate
19       parameter (nfilms=  8) ! distinct films on a sample
20       parameter (nlmnts=300) ! (n,k| w,lmnt) * Si, SiO2, etc.
21       parameter (nwaves=100) ! wavelengths of incident probe
22       parameter (nbient=  1) ! ambient environment of samples
23       parameter (nrpeat=  1) ! repeats of any given experiment
24       parameter (nanglx= 20) ! angles at one wavelength to fit experiment
25       parameter (nanglm=  2) ! multi-angle error analysis at one wavelength
26
27       parameter (nrows =nfilms*3+2   )          ! (z,n,k)/(n,k)  * 1 sample
28       parameter (nfilmm=nfilms*nsampl)          !  z
29       parameter (nrowss=nfilmm+nlmnts)          ! (z,n,k)        * n samples
30       parameter (mrowss=nsampl*nwaves*nbient*nrpeat*nanglx*2)
31       parameter (nexpts=nsampl*nwaves*nbient*nrpeat*nanglx  )
32       parameter (nseams=nsampl*nwaves*nbient*       nanglm  )
33
34       parameter (nnjaaa = 1000000)              ! convenience
35 c     parameter (nnjaaa = mrowss*nrows)         ! aa,ja, aaa,jaa
36
37       complex    cmplx, conjg, sqrtt
38
39 c     Note:      nanglm < nanglx                ! SEAMAX
```

6.1.3 FILMMM.

```
      real      ambent(nwaves*nbient)
      real      waveln(nwaves), waveqq(nwaves), waveww(nwaves)
      real      diefcn(nlmnts), uncerl(nlmnts)
      real      widths(nfilmm), uncerz(nfilmm)
      integer   lvaryl(nlmnts), lvaryz(nfilmm)
      integer   nnfilm(nsampl), iifilm(nsampl*nwaves*nrows)
      integer   nnwave(nsampl), iiwave(nsampl*nwaves)
      integer   nnbent(nsampl*nwaves), iibent(nsampl*nwaves*nbient)
      integer   nnpeat(nsampl*nwaves*nbient)

      real      psiiis(nexpts), deltas(nexpts), angles(nexpts)
      real      psiiiu(nexpts), deltau(nexpts), angleu(nexpts)
      integer   mangle(nrpeat*nbient*nwaves*nsampl)
      integer   isteps(       nbient*nwaves*nsampl)

      real      aa(nnjaaa), bb(mrowss), cc(mrowss), xx(nrowss)
      integer   ja(nnjaaa), ia(mrowss+1)
      integer   iptu(nrowss), iptv(nrowss), iptw(nrowss)
      logical   llnorm

      common /wwaves / waveln, waveqq, waveww, mwaves

      common /filmmm / ambent, diefcn, widths,
     &                 iifilm, nnfilm, iiwave, nnwave,
     &                 iibent, nnbent, nnpeat,
     &                 mbient, mlmnts, mfilmm, msampl

      common /choose / uncerl, uncerz, lvaryl, lvaryz

      common /exprmt / psiiis, deltas, angles, psiiiu, deltau, angleu,
     &                 mangle, isteps, method

      common /arrays / aa, bb, cc, xx, ja, ia, iptu, iptv, iptw,
     &                 meqns, mvary, mfroz, llnorm
```

6.1.4 FILMSS.

```
1   c     Each sample is characterized by the film/substrate:
2   c       geometry:    air / z(1) / ... / z(nfilms) / substrate
3   c     parameters:    three data (z,n,k) for each film,     may vary.
4   c                    two   data ( ,n,k) for the substrate, may vary.
5   c                    one   data ( ,n, ) for the air,   may not vary.
6
7   c     epsilon = electric permittivity ~ dielectric function ~ 1
8   c     sigma   = specific conductivity                        ~ 0
9   c     mu      = magnetic permeability                        ~ 1
10  c     omega   = angular frequency
11  c     die_r   = epsilon * mu
12  c     die_i   = 4*pi * sigma * mu / omega
13
14        complex  die(nfilms+1)
15        real     zzz(nfilms  ), air
16        integer      mfilm
17
18        common /filmss / die, zzz, air, mfilm
```

6.1.5 RSTACK.

```
1       complex   Rs,          Rp              ! reflection coeff in ambient
2       complex  dRs(nrows),  dRp(nrows)      ! Jacobians
3       complex  dRsa,        dRpa            ! d/d(angle)
4
5       common / rstack / Rs,Rp,  dRs,dRp,  dRsa,dRpa
```

6.1.6 WSTACK.

```
1       parameter (naat=(nrowss*(nrowss+1))/2)  ! upper triangle + diagonal
2       real      aat (naat)                    ! A(T)*A,  Inverse
3       real      aats(nrowss)                  ! A(T)*A,  scale factors
4       integer   ipvt(nrowss)                  ! workspace, LINPACK
5
6       common / wstack / aat, aats, ipvt
```

6.1.7 CGNXX1.

```
1  c     Storage allocation required by:   CGN
2  c     This storage is needed only in:   ZOOM, ZOOM2
3
4        real   p(mrowss), u(mrowss)
5        real   v(nrowss), w(nrowss), xw(nrowss), se(nrowss)
6
7        common / cgnxx1 / p,u,  v,w,xw,se
```

6.1.8 SCANCC.

```
1  c     Used only by:    SCAN2, SCAN3, SCAN2G.
2
3        character  bufft*8, buffd*9      ! convenience, label breakpoint
4
5  c     Nested do-loops associated with multi-parameter grid scan
6        parameter (niii=nrowss)
7        integer    iiii(niii), iii2(niii)
8        real       pppp(niii), ppp1(niii), ppp2(niii), ppp3(niii)
9        real       psav(niii)
10
```

6.1.9 SEAMX1.

```
1  c     Sensitivity analysis for multiple:   angle,ambient,wave,sample.
2
3        integer   iptx(nrows), ipty(nrows)                    ! vary
4        integer   kptx(nrows), kpty(nrows)                    ! frozen
5
6        real      aaa(nnjaaa), bbb(mrowss  ), xxx(nrowss)     ! frozen
7        integer   jaa(nnjaaa), iaa(mrowss+1)
8
9        common / seamx1 / iptx, ipty, kptx, kpty,
10       &                 aaa, bbb, xxx, jaa, iaa
```

6.1.10 SEAMX2.

```
1  c     Sensitivity analysis for multiple:   angle,ambient,wave,sample.
2
3  c     Storage of forward scattering problem at various incident angles.
4  c     For the error analysis,   mrpeat=1,    ... in all cases.
5
6        parameter (ndegr = 89)                          ! grid of angles
7        parameter (ndegs = ndegr*nbient*nwaves*nsampl)  ! (ndegr,)
8        parameter (ndegw = nrows*nbient*nwaves*nsampl)  ! (nrows,)
9        real      psii(ndegs),       dell(ndegs)
10       real      psia(ndegs),       dela(ndegs)       ! dR /d(phi)
11       real      psid(nrows,ndegs), deld(nrows,ndegs) ! dR /d(e) *Jacobian
12
13 c     Arrays associated with the nesting of do-loops within SEAM,
14 c     Sensitivity/error analysis of multiple:
15 c                       angle, ambient, wave, sample.
16
17       integer   iiii(nseams), iii1(nseams), iii2(nseams)
18
19       common / seamx2 / psii, dell, psia, dela, psid, deld,
20      &                  iiii, iii1, iii2, mraws, maraws,
21      &                  mm, raddeg, uang
```

6.1.11 BLKDAT.FOR

```
1         block data  blkdat
2         include  'iounit.'
3
4         data  in,iout,idat,isout,iplt / 5,6,7,8,9 /
5
6         end
```

6.2 Source Programs

6.2.1 MAIN.FOR

```
1       program   main
2       include  'iounit.'
3
4       call fileop                        ! open relevant files
5
6       write (iout,101)
7       read  (idat,*,err=12,end=12)  ichoic
8       write (iout,102)              ichoic
9       if (ichoic.lt.1 .or. ichoic.gt.5) then
10          write (iout,122)
11          stop
12      end if
13
14      call inpdat (ichoic)
15      call arrang
16      it1 = istime (i)                   ! CPU time (milli-sec)
17      goto (1,2,3,4,5,6), ichoic
18
19    1 continue                           ! forward problem, plots, ...
20          call pltdat (0)
21          goto 11
22    2 continue                           ! unconstrained optimization
23          call zoom                      ! seek single minimum
24          call corlat
25 c*       call seama
26          goto 11
27    3 continue                           ! grid scan  (    ,vary)
28          call scan2
29          call corlat
30 c*       call seama
31          goto 11
32    4 continue                           ! grid scan  (froz,vary)
33          call scan3
34          call corlat
35 c*       call seama
36          goto 11
37    5 continue                           ! sensitivity analysis
38          call corlat
39          call seama
40          goto 11
41    6 continue                           ! sensitivity analysis scans
42 c*       call seamax                    ! error plots, DOS
43          goto 11
44
45   11 it2 = iftime (i)                   ! CPU time (milli-sec)
46      it  = (it2-it1)/10                 ! CPU time (centi-sec) elapsed
47      its = 100                          ! clock units / second
48      itm = its*60                       !             / minute
```

```
      ith = itm*60                    !             / hour
      itd = ith*24                    !             / day

      id =    it/itd                  ! days
      it = it-id*itd
      ih =    it/ith                  ! hours
      it = it-ih*ith
      im =    it/itm                  ! minutes
      it = it-im*itm
      is =    it/its                  ! seconds
      it = it-is*its                  ! centi-seconds

      if (id.ne.0) then
          write (iout,111)  it,is,im,ih,id
      else if (ih.ne.0) then
          write (iout,111)  it,is,im,ih
      else if (im.ne.0) then
          write (iout,111)  it,is,im
      else if (is.ne.0) then
          write (iout,111)  it,is
      else
          write (iout,111)  it
      end if

      close (iout)
      stop

   12 write (iout,121)
      stop

  101 format (/' options:   1,  forward problems, plots, ... '
     &        /'            2,  search       (     vary) '
     &        /'            3,  search grid  (     vary) '
     &        /'            4,  search grid  (froz,vary) '
     &        /'            5,  sensitivity analysis '
     &        /'            6,  MAE plots ~ uncertainty '
     &       //' Enter:   option ~ ' )
  102 format ( '           option ~ ', i1 )


  111 format (/' elapsed cpu-time = ', i3, '  centi-seconds', :
     &        /'                  + ', i3, '  seconds      ', :
     &        /'                  + ', i3, '  minutes      ', :
     &        /'                  + ', i3, '  hours        ', :
     &        /'                  + ', i3, '  days         ')

  121 format (/' ... oops,  unable to discern:    option')
  122 format (/' ... oops, inconsistent value:    option')
      end
```

6.2.2 FILEOP.FOR

```
1        subroutine fileop
2        include  'iounit.'
3
4        open (idat,file='x.dat',status='old',readonly,shared)
5
6     11 open (iout,file='x.out',status='old',disp='delete',err=12)
7        close(iout)
8        goto 11
9     12 open (iout,file='x.out',status='new')
10
11    13 open (isout,file='x.sout',status='old',disp='delete',err=14)
12       close(isout)
13       goto 13
14    14 continue     ! open (isout,file='x.sout',status='new')
15
16    15 open (iplt,file='x.plot',status='old',disp='delete',err=16)
17       close(iplt)
18       goto 15
19    16 open (iplt,file='x.plot',status='new')
20
21       return
22       end
```

6.2.3 INPDAT.FOR

```
      subroutine inpdat (ichoic)
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      logical   first

      data    pi / 3.14159265E+00 /
      data  cccc / 2.99792458E+17 /    ! speed of light, (nm/sec)
c     data  cccc / 2.99792458E+14 /    ! speed of light, (micro-m/sec)
c     data  cccc / 2.99792458E+10 /    ! speed of light, (cm/sec)

c     Note:   1.0  micro-m  = 1E3  nano-m  =  1E4 Angstrom
c             1E-3 micro-m  =   1  nano-m  =   10 A
c             1E-4 micro-m  = 1E-1 nano-m  =    1 A


      raddeg = pi/180.0
      first  = .false.
c     ------------------------------------------------------

c     Distinct wavelengths (free space) incident on sample.
      read  (idat,  *)  mwaves
      write (iout,101)  mwaves
      if (mwaves.lt.1 .or. mwaves.gt.nwaves) then
          write (iout,102) nwaves
          stop
      end if
      do i=1,mwaves
          read  (idat,  *)  j, wavlen
          write (iout,103)  j, wavlen
          if (j.ne.i  .or.  wavlen.le.0.0) then
              write (iout,104)
              stop
          end if
          if (i.ne.1) then                       ! impose ordering
              if (waveln(i-1) .le. wavlen) then  !     longer waves first,
                  write (iout,105)               !     waves be distinct.
                  stop
              end if
          end if
          waveln(i) = wavlen                     ! nano-meters
          waveqq(i) = 2.0*pi/ wavlen
          waveww(i) = 2.0*pi* (cccc/wavlen)
      end do
c     ------------------------------------------------------

c     Distinct refractive indices of ambients ' (n| ambient,wave)
      read  (idat,  *)  mbient
      write (iout,111)  mbient
      nbnwav = nbient*nwaves                     ! ambients, waves
```

```
      if (mbient.lt.1 .or. mbient.gt.nbnwav) then
          write (iout,112)  nbnwav
          stop
      end if
      do i=1,mbient
          read  (idat,  *)  j, air              ! refractive index
          write (iout,113)  j, air
          if (j.ne.i .or. air.lt.1.0) then
              write (iout,114)
              stop
          else
              ambent(i) = air
          end if
      end do
c     ------------------------------------------------------

c     Distinct material elements:     (n,k| wave,sampl)
      read  (idat,  *)  mlmnts
      write (iout,121)  mlmnts
      if (mlmnts.lt.1 .or. mlmnts.gt.nlmnts) then
          write (iout,122) nlmnts
          stop
      end if
      do i=1,mlmnts
          read  (idat,  *)  j, dielec, uncert, ivary
          write (iout,123)  j, dielec, uncert, ivary
          if (j.ne.i        .or.
     &        dielec.lt.0.0 .or. uncert.lt.0.0 .or.
     &         ivary.lt.0   .or.  ivary.gt.2      ) then
              write (iout,124)
              stop
          end if
          if (ivary.eq.1 .and. uncert.eq.0.0) then
              write (iout,125)
              stop
          end if

          diefcn(i) = dielec                    ! n,k
          uncerl(i) = uncert
          lvaryl(i) = ivary

          iptw(i) = -1                          ! discern utilization
          if (ivary.eq.2 .and. ichoic.ne.4) then ! convenience
              first = .true.
          end if
      end do
c     ------------------------------------------------------

c     Distinct film thickness-es:   z
      read  (idat,  *)  mfilmm
      write (iout,131)  mfilmm
      if (mfilmm.lt.0 .or. mfilmm.gt.nfilmm) then
          write (iout,132) nfilmm
```

```
104           stop
105         end if
106         if (mfilmm.gt.0) then
107           do i=1,mfilmm
108             read  (idat,  *)  j, width, uncert, ivary
109             write (iout,133)  j, width, uncert, ivary
110             if (j.ne.i       .or.
111      &          width.lt.0.0 .or. uncert.lt.0.0 .or.
112      &          ivary.lt.0   .or.  ivary.gt.2      ) then
113               write (iout,134)
114               stop
115             end if
116             if (ivary.eq.1 .and. uncert.eq.0.0) then
117               write (iout,135)
118               stop
119             end if
120
121             widths(i) = width
122             uncerz(i) = uncert
123             lvaryz(i) = ivary
124
125             j = mlmnts+i
126             iptw(j) = -1                        ! discern utilization
127             if (ivary.eq.2 .and. ichoic.ne.4) then  ! convenience
128               first = .true.
129             end if
130           end do
131         end if
132
133         if (first) then                         ! impose constraint:
134           write (iout,137)                      !     ivary=2
135           stop                                  ! be allowed only for:
136         end if                                  !     ichoic=4
137   c     -----------------------------------------------------
138
139   c     Distinct sample configurations:     (films/substrate)
140         read  (idat,  *)  msampl
141         write (iout,141)  msampl
142         if (msampl.lt.1 .or. msampl.gt.nsampl) then
143           write (iout,142) nsampl
144           stop
145         end if
146         i = 0
147         iws = 0
148         iaws = 0
149
150         do is=1,msampl                  ! distinct samples
151           read  (idat,  *)  js, mfilm, mwave
152           write (iout,143)  js, mfilm, mwave
153           if (js.ne.is   .or.
154      &        mfilm.lt.0 .or. mfilm.gt.nfilms .or.
155      &        mwave.lt.1 .or. mwave.gt.mwaves     ) then
156             write (iout,144) nfilms, mwaves
```

```
157             stop
158         end if
159         mfilms = mfilm+1              ! films/substrate
160         nnfilm(is) = mfilm            ! number of films on sample
161         nnwave(is) = mwave            ! number of waves on sample
162
163         do iw=1,mwave                 ! scan distinct waves on sample
164             read  (idat,  *)  iwave, mbien
165             write (iout,145)  iwave, mbien
166             if (iwave.lt.1 .or. iwave.gt.mwaves .or.
167      &          mbien.lt.1 .or. mbien.gt.mbient     ) then
168                 write (iout,146)
169                 stop
170             end if
171             if (iw.ne.1) then         ! impose ordering
172                 if (iiwave(iws).ge.iwave) then
173                     write (iout,147)
174                     stop
175                 end if
176             end if
177             iws = iws+1
178             iiwave(iws) = iwave       ! specify the distinct wave
179             nnbent(iws) = mbien       ! number of distinct ambients
180
181             do m=1,mfilms                                  ! films/substrate
182                 if (m.eq.mfilms) then                      ! substrate
183                     read  (idat,  *)  j,  n,k
184                     write (iout,148)  j,  n,k
185                 else
186                     read  (idat,  *)  j,  n,k,iz           ! film
187                     write (iout,149)  j,  n,k,iz
188                     if (iz.lt.1 .or. iz.gt.mfilmm) then
189                         write (iout,150)
190                         stop
191                     end if
192                     i = i+1
193                     iifilm(i) = iz
194                 end if
195                 if (j.ne.m .or. n.eq.k      .or.
196      &              n.lt.1 .or. n.gt.mlmnts .or.
197      &              k.lt.1 .or. k.gt.mlmnts     ) then
198                     write (iout,150)
199                     stop
200                 end if
201                 i = i+1
202                 iifilm(i) = n
203                 i = i+1
204                 iifilm(i) = k
205
206                 j = mlmnts+iz                  ! indicate utilization
207                 iptw(j) = 0
208                 iptw(n) = 0
209                 iptw(k) = 0
```

```
210                 end do                                  ! film
211
212                 if (iw.ne.1) then                       ! test widths ~ z
213                     ii = i+1 - (mfilm*3+2)
214                     jj = i+1 - (mfilm*3+2)*2
215                     do iz=1,mfilm
216                         if (iifilm(jj).ne.iifilm(ii)) then
217                             write (iout,151) iz, iifilm(jj), iifilm(ii)
218                             stop
219                         end if
220                         ii = ii+3
221                         jj = jj+3
222                     end do
223                 end if    ! test z
224
225                 do mbn=1,mbien                          ! scan distinct ambients
226                     read  (idat,  *)  imbien, mrpeat
227                     write (iout,152)  imbien, mrpeat
228                     if (imbien.lt.1 .or. imbien.gt.mbient .or.
229       &                 mrpeat.lt.1 .or. mrpeat.gt.nrpeat     ) then
230                         write (iout,153)
231                         stop
232                     end if
233                     if (mbn.ne.1) then                  ! impose ordering
234                         if (imbien .le. iibent(iaws)) then
235                             write (iout,154)
236                             stop
237                         end if
238                     end if
239                     iaws = iaws+1
240                     iibent(iaws) = imbien               ! specify distinct ambient
241                     nnpeat(iaws) = mrpeat               ! repeats of experiment
242                 end do    ! ambient
243             end do       ! wave
244             write (iout,175)
245         end do           ! sample
246
247   c*    if (ichoic .eq. 1) then
248   c*        call pltdat (1)                             ! plot:  (z,n,k)
249   c*        return
250   c*    end if
251
252         k = 0                                           ! discern utilization of
253         mm = mlmnts+mfilmm                              !   the model parameters
254         do i=1,mm
255             if (iptw(i) .eq. -1)  then                  ! not utilized
256                 k = k+1
257                 if (i .le. mlmnts) then
258                     write (iout,156)  '(n+ik)', i
259                 else
260                     j = i-mlmnts
261                     write (iout,156)  '(z   )', j
262                 end if
```

```
263           end if
264         end do
265
266         if (k.ne.0) then                          ! impose constraint that
267              stop                                 !    all model parameters
268         end if                                    !    be of use.
269   c     ================================================================
270
271         if (ichoic .eq. 6)  goto 30               ! sensitivity analysis
272
273   c     Experimental, measurement, or target data.
274   c     (psi,delta |  phi,repeat,ambient,wave,sample)
275
276         i = 0                                     ! index measured data
277         ii = 0                                    ! overflow indicator
278         iws = 0
279         iaws = 0
280         iraws = 0
281         do is=1,msampl                            ! scan samples
282             mwave = nnwave(is)
283             first = .true.
284             do iw=1,mwave                         ! scan wavelengths
285                 iws = iws+1
286                 iwave = iiwave(iws)               ! specify wavelength
287                 mbien = nnbent(iws)
288                 do mbn=1,mbien                    ! scan ambients
289                     iaws = iaws+1
290                     imbien = iibent(iaws)         ! specify ambient
291                     mrpeat = nnpeat(iaws)
292                     do irpeat=1,mrpeat            ! repeats of experiment
293                         read (idat,  *)      mangl,       jmbn,jw
294                         if (first) then
295                             first = .false.
296                             write (iout,172) mangl,irpeat,jmbn,jw,is
297                         else
298                             write (iout,172) mangl,irpeat,jmbn,jw
299                         end if
300                         if (mangl.lt.1 .or. mangl.gt.nanglx .or.
301         &                   jmbn.ne.imbien .or. jw.ne.iwave     ) then
302                             write(iout,173)nanglx,irpeat,imbien,iwave,is
303                             stop
304                         end if
305                         iraws = iraws+1
306                         mangle(iraws) = mangl
307
308                         do iangl=1,mangl                    ! incident angles
309                             read  (idat,  *) j,angle,au,delta,du,psi,pu
310                             write (iout,174) j,angle,au,delta,du,psi,pu
311
312                             if (delta .lt. 0.0) then        ! [0, 360)
313                                 delta = delta + 360.0
314                             end if
315   c     ----------------------------------------------------------------
```

```
c     The above incoming measurement data (delta, psi)
c     is assumed to be of the Nebraska convention.
c     engineering assumes:    n-ik ---> (psi,delta) (Nebraska)
c        physics assumes:     n+ik ---> (psi,delta) (this software program)
c           both assume :     TM mode or p-polarization uses the H field.
c     Consequently, this affects only the Delta.
c   The transformation is:    (physics) <--- conjg (Nebraska)

                          if (delta .ne. 0.0) then
                              delta = 360.0 - delta
                          end if
c     ------------------------------------------------------------------

                          psi   =  abs (psi  )             ! [0,  90)
                          pu    =  abs (pu   )
                          delta =  abs (delta)             ! [0, 360)
                          du    =  abs (du   )
                          angle =  abs (angle)             ! [0,  90)
                          au    =  abs (au   )

                          psi   = amod (psi  ,  90.0)      ! degrees
                          pu    = amod (pu   ,  90.0)
                          delta = amod (delta, 360.0)
                          du    = amod (du   , 360.0)
                          angle = amod (angle,  90.0)
                          au    = amod (au   ,  90.0)

                          psi   = psi   *raddeg            ! radians
                          pu    = pu    *raddeg
                          delta = delta *raddeg
                          du    = du    *raddeg
                          angle = angle *raddeg
                          au    = au    *raddeg

                          i = i+1                          ! sequential index
                          if (i.gt.nexpts) then
                              i = i-1
                              ii = ii+1                    ! extention
                          end if

                          psiiis(i) = psi
                          psiiiu(i) = pu
                          deltas(i) = delta
                          deltau(i) = du
                          angles(i) = angle
                          angleu(i) = au

                      end do      ! angle
                      write (iout,175)
                  end do         ! repeat
              end do            ! ambient
          end do                ! wave
      end do                    ! sample
```

```
369
370         if (ii.gt.0) then
371             i = i+ii
372             write (iout,176) nexpts, i
373             stop
374         end if
375         return
376   c     ================================================================
377
378   c     Sensitivity/error analysis of multiple:  angle,ambient,wave,sample.
379   c     No need for experimental data, here.  But rather
380   c     we need:    manglm, isteps.
381   c     Assume:     mrpeat = 1,    ... in all cases.
382
383       30 read  (idat,  *)  method
384          write (iout,161)  method
385          if (method.lt.1 .or. method.gt.3) then
386              write (iout,162)
387              stop
388          end if
389
390          i = 0                                        ! index measured data
391          ii = 0                                       ! overflow indicator
392          iws = 0
393          iaws = 0
394          iraws = 0
395          do is=1,msampl                               ! scan samples
396              mwave = nnwave(is)
397              first = .true.
398              do iw=1,mwave                            ! scan wavelengths
399                  iws = iws+1
400                  iwave = iiwave(iws)                  ! specify wavelength
401                  mbien = nnbent(iws)
402                  do mbn=1,mbien                       ! scan ambients
403                      iaws = iaws+1
404                      imbien = iibent(iaws)            ! specify ambient
405                      mrpeat = nnpeat(iaws)
406                      if (mrpeat.gt.1) then            ! consistencey
407                          write (iout,181) mrpeat
408                          stop
409                      end if
410   c*                 do irpeat=1,mrpeat               ! repeats of experiment
411                          read (idat,  *)      istep,mangl, jmbn,jw,js
412                          if (first) then
413                              first = .false.
414                              write (iout,182) istep,mangl, jmbn,jw,js
415                          else
416                              write (iout,182) istep,mangl, jmbn,jw
417                          end if
418                          if (istep.lt.1 .or. istep.gt.10     .or.
419         &                    mangl.lt.1 .or. mangl.gt.nanglm .or.
420         &                     jmbn.ne.imbien .or.
421         &                       jw.ne.iwave  .or.  js.ne.is    ) then
```

```
                        write (iout,183) nanglm,imbien,iwave,is
                        stop
                     end if
                     iraws = iraws+1
                     mangle(iraws) = mangl      ! multiple angles
                     isteps(iraws) = istep      ! increment of grid
c*                end do         ! repeat
               end do            ! ambient
            end do               ! wave
         end do                  ! sample
         return

  101 format(/' mwaves =', i4, ', number of distinct wavelengths')
  102 format (' nwaves =', i4, ', <--------- oops')
  103 format ('         ', i4, f12.4)
  104 format (' oops,   ... card info inconsistent')
  105 format (' oops,   ... wavelengths not:    ordered,distinct.')
  111 format(/' mbient =', i4, ', number of distinct ambient',
     &                          ' environments and waves.')
  112 format (' nbnwav =', i4, ', <--------- oops')
  113 format ('         ', i4, 1f12.6)
  114 format (' oops,   ... card info inconsistent.')
  121 format(/' mlmnts =', i4, ', number of distinct:   n+ik')
  122 format (' nlmnts =', i4, ', <--------- oops')
  123 format ('         ', i4, 2f12.4, i5)
  124 format (' oops,   ... card info inconsistent')
  125 format (' oops,   since:     ivary =   1,  '
     &       /'          then:    uncert =/= 0.0 ')
  131 format(/' mfilmm =', i4, ', number of distinct film widths')
  132 format (' nfilmm =', i4, ', <--------- oops')
  133 format ('         ', i4, 2f12.4, i5)
  134 format (' oops,   ... card info inconsistent')
  135 format (' oops,   since:    ivary =   1,  '
     &       /'          then:   uncert =/= 0.0 ')
  137 format (' oops,   an imposed constraint has been violated:'
     &       /'         it is only in the case of:    ichoic = 4'
     &       /'         may one be allowed to use:     ivary = 2' )
  141 format(/' msampl =', i4, ', number of distinct samples')
  142 format (' nsampl =', i4, ', <--------- oops')
  143 format ('         ', 3i4, 8x, ' ' sample, mfilm, mwave')
  144 format (' oops,     nfilms=', i4, ',    mwaves=', i4)
  145 format ('         ', 2i4, 12x, ' ' iwave, mbien')
  146 format (' oops,   ... card info inconsistent')
  147 format (' oops,   ... distinct, ordered     ')
  148 format (  9x, 3i4, 8x, ' ' i,n,k  ')
  149 format (  9x, 4i4, 4x, ' ' i,n,k,z')
  150 format (' oops,   ... card info inconsistent')
  151 format (' oops,       compare corresponding widths: '
     &       /'            iz, iifilm(jj), iifilm(ii)  : ', 3i5)
  152 format ('         ', 2i4, 12x, ' ' imbien, mrpeat')
  153 format (' oops,   ... card info inconsistent')
  154 format (' oops,   ... impose ordering on the:   ambients')
  156 format (' oops,   parameter not in sample configuration,  ',
```

```
475      &                                                  a6, ' ' ', i3)
476    161 format (' method =', i4, ', (1 ' normal eqns,  '
477      &       /     16x, '2 ' permute signs in ''forward'' problem,'
478      &       /     16x, '3 ' singular value decomposition ')
479    162 format (' oops,   ... card info inconsistent')
480    172 format (1x, 5i4, ' ' mangl, repeat,ambient,wave,sample',
481      &                '/ (phi,delta,psi)')
482    173 format (' oops,   ... card info inconsistent'
483      &       /1x, 5i4, ' ' nanglx, repeat, ambient, wave, sample')
484    174 format (1x, i4, 6f12.4)
485    175 format (' ')
486    176 format (' oops,   enlarge:  nexpts=', i4, ' ----> ', i4)
487    181 format (' oops,   repeats =', i4, ',  ----> 1')
488    182 format (1x, 5i4, ' ' istep,mangl, ambient,wave,sample')
489    183 format (' oops,   ... card info inconsistent'
490      &       /'      istep <  11'
491      &       /'      mangl <', i4, ' ' nanglm '
492      &       / 13x, 3i4, 4x, ' ' (ambient, wave, sample)')
493       end
```

6.2.4 ARRANG.FOR

This subroutine sets up the necessary pointers for indexing/ordering the matrix elements in the sparse matrix of the Jacobian.

```
1       subroutine  arrang
2       include  'defnit.'
3       include  'filmmm.'
4
5  c    The sparse matrix format for the model parameters is of the form:
6  c        [(n/k)_(1), ..., (n/k)_(mlmnts)]  ' diefcn (w)
7  c        [    z_(1), ...,     z_(mfilms)]  ' widths
8
9  c    Ordering of indices in IPTU:     [vary-->     <--froz]
10
11
12      mm = mlmnts+mfilmm                          ! model parameters
13      i = 0                                       ! vary ' compress
14      k = 0                                       ! froz ' compress
15      do m=1,mlmnts
16          iptw(m) = 0                             ! local uniqueness
17          if (lvaryl(m) .eq. 1) then              ! vary
18              i = i+1                             !     compress
19              iptu(i) = m                         !     full (vary)
20              iptv(m) = i                         !     vary (full)
21          else                                    ! frozen
22              k = k+1                             !     compress
23              km = mm+1-k                         !     backwards
24              iptu(km) = m                        !     full (froz)
25              iptv( m) = km                       !     froz (full)
26          end if
27      end do
28
29      if (mfilmm.ne.0) then
30          do m=1,mfilmm
31              j = mlmnts+m
32              iptw(j) = 0                         ! local uniqueness
33              if (lvaryz(m) .eq. 1) then          ! vary
34                  i = i+1                         !     compress
35                  iptu(i) = j                     !     full (vary)
36                  iptv(j) = i                     !     vary (full)
37              else                                ! frozen
38                  k = k+1                         !     compress
39                  km = mm+1-k                     !     backwards
40                  iptu(km) = j                    !     full (froz)
41                  iptv( j) = km                   !     froz (full)
42              end if
43          end do
44      end if
45
46      mvary = i                                   ! compress
```

```
47          mfroz = k                                   ! compress
48
49          return
50          end
```

6.2.5 PLTDAT.FOR

```
      subroutine pltdat (ichoic)
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      include  'filmss.'
      include  'rstack.'
      include  'seamx1.'

      real  a (nrows*2), b(2), c(2)
      real  uz(nrows*2), un(nrows*2), uk(nrows*2)
      character*12  labelo                        ! convenience label
      data  pi / 3.14159265 /

c     The sparse matrix format for the model parameters is of the form:
c         [(n/k)_(1), ..., (n/k)_(mlmnts)]  ' diefcn (w)
c         [    z_(1), ...,     z_(mfilms)]  ' widths


      j = ichoic
      if (j.lt.1 .or. j.gt. 5) then
          write (iout,101)
          read  (idat,  *)  j
          write (iout,102)  j
          if (j.lt.1 .or. j.gt.5) then
              stop
          end if
      end if

      if (j .eq. 1)  goto 1
      if (j .eq. 2)  goto 2
      if (j .eq. 3)  goto 3
      if (j .eq. 4)  goto 4
      if (j .eq. 5)  goto 5

c     ================================================================
c     Plot distribution:  (z,n,k)  for those samples
c     which have one or more films.

    1 continue
      i = 0
      iws = 0
      iaws = 0
      do is=1,msampl
          mfilm = nnfilm(is)
          mwave = nnwave(is)
          if (mfilm .ne. 0) then
              mfilms = mfilm+1                    ! films/substrate
              do iw=1,mwave
                  iws = iws+1
                  iwave = iiwave(iws)
```

```
51                  mbien = nnbent(iws)
52                  qq  = waveqq(iwave)             ! FILMSS
53
54                  k = 2                           ! index of plot array
55                  uk(1) = 0.0                     ! extinction of ambient
56                  uk(2) = 0.0
57                  uz(2) = 0.0                     ! surface
58
59                  do m=1,mfilms                   ! films/substrate
60                      if (m.ne.mfilms) then       ! z
61                          i = i+1
62                          iz = iifilm(i)
63                          zzz(m) = widths(iz)
64                          uz(k+1) = uz(k)
65                          uz(k+2) = uz(k) + zzz(m)
66                          if (m.eq.1) then
67                              thin = zzz(m)
68                          else
69                              thin = amin1 (thin, zzz(m))
70                          end if
71                      end if
72                      do ink=1,2                  ! n+ik
73                          i = i+1
74                          nk = iifilm(i)
75                          c(ink) = diefcn(nk)
76                      end do
77                      un(k+1) = c(1)              ! LHS
78                      un(k+2) = c(1)              ! RHS
79                      uk(k+1) = c(2)              ! LHS
80                      uk(k+2) = c(2)              ! RHS
81                      k = k+2
82                  end do                          ! mfilms
83                  uz(  1) = -thin
84                  uz(k-1) =  uz(k-2)              ! substrate/film interface
85                  uz(k  ) =  uz(k-2) + thin       ! substrate
86
87                  do mbn=1,mbien                  ! ambients
88                      iaws = iaws+1
89                      imbien = iibent(iaws)
90                      air = ambent(imbien)        ! FILMSS
91                      un(1) = air                 ! refractive index
92                      un(2) = air
93
94                      nu = 2                      ! n+ik
95                      write (iplt,111) k,nu,mbn,iw,is
96                      do j=1,k
97                          write (iplt,112)  j, uz(j), un(j), uk(j)
98                      end do
99                  end do      ! ambient
100             end do          ! wave
101         else                                    ! advance indices
102             i = i + (mfilm*3+2)*mwave
103             do iw=1,mwave
```

```
                  iws = iws+1
                  mbien = nnbent(iws)
                  iaws = iaws+mbien
              end do
          end if
      end do                  ! sample
      return

c     ================================================================
c     Plot deviations of the fit:     b = experiment-model

    2 continue
      llnorm = .false.
      call asmbl                      ! bb

      rd  = 180.0/pi                  ! radians --> degrees
      ngl = 0
      ii  = 0
      iws = 0
      iaws = 0
      iraws = 0
      do is=1,msampl
          mwave = nnwave(is)
          do iw=1,mwave
              iws = iws+1
              mbien = nnbent(iws)
              do mbn=1,mbien
                  iaws = iaws+1
                  mrpeat = nnpeat(iaws)
                  do irpeat=1,mrpeat
                      iraws = iraws+1
                      mangl = mangle(iraws)

                      nu = 2                      !  (psi, delta)
                      write (iplt,211) mangl,nu,irpeat,mbn,iw,is
                      do iangl=1,mangl
                          ngl = ngl+1
                          angl = angles(ngl)      ! radians
                          psixm = bb(ii+1)
                          delxm = bb(ii+2)
                          angl  = angl *rd        ! degrees
                          psixm = psixm*rd
                          delxm = delxm*rd
                          write (iplt,212) iangl,angl,delxm,psixm
                          ii = ii+2
                      end do      ! angle
                  end do         ! repeat
              end do            ! ambient
          end do               ! wave
      end do                  ! sample
      return

c     ================================================================
```

```
c     Output model experimental data suitable for use as input data.

    3 continue                                        ! experiment <--- model
      call arrang

      mws = 0
      iws = 0
      iaws = 0
      iraws = 0
      ngl = 0                                         ! measured data
      do is=1,msampl
          mfilm = nnfilm(is)                          ! FILMSS
          mwave = nnwave(is)
          mfilms = mfilm+1                            ! films/substrate
          nrow = mfilm*3+2
          do iw=1,mwave
              iws = iws+1
              iwave = iiwave(iws)
              mbien = nnbent(iws)
              qq  = waveqq(iwave)                     ! FILMSS

              do m=1,mfilms                           ! films/substrate
                  if (m.ne.mfilms) then               ! films " z
                      mws = mws+1
                      iz = iifilm(mws)
                      zzz(m) = widths(iz)             ! FILMSS
                  end if
                  do ink=1,2                          ! n+ik
                      mws = mws+1
                      nk = iifilm(mws)
                      c(ink) = diefcn(nk)             ! n,k
                  end do
                  die(m) = cmplx (c(1),c(2))**2       ! FILMSS
              end do          ! mfilms

              do mbn=1,mbien                          ! ambients
                  iaws = iaws+1
                  imbien = iibent(iaws)
                  mrpeat = nnpeat(iaws)
                  air = ambent(imbien)                ! FILMSS
                  do irpeat=1,mrpeat
                      iraws = iraws+1
                      mangl = mangle(iraws)
                      write (iout,311)  mangl,imbien,iwave
                      write (iplt,311)  mangl,imbien,iwave
                      do iangl=1,mangl                ! incident angles
                          ngl = ngl+1                 ! measured data
                          angl = angles(ngl)
                          call scatr (qq,angl, b,a,c)

                          psi   = b(1)       *180.0/pi     ! psi
                          delta = b(2)       *180.0/pi     ! delta
                          angl  = angl       *180.0/pi
```

```
                        au   = angleu(ngl) *180.0/pi
                        pu   = psiiiu(ngl) *180.0/pi
                        du   = deltau(ngl) *180.0/pi

                        if (delta.lt.0.0) then          ! [0,360)
                            delta = delta + 360.0
                        end if

                        if (delta.ne.0.0) then          ! phase shift
                            delta = 360.0 - delta
                        end if

                        write (iout,312)  iangl, angl,au,
     &                                    delta,du,  psi,pu
                        write (iplt,312)  iangl, angl,au,
     &                                    delta,du,  psi,pu
                    end do        ! angle
                end do           ! repeat
            end do              ! ambient
        end do                 ! wave
      end do                  ! sample
      return

c     ================================================================
c     Output model experimental data,    (delta,psi, R,|R|, ... )
c     Scan incident angles.

    4 continue
      write (iout,411)
      read  (idat,  *)  ichoos
      write (iout,412)  ichoos
      if (ichoos.lt.1 .or. ichoos.gt.4) then
          write (iout,413)  'choose'
          stop
      end if
      write (iout,421)
      read  (idat,  *)  iselct
      write (iout,422)  iselct
      if (iselct.lt.1 .or. iselct.gt.2) then
          write (iout,413)  'select'
          stop
      end if
      write (labelo,423)  ichoos, iselct         ! label output

      call arrang
      mws = 0
      iws = 0
      iaws = 0
      iraws = 0
      ngl = 0
      do is=1,msampl
          mfilm = nnfilm(is)                     ! FILMSS
          mwave = nnwave(is)
```

```
263             mfilms = mfilm+1                        ! films/substrate
264             nrow = mfilm*3+2
265             do iw=1,mwave
266                 iws = iws+1
267                 iwave = iiwave(iws)
268                 mbien = nnbent(iws)
269                 qq  = waveqq(iwave)                 ! FILMSS
270
271                 do i=1,nrow                         ! convenience
272                     iptx(i) = 0
273                     ipty(i) = 0
274                 end do
275                 iv = 0
276                 kv = 0
277                 mv = 0
278
279                 do m=1,mfilms                       ! films/substrate
280                     if (m.ne.mfilms) then           ! films ' z
281                         mws = mws+1
282                         iz = iifilm(mws)
283                         zzz(m) = widths(iz)         ! FILMSS
284                         iv = iv+1
285                         if (lvaryz(iz) .eq. 1) then
286                             j = mlmnts+iz
287                             i = iptv(j)
288                             kv = kv+1
289                             iptx(kv) = i
290                             ipty(kv) = iv
291                             if (iptw(i).eq.0) then ! unique
292                                 mv = mv+1
293                                 iptw(i) = mv
294                             end if
295                         end if
296                     end if
297                     do ink=1,2                      ! n+ik
298                         mws = mws+1
299                         nk = iifilm(mws)
300                         c(ink) = diefcn(nk)         ! n,k
301                         iv = iv+1
302                         if (lvaryl(nk) .eq. 1) then
303                             i = iptv(nk)
304                             kv = kv+1
305                             iptx(kv) = i
306                             ipty(kv) = iv
307                             if (iptw(i).eq.0) then
308                                 mv = mv+1
309                                 iptw(i) = mv
310                             end if
311                         end if
312                     end do
313                     die(m) = cmplx (c(1),c(2))**2  ! FILMSS
314                 end do          ! mfilms
315
```

```
316             if (ichoos.eq.4 .and. mv.gt.1) then
317                 write (iout,431)
318                 stop
319             end if
320
321             do mbn=1,mbien                      ! ambients
322                 iaws = iaws+1
323                 imbien = iibent(iaws)
324                 mrpeat = nnpeat(iaws)
325                 air = ambent(imbien)            ! FILMSS
326                 do irpeat=1,mrpeat
327                     iraws = iraws+1
328                     if (iselct.eq.1) then       ! specific angles
329                         mangl = mangle(iraws)
330                     else                        ! scan angles
331                         mangl = 89
332                     end if
333                     if (ichoos.eq.1)  ncolmn=3
334                     if (ichoos.eq.2)  ncolmn=4
335                     if (ichoos.eq.3)  ncolmn=3
336                     if (ichoos.eq.4)  ncolmn=2
337
338                     write (iplt,432)  mangl,ncolmn,imbien,iwave,is,
339                                       labelo
340
341                     do iangl=1,mangl                    ! angles
342                         if (iselct.eq.1) then
343                             ngl = ngl+1                 ! measured data
344                             angl = angles(ngl)          ! radians
345                             phi = angl*(180.0/pi)       ! degrees
346                         else
347                             phi = float (iangl)         ! degrees
348                             angl = phi*(pi/180.0)       ! radians
349                         end if
350                         call forwrd (qq,angl,  Rs,Rp,  dRs,dRp,
351                                                        dRsa,dRpa)
352                         sx =  real (Rs)
353                         sy = aimag (Rs)
354                         px =  real (Rp)
355                         py = aimag (Rp)
356                         call polar (sx,sy,sr,sa,1)      !  |Rs|
357                         call polar (px,py,pr,pa,1)      !  |Rp|
358
359                         sr2 = sr*sr                     ! intensity
360                         pr2 = pr*pr
361                         sr2pr2 = (sr2+pr2)*0.5          !     total
362
363                         call scatr (qq,angl, b,a,c)
364                         psi   = b(1) *180.0/pi          !   psi
365                         delta = b(2) *180.0/pi          ! delta
366
367                         if (mv.eq.0) then               ! Jacobian
368                             dpsi = c(1) *180.0/pi       !   psi'
```

```
                                ddel = c(2) *180.0/pi        ! delta'
                            else
                                dpsi = 0.0                   ! initialize
                                ddel = 0.0
                                do k=1,kv                    ! coompress, vary
                                    iv  = ipty(k)            ! local, full
                                    iv2 = iv+iv              ! delta ' a ' local
                                    iv1 = iv2-1              !   psi ' a ' local
                                    dpsi = dpsi + a(iv1)
                                    ddel = ddel + a(iv2)
                                end do
                                dpsi = dpsi *180.0/pi
                                ddel = ddel *180.0/pi
                            end if

                            if (delta.lt.0.0) then           ! [0,360)
                                delta = delta + 360.0
                            end if

                            if (delta.ne.0.0) then           ! phase shift
                                delta = 360.0 - delta
                            end if

c                           ------------------------------------------
                            if      (ichoos.eq.1) then
                                write (iplt,433)  iangl,phi,
     &                                           delta,psi, sr2pr2
                            else if (ichoos.eq.2) then
                                write (iplt,433)  iangl,phi,
     &                                           sr,pr, sr2,pr2
                            else if (ichoos.eq.3) then
                                write (iplt,433)  iangl,phi,
     &                                           sr,pr, sr2pr2
                            else if (ichoos.eq.4) then
                                write (iplt,433)  iangl,phi,
     &                                           ddel, dpsi
                            else
                                stop
                            end if
c                           ------------------------------------------

                        end do        ! angle
                    end do           ! repeat
                end do               ! ambient

                do k=1,kv
                    i = iptx(k)
                    iptw(i) = 0
                end do

            end do                  ! wave
        end do                     ! sample
        write (iout,434)
```

```
      return

c     ================================================================
c     Output/plot |g|  over a grid of model parameters (v),  1D or 2D.

    5 continue
      call scan2g
      return

c     ================================================================
  101 format (/' PLTDAT,  forward scattering problems, plots, ... '
     &        /'          This menu of options is at level two.'
     &    //' options:   1,  plot (z,n,k)  '
     &     /'            2,  plot deviations of fit,   ',
     &                              'g = experiment-model'
     &     /'            3,  model ---> experiment/measurement data,'
     &     /'                use model to simulate measurement data,'
     &     /'                format is suitable to reading as input,'
     &     /'                (angles,repeats,ambients,waves,samples)'
     &     /'            4,  use model at measurement/scan points,  '
     &     /'                request another menu to format output, '
     &     /'                output '' (delta,psi,intensity, R,dR, ...)'
     &     /'            5,  scan grid,  plot |g|,        1D or 2D  '
     &       //' Enter:   option '' '      )
  102 format ( '          option '', i1 /)


  111 format (1x, 5i4, ' '' nz,nu,  ambient,wave,sample / (i,z,n,k)')
  112 format (1x,  i5, 1p3e13.4)
  211 format (1x, 6i4, ' '' manglx,nu,  repeat,ambient,wave,sample',
     &                                  ' / (i,phi, ddel,dpsi)')
  212 format (1x,  i5, 1p3e13.4)
  311 format (1x, 3i4, ' '' mangl,imbien,iwave/(phi,delta,psi)',
     &                                    ' (angle, d_angle)')
  312 format (1x, i4, 1x, 6f10.4)


  411 format (/' PLTDAT,  select the format of the output table.'
     &        /'          This menu of options is at level three.'
     &       //' options:   1,  (i,ai,  delta,psi,  intensity      )'
     &        /'            2,  (i,ai,  |Rs|,|Rp|,  |Rs|**2,|Rp|**2)'
     &        /'            3,  (i,ai,  |Rs|,|Rp|,  intensity      )'
     &        /'            4,  (i,ai,  d/d parameter (delta, psi) )'
     &       //' Enter:    option '' ')
  412 format ( '           option '', i1 )
  413 format ( ' oops, inconsistent value ascribed unto:   ', a6)


  421 format (/' PLTDAT,   select the domain of incident angles.  '
     &        /'          This menu of options is at level three.'
     &       //' options:   1,  at experiment/measurement points  '
     &        /'            2,  grid scan, incident angles '' (1,89)'
     &       //' Enter:    option '' ')
```

```
475     422 format ( '           option ' ', i1 /)
476     423 format ( 'option:(', i1, ',', i1, ')')     ! 12 characters
477
478
479     431 format ( ' ... oops, only (0 or 1) parameters may   VARY')
480     432 format (1x, 5i4, ' ' mangl,nu, imbien,iwave,sampl,', a)
481     433 format (1x, i4, 1x, f10.4,  1p4e13.4)
482     434 format (/' output written to:   x.plot')
483
484         end
```

6.2.6 SCAN2.FOR

```
      subroutine scan2
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      include  'scancc.'

      logical   lbdry, local
      character blank, comma
      data      blank /' '/,    comma /','/

c     Solve the forward scattering problem over a grid of model parameters.

c     Input grid info for scanning the model parameters

      raddeg = 180.0 / 3.14159265                ! degrees <--- radians

      write (iout,101)
      i = 0
      do m=1,mlmnts                              ! (n+ik)
          if (lvaryl(m) .eq. 1) then
              i = i+1
              read  (idat,  *)     j, p1, p2, p3
              write (iout,102)  i,    p1, p2, p3, j, '(n+ik)'

              if ( j.ne.m   .or. p2.lt.p1  .or.
     &             p1.lt.0.0 .or. p2.lt.0.0 .or. p3.lt.0.0) then
                  write (iout,103)
                  stop
              end if
              if (p3.eq.0.0) p3=p2-p1
              if (p3.eq.0.0) then
                  iii2(i) = 1
              else
                  iii2(i) = 1 + nint ((p2-p1)/p3)
              end if
              ppp1(i) = p1
              ppp2(i) = p2
              ppp3(i) = (p2-p1) /float (max (1, iii2(i)-1))
c             psav(i) = diefcn(m)                 ! n+ik
              iptu(i) = m
          end if
      end do       ! mlmnts

      if (mfilmm.ne.0) then
          do m=1,mfilmm                          ! z ' thickness
              if (lvaryz(m) .eq. 1) then
                  i = i+1
                  read  (idat,  *)     j, p1, p2, p3
                  write (iout,102)  i,    p1, p2, p3, j, '(z   )'

```

```
51                      if ( j.ne.m   .or. p2.lt.p1  .or.
52       &                   p1.lt.0.0 .or. p2.lt.0.0 .or. p3.lt.0.0) then
53                          write (iout,103)
54                          stop
55                      end if
56                      if (p3.eq.0.0) p3=p2-p1
57                      if (p3.eq.0.0) then
58                          iii2(i) = 1
59                      else
60                          iii2(i) = 1 + nint ((p2-p1)/p3)
61                      end if
62                      ppp1(i) = p1
63                      ppp2(i) = p2
64                      ppp3(i) = (p2-p1) /float (max (1, iii2(i)-1))
65   c                  psav(i) = widths(m)
66                      iptu(i) = mlmnts+m
67                  end if
68              end do   ! mfilmm
69          end if
70          mvary = i    ! depth of do-loop nest
71
72   c      ------------------------------------------------------
73   c      Discern breakpoint
74
75          read  (idat,*,err=11,end=11)  mvarii, kt, kts, old
76          write (iout,121)
77          write (iout,111)              mvarii, kt, kts, old
78          if (mvarii.ne.mvary .or. kt.lt.1 .or. kts.lt.1 .or.
79         &                                      old.le.0.0   ) goto 11
80          do i=1,mvary
81              read  (idat,*,err=11,end=11)  j, iiii(i), pppp(i), psav(i)
82              write (iout,112)              j, iiii(i), pppp(i), psav(i)
83
84              if (iiii(i) .lt. 1        .or.   j.ne.i  .or.
85         &        iiii(i) .gt. iii2(i)                  .or.
86         &        pppp(i) .lt. ppp1(i)-ppp3(i)*0.001    .or.
87         &        pppp(i) .gt. ppp2(i)+ppp3(i)*0.001    .or.
88         &        psav(i) .lt. ppp1(i)-ppp3(i)*0.001    .or.
89         &        psav(i) .gt. ppp2(i)+ppp3(i)*0.001      ) then
90                  write (iout,113)
91                  stop                ! goto 11
92              end if
93
94              if (iiii(i).lt.iii2(i)  .or.  iii2(i).eq.1) then
95                  ppppi = ppp1(i)+ppp3(i)*float(iiii(i) -1)    ! [,)
96              else
97                  ppppi = ppp2(i)                              !   ]
98              end if
99              if (abs(pppp(i)-ppppi) .gt. ppp3(i)*0.01) then
100                 write (iout,113)
101                 stop                ! goto 11
102             end if
103         end do
```

```
104         write (iout,114)
105         iiistp = 1
106         iii = mvary
107         it1 = iftime (i)                        ! start clock
108         goto 2
109      11 continue
110         write (iout,115)
111         it1 = iftime (i)                        ! start clock
112
113   c     ========================================================
114   c     Emulate, initialize, activate:   the nest of do-loops
115
116         do i=1,mvary
117             iiii(i) = 0                         ! initialization:   i1-i3
118         end do
119         kt = 0
120         iiistp = 1                              ! do-loop increment
121         iii = 0                                 ! index of:  nested do-variables.
122       1 iii = iii+1                             ! index of:  iii-th    nested do.
123         if (iii .gt. mvary) goto 3
124       2 iiii(iii) = iiii(iii) + iiistp          ! update do-loop variable
125         if (iiii(iii) .lt. iii2(iii)) then      ! test upper limit
126             pppp(iii) = ppp1(iii) + ppp3(iii)*float (iiii(iii) -1)
127             goto 1
128         else if (iiii(iii) .eq. iii2(iii)) then
129             if (iii2(iii) .eq. 1) then
130                 pppp(iii) = ppp1(iii)
131             else
132                 pppp(iii) = ppp2(iii)
133             end if
134             goto 1
135         end if
136         iiii(iii) = 0                           ! reset inner do,  i1-i3
137         iii = iii-1                             ! backup one do-level
138         if (iii .eq. 0) goto 4                  ! escape nest of do-loops
139         goto 2
140       3 iii = iii-1                             ! level of inner-most nested do.
141   c     ------------------------------------    ! begin processing body
142
143         kt = kt+1                               ! simple counter
144         do i=1,mvary                            ! re-set model parameters
145             m = iptu(i)
146             if (m.le.mlmnts) then
147                 diefcn(m) = pppp(i)             ! n+ik
148             else
149                 m = m-mlmnts
150                 widths(m) = pppp(i)             ! z
151             end if
152         end do       ! vary
153
154         call zoom2 (ppp1, ppp2, ppp3)           ! iterate ' steepest descent
155
156         llnorm = .false.
```

```
157         call asmbl
158         call norm (meqns,bb,bnorm,1)          ! retain norm of residual
159         bnorm = bnorm*raddeg                  ! degrees
160
161         if (kt.eq.1) then
162             kts = 1                           ! density of states
163             old = bnorm
164             do i=1,mvary                      ! retain model parameters
165                 m = iptu(i)
166                 if (m.le.mlmnts) then
167                     psav(i) = diefcn(m)
168                 else
169                     m = m-mlmnts
170                     psav(i) = widths(m)
171                 end if
172             end do        ! vary
173         else if (bnorm.lt.old) then
174             kts = 1
175             old = bnorm
176             do i=1,mvary
177                 m = iptu(i)
178                 if (m.le.mlmnts) then
179                     psav(i) = diefcn(m)
180                 else
181                     m = m-mlmnts
182                     psav(i) = widths(m)
183                 end if
184             end do        ! vary
185         else if (bnorm.eq.old) then           ! retain density of states
186             kts = kts+1                       !        along the minimum
187         end if
188
189         it2 = iftime (i)                      ! CPU clock
190         tim = float (it2-it1) /6.0E4          ! CPU minutes elapsed
191         if (tim .gt. 15.0) then               ! breakpoint
192             it1 = it2
193             open (unit=isout, file='x.sout', status='unknown')
194             write (isout,111) mvary, kt, kts, old
195             do i=1,mvary
196                 write (isout,112) i, iiii(i), pppp(i), psav(i)
197             end do
198             call time (bufft)
199             call date (buffd)
200             write (isout,116)  bufft, buffd   ! convenience
201             close (unit=isout)
202         end if
203 c       ------------------------------------ ! finish processing body
204         goto 2                                ! nested do-loop:   iiii
205       4 continue                              ! last line of do-nest,  iiii
206 c       ====================================
207         lbdry = .false.
208
209         write (iout,104) kt, kts, old         ! density of states along minimum
```

```
      write (iout,105)
      do i=1,mvary
          local = (psav(i) .le. ppp1(i)+ppp3(i)*0.001) .or.
     &            (psav(i) .ge. ppp2(i)-ppp3(i)*0.001)
c*        local = local .and. (iii2(i).ne.1)
          if (local)  lbdry=.true.
          m = iptu(i)
          if (m.le.mlmnts) then             ! n+ik ' diefcn
              diefcn(m) = psav(i)
              if (local) then
                  write (iout,106)  i, psav(i), m, '(n+ik)', blank
              else
                  write (iout,106)  i, psav(i), m, '(n+ik)'
              end if
          else                              ! z    ' widths
              m = m-mlmnts
              widths(m) = psav(i)
              if (local) then
                  write (iout,106)  i, psav(i), m, '(z   )', blank
              else
                  write (iout,106)  i, psav(i), m, '(z   )'
              end if
          end if
      end do       ! vary
      if (lbdry)  write (iout,107)

      return

  101 format (/' Scan a grid of model parameters.'
     &        /' Grid info:',  4x,  'initial,',  6x,  'final,',
     &                                          5x, 'increment' )
  102 format ( 3x,  i4,  ') ',  3f13.4, '   ' for:', i5, ',  ', a6)
  103 format (/' scan,     error:   inconsistent data input')
  104 format (/' number of grid points scanned, kt  = ', i10
     &        /'  population along the minimum, kts = ', i10
     &        /'              norm of residual, |g| = ', 1pe13.5,
     &                          '   (degrees) ')
  105 format (/' Model parameter value along the minimum:')
  106 format (1x,i4, ')',  1p1e15.6,  '   ' for:',  i5,  ',  ',  a6,
     &                         a1,   '  ' boundary')
  107 format (/' Note that the minimum point is near a boundary.')

  111 format (1x, i4,1x, 2i10, 1p1e15.6, ' ' mvary, kt, kts, b(min)')
  112 format (1x, 2(i4,1x),    1p2e15.6, ' ' i, ip, p, p(min)')
  113 format (/' The attempted restart was NOT acceptable.')
  114 format (/' The attempted restart was     acceptable.')
  115 format (/' Note:   NO attempt was made to restart.')
  116 format ( ' wall clock:    time = ', a8, 1x, ' ' hh:mm:ss '
     &       / '               date = ', a9,     ' ' dd-mmm-yy' )

  121 format (' ')
      end
```

6.2.7 SCAN2G.FOR

```
1       subroutine scan2g                                ! Plot |g|
2       include 'iounit.'                                ! Called by: PLTDAT
3       include 'defnit.'
4       include 'filmmm.'
5       include 'scancc.'
6
7       logical   lbdry, local
8       character blank, comma
9       data      blank /' '/,    comma /','/
10
11  c   Solve the forward scattering problem over a grid of model parameters.
12
13  c   Input grid info for scanning the model parameters
14
15      raddeg = 180.0 / 3.14159265                      ! degrees <--- radians
16
17      write (iout,101)
18      i = 0
19      do m=1,mlmnts                                    ! (n+ik)
20          if (lvaryl(m) .eq. 1) then
21              i = i+1
22              read  (idat,  *)     j, p1, p2, p3
23              write (iout,102)  i,    p1, p2, p3, j, '(n+ik)'
24
25              if ( j.ne.m   .or. p2.lt.p1  .or.
26     &            p1.lt.0.0 .or. p2.lt.0.0 .or. p3.lt.0.0) then
27                  write (iout,103)
28                  stop
29              end if
30              if (p3.eq.0.0) p3=p2-p1
31              if (p3.eq.0.0) then
32                  iii2(i) = 1
33              else
34                  iii2(i) = 1 + nint ((p2-p1)/p3)
35              end if
36              ppp1(i) = p1
37              ppp2(i) = p2
38              ppp3(i) = (p2-p1) /float (max (1, iii2(i)-1))
39  c           psav(i) = diefcn(m)                      ! n+ik
40              iptu(i) = m
41          end if
42      end do       ! mlmnts
43
44      if (mfilmm.ne.0) then
45          do m=1,mfilmm                                ! z ' thickness
46              if (lvaryz(m) .eq. 1) then
47                  i = i+1
48                  read  (idat,  *)     j, p1, p2, p3
49                  write (iout,102)  i,    p1, p2, p3, j, '(z   )'
50
```

```
51                      if ( j.ne.m   .or. p2.lt.p1  .or.
52       &                  p1.lt.0.0 .or. p2.lt.0.0 .or. p3.lt.0.0) then
53                          write (iout,103)
54                          stop
55                      end if
56                      if (p3.eq.0.0) p3=p2-p1
57                      if (p3.eq.0.0) then
58                          iii2(i) = 1
59                      else
60                          iii2(i) = 1 + nint ((p2-p1)/p3)
61                      end if
62                      ppp1(i) = p1
63                      ppp2(i) = p2
64                      ppp3(i) = (p2-p1) /float (max (1, iii2(i)-1))
65   c                  psav(i) = widths(m)
66                      iptu(i) = mlmnts+m
67                  end if
68              end do   ! mfilmm
69          end if
70          mvary = i    ! depth of do-loop nest
71
72   c      ------------------------------------------------------
73   c      Test and Ensure that we consider only:     1D or 2D   plots
74
75          if (mvary.eq.0 .or. mvary.gt.2) then
76              write (iout,211)
77              stop
78          end if
79   c      ------------------------------------------------------
80   c      Discern breakpoint
81
82          read  (idat,*,err=11,end=11)  mvarii, kt, kts, old
83          write (iout,121)
84          write (iout,111)              mvarii, kt, kts, old
85          if (mvarii.ne.mvary .or. kt.lt.1 .or. kts.lt.1 .or.
86         &                                    old.le.0.0   ) goto 11
87          do i=1,mvary
88              read  (idat,*,err=11,end=11)  j, iiii(i), pppp(i), psav(i)
89              write (iout,112)              j, iiii(i), pppp(i), psav(i)
90
91              if (iiii(i) .lt. 1        .or.   j.ne.i  .or.
92       &          iiii(i) .gt. iii2(i)                .or.
93       &          pppp(i) .lt. ppp1(i)-ppp3(i)*0.001  .or.
94       &          pppp(i) .gt. ppp2(i)+ppp3(i)*0.001  .or.
95       &          psav(i) .lt. ppp1(i)-ppp3(i)*0.001  .or.
96       &          psav(i) .gt. ppp2(i)+ppp3(i)*0.001     ) then
97                  write (iout,113)
98                  stop                ! goto 11
99              end if
100
101             if (iiii(i).lt.iii2(i)  .or.  iii2(i).eq.1) then
102                 ppppi = ppp1(i)+ppp3(i)*float(iiii(i) -1)    ! [,)
103             else
```

```
104               ppppi = ppp2(i)                              !   ]
105           end if
106           if (abs(pppp(i)-ppppi) .gt. ppp3(i)*0.01) then
107               write (iout,113)
108               stop                ! goto 11
109           end if
110       end do
111       write (iout,114)
112       iiistp = 1
113       iii = mvary
114       it1 = iftime (i)                        ! start clock
115       goto 2                                  ! start at interior, resume
116    11 continue                                ! start at beginning
117       write (iout,115)
118       it1 = iftime (i)                        ! start clock
119
120 c     ---------------------------------------
121 c     Formulate header card for the plot utility
122 c     Plot format:   nx,ny,nu /(i,x,y,u)  =====>  index backwards
123
124       m = 1                                             ! (v,g)
125       kt = mvary+1
126       if (mvary .eq. 1) then
127           write (iplt,212)  (iii2(kt-j), j=1,mvary), m     ! header card
128       else
129           write (iplt,213)  (iii2(kt-j), j=1,mvary), m     ! header card
130       end if
131 c     =====================================================
132 c     Emulate, initialize, activate:   the nest of do-loops
133
134       do i=1,mvary
135           iiii(i) = 0                         ! initialization:   i1-i3
136       end do
137       kt = 0
138       iiistp = 1                              ! do-loop increment
139       iii = 0                                 ! index of:  nested do-variables.
140     1 iii = iii+1                             ! index of:  iii-th    nested do.
141       if (iii .gt. mvary) goto 3
142     2 iiii(iii) = iiii(iii) + iiistp          ! update do-loop variable
143       if (iiii(iii) .lt. iii2(iii)) then      ! test upper limit
144           pppp(iii) = ppp1(iii) + ppp3(iii)*float (iiii(iii) -1)
145           goto 1
146       else if (iiii(iii) .eq. iii2(iii)) then
147           if (iii2(iii) .eq. 1) then
148               pppp(iii) = ppp1(iii)
149           else
150               pppp(iii) = ppp2(iii)
151           end if
152           goto 1
153       end if
154       iiii(iii) = 0                           ! reset inner do,  i1-i3
155       iii = iii-1                             ! backup one do-level
156       if (iii .eq. 0) goto 4                  ! escape nest of do-loops
```

```
157         goto 2
158       3 iii = iii-1                          ! level of inner-most nested do.
159   c     ------------------------------------ ! begin processing body
160
161         kt = kt+1                              ! simple counter
162         do i=1,mvary                           ! re-set model parameters
163             m = iptu(i)
164             if (m.le.mlmnts) then
165                 diefcn(m) = pppp(i)            ! n+ik
166             else
167                 m = m-mlmnts
168                 widths(m) = pppp(i)            ! z
169             end if
170         end do       ! vary
171
172   c*    call zoom2 (ppp1, ppp2, ppp3)          ! iterate ' steepest descent
173
174         llnorm = .false.
175         call asmbl
176         call norm (meqns,bb,bnorm,1)           ! retain norm of residual
177         bnorm = bnorm*raddeg                   ! degrees
178   c     ---------------------------------------
179
180         m = mvary+1                            ! backwards
181         write (iplt, 214)  kt, (pppp(m-j), j=1,mvary), bnorm
182
183   c     ---------------------------------------
184         if (kt.eq.1) then
185             kts = 1                            ! density of states
186             old = bnorm
187             do i=1,mvary                       ! retain model parameters
188                 m = iptu(i)
189                 if (m.le.mlmnts) then
190                     psav(i) = diefcn(m)
191                 else
192                     m = m-mlmnts
193                     psav(i) = widths(m)
194                 end if
195             end do        ! vary
196         else if (bnorm.lt.old) then
197             kts = 1
198             old = bnorm
199             do i=1,mvary
200                 m = iptu(i)
201                 if (m.le.mlmnts) then
202                     psav(i) = diefcn(m)
203                 else
204                     m = m-mlmnts
205                     psav(i) = widths(m)
206                 end if
207             end do        ! vary
208         else if (bnorm.eq.old) then            ! retain density of states
209             kts = kts+1                        !        along the minimum
```

```
210         end if
211
212         it2 = iftime (i)                        ! CPU clock
213         tim = float (it2-it1) /6.0E4            ! CPU minutes elapsed
214         if (tim .gt. 15.0) then                 ! breakpoint
215             it1 = it2
216             open (unit=isout, file='x.sout', status='unknown')
217             write (isout,111) mvary, kt, kts, old
218             do i=1,mvary
219                 write (isout,112) i, iiii(i), pppp(i), psav(i)
220             end do
221             call time (bufft)
222             call date (buffd)
223             write (isout,116)  bufft, buffd   ! convenience
224             close (unit=isout)
225         end if
226 c       ------------------------------------ ! finish processing body
227         goto 2                                ! nested do-loop:   iiii
228       4 continue                              ! last line of do-nest,  iiii
229 c       ====================================
230         lbdry = .false.
231
232         write (iout,104) kt, kts, old         ! density of states along minimum
233         write (iout,105)
234         do i=1,mvary
235             local = (psav(i) .le. ppp1(i)+ppp3(i)*0.001) .or.
236      &              (psav(i) .ge. ppp2(i)-ppp3(i)*0.001)
237 c*          local = local .and. (iii2(i).ne.1)
238             if (local)  lbdry=.true.
239             m = iptu(i)
240             if (m.le.mlmnts) then             ! n+ik " diefcn
241                 diefcn(m) = psav(i)
242                 if (local) then
243                     write (iout,106)  i, psav(i), m, '(n+ik)', blank
244                 else
245                     write (iout,106)  i, psav(i), m, '(n+ik)'
246                 end if
247             else                              ! z    " widths
248                 m = m-mlmnts
249                 widths(m) = psav(i)
250                 if (local) then
251                     write (iout,106)  i, psav(i), m, '(z   )', blank
252                 else
253                     write (iout,106)  i, psav(i), m, '(z   )'
254                 end if
255             end if
256         end do       ! vary
257         if (lbdry)  write (iout,107)
258         write (iout,108)
259
260         return
261
262    101 format (/' Scan a grid of model parameters.    (scan2g)'
```

```
263       &        /' Grid info:',  4x,  'initial,',  6x,  'final,',
264       &                                           5x, 'increment' )
265     102 format ( 3x,  i4,  ') ',  3f13.4,
266       &        '   ' for:', i5, ',  ', a6)
267
268     103 format (/' scan2g,     error:   inconsistent data input')
269     104 format (/' number of grid points scanned, kt  = ', i10
270       &        /'  population along the minimum, kts = ', i10
271       &        /'              norm of residual, |g| = ', 1pe13.5,
272       &                         '   (degrees) ')
273     105 format (/' Model parameter value along the minimum:')
274     106 format (1x,i4, ')',  1p1e15.6,  '   ' for:',  i5,  ',  ',  a6,
275       &                           a1,   '  ' boundary')
276     107 format (/' Note that the minimum point is near a boundary.')
277     108 format (/' output written to:    x.plot')
278
279     111 format (1x, i4,1x, 2i10, 1p1e15.6, ' ' mvary, kt, kts, b(min)')
280     112 format (1x, 2(i4,1x),    1p2e15.6, ' ' i, ip, p, p(min)')
281     113 format (/' The attempted restart was NOT acceptable.')
282     114 format (/' The attempted restart was     acceptable.')
283     115 format (/' Note:   NO attempt was made to restart.')
284     116 format ( ' wall clock:    time = ', a8, 1x, ' ' hh:mm:ss '
285       &       / '                date = ', a9,     ' ' dd-mmm-yy' )
286
287     121 format (' ')
288
289     211 format (/' ... oops,   we consider only:   1D or 2D  plots, '
290       &        /'             we allow only:      mvary '  1 or 2  ')
291     212 format ( 1x, 2i4, ' ' mx,mu / (i, x, u(x)'|g|) ')
292     213 format ( 1x, 3i4, ' ' mx(2),mx(1), mu /(i, x(2),x(1), u'|g|)')
293     214 format ( 1x,  i6, 1p3e15.6)
294         end
```

6.2.8 SCAN3.FOR

```
      subroutine scan3
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      include  'filmss.'
      include  'scancc.'

      integer   iipt(niii), jjpt(niii)
      integer   jjjj(niii), jjj2(niii)
      real      qqqq(niii), qqq1(niii), qqq2(niii), qqq3(niii)
      real      qsav(niii)

      logical   lbdry, local, lcoupl
      character blank, comma
      data      blank /' '/,   comma/','/

c     Solve the forward scattering problem over a grid of model parameters.
c     Note:    the model parameters involve partitions of:
c              [ froz / froz (grid) / vary (grid, continuous) ]
c              [froz --->    <--- vary]


      raddeg = 180.0 / 3.14159265               ! degrees <--- radians

c     ================================================================
c     Input grid info for scanning the model parameters

      write (iout,111)
      ki = 0                                    ! simple counter
      k = 0                                     ! froz, grid
      i = 0                                     ! vary, grid
      do m=1,mlmnts                             ! (n+ik)
          if (lvaryl(m) .ne. 0) then            ! grid
              ki = ki+1
              read  (idat,  *)      j, p1, p2, p3
              write (iout,112)  ki,    p1, p2, p3, j, '(n+ik)'

              if ( j.ne.m   .or. p2.lt.p1  .or.
     &            p1.lt.0.0 .or. p2.lt.0.0 .or. p3.lt.0.0) then
                  write (iout,113)
                  stop
              end if
              if (lvaryl(m) .eq. 1) then        ! vary
                  i = i+1
                  j = niii+1-i                  ! backward
              else                              ! froz
                  k = k+1
                  j = k
              end if

```

```
51              if (p3.eq.0.0) p3=p2-p1
52              if (p3.eq.0.0) then
53                  iii2(j) = 1
54              else
55                  iii2(j) = 1 + nint ((p2-p1)/p3)
56              end if
57              ppp1(j) = p1
58              ppp2(j) = p2
59              ppp3(j) = (p2-p1) /float (max (1, iii2(j)-1))
60  c*          psav(j) = diefcn(m)                ! n+ik
61              iipt(j) = m
62              jjpt(m) = j
63          end if
64      end do       ! mlmnts
65
66      if (mfilmm.ne.0) then
67          do m=1,mfilmm                          ! z ' thickness
68              if (lvaryz(m) .ne. 0) then
69                  ki = ki+1
70                  read  (idat,  *)      j, p1, p2, p3
71                  write (iout,112)  ki,    p1, p2, p3, j, '(z   )'
72
73                  if ( j.ne.m   .or. p2.lt.p1  .or.
74     &                 p1.lt.0.0 .or. p2.lt.0.0 .or. p3.lt.0.0) then
75                      write (iout,113)
76                      stop
77                  end if
78                  if (lvaryz(m) .eq. 1) then     ! vary
79                      i = i+1
80                      j = niii+1-i               ! backwards
81                  else                           ! froz
82                      k = k+1
83                      j = k
84                  end if
85
86                  if (p3.eq.0.0) p3=p2-p1
87                  if (p3.eq.0.0) then
88                      iii2(j) = 1
89                  else
90                      iii2(j) = 1 + nint ((p2-p1)/p3)
91                  end if
92                  ppp1(j) = p1
93                  ppp2(j) = p2
94                  ppp3(j) = (p2-p1) /float (max (1, iii2(j)-1))
95  c*              psav(j) = widths(m)
96                  iipt(j) = m+mlmnts
97                  jjpt(m+mlmnts) = j
98              end if
99          end do   ! mfilmm
100     end if
101
102     mfrozz = k        ! depth of do-loop nest/grid:   froz
103     mvaryy = i        ! depth of do-loop nest/grid:   vary
```

```
104         muv = mfrozz+mvaryy
105
106   c     ---------------------------------------------------------------
107   c     Output range of do-loops for the grid:     froz, vary
108
109         if (mfrozz .eq. 0) then
110             write (iout,114)  mfrozz
111             stop
112         else
113             write (iout,121)  'froz'
114             do i=1,mfrozz
115                 m = iipt(i)
116                 if (m.le.mlmnts) then
117                     write (iout,122)  i, iii2(i), m, '(n+ik)'
118                 else
119                     m = m-mlmnts
120                     write (iout,122)  i, iii2(i), m, '(z   )'
121                 end if
122             end do
123         end if
124
125         if (mvaryy .eq. 0) then
126             write (iout,115)  mvaryy
127             stop
128         else
129             write (iout,121)  'vary'
130             do i=1,mvaryy
131                 j = niii+1-i                          ! backwards
132                 m = iipt(j)
133                 if (m.le.mlmnts) then
134                     write (iout,122)  i, iii2(j), m, '(n+ik)'
135                 else
136                     m = m-mlmnts
137                     write (iout,122)  i, iii2(j), m, '(z   )'
138                 end if
139             end do
140         end if
141
142   c     ===================================================================
143   c     Discern whether:  VARY model parameters are coupled between samples.
144   c     Note here that the couplings:
145   c     1)  between samples are suppressed, while those
146   c     2)  within the sample are fully accounted.
147
148         mm = mlmnts+mfilmm
149         do m=1,mm                                     ! initialize template
150             iptw(m) = 0                               ! cumulative
151             iptv(m) = 0                               ! individual
152         end do
153
154         iws = 0
155         mws = 0
156         do is=1,msampl
```

```
157             mwave = nnwave(is)
158             mfilm = nnfilm(is)
159             mfilms = mfilm+1
160             do iw=1,mwave
161                 iws = iws+1
162                 iwave = iiwave(iws)
163                 mbien = nnbent(iws)
164                 do m=1,mfilms
165                     if (m.ne.mfilms) then
166                         mws = mws+1
167                         iz = iifilm(mws)
168                         if (lvaryz(iz) .eq. 1) then
169                             j = mlmnts+iz
170                             iptv(j) = 1             ! vary
171                         end if
172                     end if
173                     do ink=1,2
174                         mws = mws+1
175                         nk = iifilm(mws)
176                         if (lvaryl(nk) .eq. 1) then
177                             iptv(nk) = 1            ! vary
178                         end if
179                     end do      ! n+ik
180                 end do         ! films
181             end do             ! waves
182
183             do m=1,mm                               ! overlap
184                 iptw(m) = iptw(m)+iptv(m)           ! cumulative
185                 iptv(m) = 0                         ! individual ' initialize
186             end do
187         end do        ! samples
188
189         k = 0                                       ! count couplings
190         do m=1,mm
191             if (iptw(m).gt.1)  k=k+1
192         end do
193
194         if (k.eq.0) then                            ! no vary couplings
195             lcoupl = .false.                        !    between samples
196         else                                        ! couplings/overlap
197             lcoupl = .true.
198             write (iout,131)  k
199             do m=1,mm
200                 if (iptw(m).gt.1) then
201                     if (m.le. mlmnts) then
202                         write (iout,132)  m, '(n+ik)'
203                     else
204                         j = m-mlmnts
205                         write (iout,132)  j, '(z   )'
206                     end if
207                 end if
208             end do
209             write (iout,133)
```

```
c*        stop
       end if

c      ================================================================
c      Discern breakpoint,                          [froz --->   <--- vary]

       write (iout,141)
       read  (idat,*,err=11,end=11)  kk, ii, ktu, bmin
       write (iout,142)              kk, ii, ktu, bmin
       if (kk.ne.mfrozz .or. ii.ne.mvaryy .or.
      &    ktu.lt.1 .or. bmin.lt.0.0         )  goto 11

       do i=1,mfrozz
           j = i
           read  (idat,*,err=11,end=11)  ii, iiii(j),pppp(j),psav(j),im
           write (iout,143)              ii, iiii(j),pppp(j),psav(j),im
           m = iipt(j)
           if (m.gt.mlmnts)  m=m-mlmnts

           if (     ii .ne. i                      .or.
      &             im .ne. m                      .or.
      &        iiii(j) .lt. 1                      .or.
      &        iiii(j) .gt. iii2(j)                .or.
      &        pppp(j) .lt. ppp1(j)-ppp3(j)*0.001 .or.
      &        pppp(j) .gt. ppp2(j)+ppp3(j)*0.001 .or.
      &        psav(j) .lt. ppp1(j)-ppp3(j)*0.001 .or.
      &        psav(j) .gt. ppp2(j)+ppp3(j)*0.001     ) then
               write (iout,146)
               stop           ! goto 11
           end if

           if (iiii(j).lt.iii2(j)  .or.  iii2(j).eq.1) then
               ppppi = ppp1(j) + ppp3(j)*float (iiii(j) -1)      ! [,)
           else
               ppppi = ppp2(j)                                   !   ]
           end if
           if (abs(pppp(j)-ppppi) .gt. ppp3(j)*0.001) then
               write (iout,146)
               stop           ! goto 11
           end if
       end do                 ! mfrozz

       if (mvaryy.ne.0) then
           do i=1,mvaryy
               j = niii+1-i                        ! backwards
               read  (idat,*,err=11,end=11)  ii, psav(j), im
               write (iout,144)              ii, psav(j), im
               m = iipt(j)
               if (m.gt.mlmnts)  m=m-mlmnts

               if (     ii .ne. i+mfrozz             .or.
      &                 im .ne. m                    .or.
      &            psav(j) .lt. ppp1(j)-ppp3(j)*0.001 .or.
```

```
263       &            psav(j) .gt. ppp2(j)+ppp3(j)*0.001     ) then
264                    write (iout,146)
265                    stop       ! goto 11
266                end if
267            end do
268        end if
269
270        write (iout,147)
271        iiistp = 1
272        iii = mfrozz
273        it1 = iftime (i)                        ! start clock
274        goto 22                                 ! re-start
275
276     11 continue
277        write (iout,148)                        ! start from beginning
278        it1 = iftime (i)                        ! start clock
279
280  c     ================================================================
281  c     Emulate, initialize, activate:    nest of do-loops    (froz)
282
283        ktu = 0                                 ! index/counter
284        do i=1,mfrozz
285            iiii(i) = 0                         ! initialization:   i1-i3
286        end do
287        iiistp = 1                              ! do-loop increment
288        iii = 0                                 ! index of:  nested do-variables.
289     21 iii = iii+1                             ! index of:  iii-th    nested do.
290        if (iii .gt. mfrozz) goto 23
291     22 iiii(iii) = iiii(iii) + iiistp          ! update do-loop variable
292        if (iiii(iii) .lt. iii2(iii)) then      ! test upper limit
293            pppp(iii) = ppp1(iii) + ppp3(iii)*float (iiii(iii) -1)
294            goto 21
295        else if (iiii(iii) .eq. iii2(iii)) then
296            if (iii2(iii) .eq. 1) then
297                pppp(iii) = ppp1(iii)
298            else
299                pppp(iii) = ppp2(iii)
300            end if
301            goto 21
302        end if
303        iiii(iii) = 0                           ! reset inner do,  i1-i3
304        iii = iii-1                             ! backup one do-level
305        if (iii .eq. 0) goto 24                 ! escape nest of do-loops
306        goto 22
307     23 iii = iii-1                             ! level of inner-most nested do.
308  c     ---------------------------------- ! begin processing body
309
310        ktu = ktu+1                             ! simple counter
311        do i=1,mfrozz                           ! re-set model parameters
312            m = iipt(i)
313            if (m.le.mlmnts) then
314                diefcn(m) = pppp(i)             ! n+ik
315            else
```

```
316                 m = m-mlmnts
317                 widths(m) = pppp(i)           ! z
318             end if
319         end do
320
321         isum = 0                              ! sum:   meqns
322         sumi = 0.0                            ! sum:   |g|
323 c       ---------------------------------------------------------------
324 c       Output model parameters associated with grid:   froz
325
326         write (iout,151)
327         write (iout,152)  ktu
328         do i=1,mfrozz
329             m = iipt(i)
330             if (m.le.mlmnts) then
331                 write (iout,153)  i, iiii(i), pppp(i), m, '(n+ik)'
332             else
333                 m = m-mlmnts
334                 write (iout,153)  i, iiii(i), pppp(i), m, '(z   )'
335             end if
336         end do
337 c       ---------------------------------------------------------------
338 c       Assume that VARY model parameters are uncoupled between samples.
339 c       Each sample is separately scanned across its own VARY parameters.
340
341         lbdry = .false.
342         ngl = 0
343         iws = 0
344         mws = 0
345         iaws = 0
346         iraws = 0
347         do is=1,msampl
348 c           -----------------------------------------------------------
349             isx = is                                  ! save
350             nglx = ngl
351             iwsx = iws
352             mwsx = mws
353             iawsx = iaws
354             irawsx = iraws
355
356             do m=1,mm                                 ! localize:   vary
357                 iptw(m) = 0
358             end do
359
360             mwave = nnwave(is)
361             mfilm = nnfilm(is)
362             mfilms = mfilm+1
363             do iw=1,mwave
364                 iws = iws+1
365                 iwave = iiwave(iws)
366                 mbien = nnbent(iws)
367                 do m=1,mfilms
368                     if (m.ne.mfilms) then
```

```
                        mws = mws+1
                        iz = iifilm(mws)
                        zzz(m) = widths(iz)
                        if (lvaryz(iz) .eq. 1) then
                            iptw(mlmnts+iz) = 1
                        end if
                    end if
                    do ink=1,2
                        mws = mws+1
                        nk = iifilm(mws)
                        if (lvaryl(nk) .eq. 1) then
                            iptw(nk) = 1
                        end if
                    end do
                end do          ! films

                do mbn=1,mbien                      ! pointers
                    iaws = iaws+1
                    mrpeat = nnpeat(iaws)
                    do irpeat=1,mrpeat
                        iraws = iraws+1
                        mangl = mangle(iraws)
                        ngl = ngl+mangl             ! multiple angles
                    end do      ! repeat
                end do         ! ambient
            end do            ! wave
c           ------------------------------------ ! arrang,   localized
            i = 0
            k = 0
            do m=1,mm
                if (iptw(m) .eq. 1) then            ! vary, local
                    i = i+1
                    iptu(i) = m
                    iptv(m) = i
                    iptw(m) = 0
                else                                ! froz, elsewise
                    k = k+1
                    km = mm+1-k
                    iptu(km) = m
                    iptv( m) = km
                end if
            end do
            mvary = i
            mfroz = k
c           ----------------------------------------
            if (mvary.eq.0) then                    ! everything is frozen
c*              llnorm = .false.
c*              call asmbl3 (isx, iwsx, mwsx, iawsx, irawsx, nglx)
c*              call norm (meqns, bb, old,1)        ! retain norm of residual
c*              old = old*raddeg                    ! degrees
c*              goto  34

                write (iout,161)  mvary, is
```

```
422             stop
423           end if
424   c       ------------------------------------ ! do-nest:   local
425           do i=1,mvary
426               m = iptu(i)                      ! n+ik, iz+mlmnts <---
427               j = jjpt(m)
428               jjj2(i) = iii2(j)
429               qqq1(i) = ppp1(j)
430               qqq2(i) = ppp2(j)
431               qqq3(i) = ppp3(j)
432           end do
433
434   c       ============================================================
435   c       Emulate, initialize, activate:    nest of do-loops   (vary)
436
437           ktv = 0                                  ! index/counter
438           do i=1,mvary
439               jjjj(i) = 0
440           end do
441   c       iiistp = 1
442           jjj = 0                        ! index of:  nested do-variables.
443      31   jjj = jjj+1                    ! index of:  iii-th    nested do.
444           if (jjj .gt. mvary) goto 33
445      32   jjjj(jjj) = jjjj(jjj) + iiistp     ! update do-loop variable
446           if (jjjj(jjj) .lt. jjj2(jjj)) then ! test upper limit
447               qqqq(jjj) = qqq1(jjj) + qqq3(jjj)*float (jjjj(jjj) -1)
448               goto 31
449           else if (jjjj(jjj) .eq. jjj2(jjj)) then
450               if (jjj2(jjj) .eq. 1) then
451                   qqqq(jjj) = qqq1(jjj)
452               else
453                   qqqq(jjj) = qqq2(jjj)
454               end if
455               goto 31
456           end if
457           jjjj(jjj) = 0                      ! reset inner do,  i1-i3
458           jjj = jjj-1                        ! backup one do-level
459           if (jjj .eq. 0) goto 34            ! escape nest of do-loops
460           goto 32
461      33   jjj = jjj-1                        ! level of inner-most nested do.
462   c       -------------------------------- ! begin processing body
463
464           ktv = ktv+1                        ! simple counter
465           do i=1,mvary                       ! reset model parameters
466               m = iptu(i)
467               if (m.le.mlmnts) then
468                   diefcn(m) = qqqq(i)
469               else
470                   m = m-mlmnts
471                   widths(m) = qqqq(i)
472               end if
473           end do
474
```

```
475           call zoom3 (qqq1, qqq2, qqq3,          ! damped least squares
476      &                isx, iwsx, mwsx, iawsx, irawsx, nglx)
477           llnorm = .false.
478           call asmbl3 (isx, iwsx, mwsx, iawsx, irawsx, nglx)
479           call norm (meqns, bb, bnorm, 1)        ! retain norm of residual
480           bnorm = bnorm*raddeg                   ! degrees
481
482           if (ktv.eq.1) then                     ! initial
483               ktvmin = 1
484               old = bnorm
485               do i=1,mvary
486                   m = iptu(i)
487                   j = jjpt(m)
488                   if (m.le.mlmnts) then
489                       qsav(j) = diefcn(m)
490                   else
491                       m = m-mlmnts
492                       qsav(j) = widths(m)
493                   end if
494               end do
495           else if (bnorm.lt.old) then
496               ktvmin = 1
497               old = bnorm
498               do i=1,mvary
499                   m = iptu(i)
500                   j = jjpt(m)
501                   if (m.le.mlmnts) then
502                       qsav(j) = diefcn(m)
503                   else
504                       m = m-mlmnts
505                       qsav(j) = widths(m)
506                   end if
507               end do
508           else if (bnorm.eq.old) then       ! along minimum
509               ktvmin = ktvmin+1             ! population
510           end if
511
512   c       ---------------------------------! finish processing body
513           goto 32                           ! nested do-loop:   jjjj
514       34  continue                          ! last line of do-nest
515   c       ============================================================
516   c       Output results:   VARY   model parameters of the sample.
517
518           if (is.eq.1) then
519               write (iout,171)
520           end if
521           write (iout,172)  is, old
522
523           if (mvary.ne.0) then
524           do i=1,mvary
525               m = iptu(i)
526               j = jjpt(m)
527               local = (qsav(j) .le. ppp1(j)+ppp3(j)*0.001) .or.
```

```
528       &                (qsav(j) .ge. ppp2(j)-ppp3(j)*0.001)
529   c*            local = local .and. (iii2(j).ne.1)
530                 if (local)  lbdry=.true.
531
532                 if (m.le.mlmnts) then
533                     if (local) then
534                         write (iout,173)  qsav(j), m, '(n+ik)', blank
535                     else
536                         write (iout,173)  qsav(j), m, '(n+ik)'
537                     end if
538                 else
539                     m = m-mlmnts
540                     if (local) then
541                         write (iout,173)  qsav(j), m, '(z   )', blank
542                     else
543                         write (iout,173)  qsav(j), m, '(z   )'
544                     end if
545                 end if
546             end do
547             end if
548
549             isum = isum + meqns                                  !  meqns
550             sumi = sumi + old*old*float (meqns)                  !  |g|
551
552         end do       ! samples
553   c     ------------------------------------------------------------------
554   c     Note:   summing   |b(s)|   from each sample   ===>
555   c             NO coupling between samples by VARY parameters.
556
557         sumi = sqrt (sumi/float (isum))       !  |g|  without coupling
558         if (lbdry) then
559             write (iout,174)  sumi, comma
560         else
561             write (iout,174)  sumi
562         end if
563
564   c     ---------------------------------------
565   c     If coupling exists between samples via VARY model parameters,
566   c     the calculation of the cumulative residual is meaningless,
567   c     because the coupled VARY parameter is NOT necessarily held common.
568
569   c     Discern the cumulative residual along the minimum.
570   c     Restore the VARY model parameters associated with the minimum.
571
572         if (mvaryy.ne.0) then
573             do i=1,mvaryy
574                 j = niii+1-i                  ! backwards
575                 m = iipt(j)
576                 if (m.le.mlmnts) then
577                     diefcn(m) = qsav(j)
578                 else
579                     m = m-mlmnts
580                     widths(m) = qsav(j)
```

```
581                 end if
582             end do
583         end if
584
585 c       ------------------------------------
586         llnorm = .false.
587         call asmbl
588         call norm (meqns,bb,bnorm,1)          ! retain norm of residual
589         bnorm = bnorm*raddeg                  ! degrees
590
591         if (ktu.eq.1) then
592             ktumin = 1                        ! density of states
593             bmin = bnorm
594             do i=1,muv                        ! retain model parameters
595                 if (i.le.mfrozz) then
596                     j = i
597                 else
598                     j = niii+1 - (i-mfrozz)
599                 end if
600                 m = iipt(j)
601                 if (m.le.mlmnts) then
602                     psav(j) = diefcn(m)
603                 else
604                     m = m-mlmnts
605                     psav(j) = widths(m)
606                 end if
607             end do
608         else if (bnorm.lt.bmin) then
609             ktumin = 1
610             bmin = bnorm
611             do i=1,muv                        ! retain model parameters
612                 if (i.le.mfrozz) then
613                     j = i
614                 else
615                     j = niii+1 - (i-mfrozz)
616                 end if
617                 m = iipt(j)
618                 if (m.le.mlmnts) then
619                     psav(j) = diefcn(m)
620                 else
621                     m = m-mlmnts
622                     psav(j) = widths(m)
623                 end if
624             end do
625         else if (bnorm.eq.bmin) then          ! retain density of states
626             ktumin = ktumin+1                 !        along the minimum
627         end if
628
629 c       ------------------------------------------------------------
630 c       Breakpoint
631
632         it2 = iftime (i)                      ! CPU clock
633         tim = float (it2-it1) /6.0E4          ! CPU minutes elapsed
```

```
634         if (tim .gt. 15.0) then                ! breakpoint
635             it1 = it2
636
637             close (unit=iout)
638             open (unit=iout,file='x.out', status='old', access='append')
639
640             open (unit=isout, file='x.sout', status='unknown')
641             write (isout,142)  mfrozz, mvaryy, ktu, bmin
642             do i=1,mfrozz
643                 m = iipt(i)
644                 if (m.le.mlmnts) then
645                     write (isout,143)  i, iiii(i), pppp(i), psav(i), m,
646       &                                          ',  (n+ik),  froz'
647                 else
648                     m = m-mlmnts
649                     write (isout,143)  i, iiii(i), pppp(i), psav(i), m,
650       &                                          ',  (z   ),  froz'
651                 end if
652             end do
653             if (mvaryy.ne.0) then
654                 do i=1,mvaryy
655                     ii = i+mfrozz              ! convenience counter
656                     j = niii+1-i               ! backwards
657                     m = iipt(j)
658                     if (m.le.mlmnts) then
659                         write (isout,144)  ii, psav(j), m,
660       &                                       ',  (n+ik),  vary'
661                     else
662                         m = m-mlmnts
663                         write (isout,144)  ii, psav(j), m,
664       &                                       ',  (z   ),  vary'
665                     end if
666                 end do
667             end if
668
669             call time (bufft)
670             call date (buffd)
671             write (isout,200)  bufft, buffd   ! convenience
672             close (unit=isout)
673         end if
674   c     ---------------------------------- ! finish processing body
675         goto 22                            ! nested do-loop:   iiii
676        24 continue                         ! last line of do-nest
677   c     ================================================================
678
679         write (iout,151)
680         write (iout,181)  ktu, ktumin, bmin
681         lbdry = .false.
682
683         write (iout,182)  'froz'
684         do i=1,mfrozz
685             j = i
686             m = iipt(i)
```

```
687           local = (psav(j) .le. ppp1(j)+ppp3(j)*0.001) .or.
688      &            (psav(j) .ge. ppp2(j)-ppp3(j)*0.001)
689  c*          local = local .and. (iii2(j).ne.1)
690           if (local)  lbdry=.true.
691           if (m.le.mlmnts) then
692               diefcn(m) = psav(j)
693               if (local) then
694                   write (iout,183)  psav(j), m, '(n+ik)', blank
695               else
696                   write (iout,183)  psav(j), m, '(n+ik)'
697               end if
698           else
699               m = m-mlmnts
700               widths(m) = psav(j)
701               if (local) then
702                   write (iout,183)  psav(j), m, '(z   )', blank
703               else
704                   write (iout,183)  psav(j), m, '(z   )'
705               end if
706           end if
707       end do
708
709       if (mvaryy .ne. 0) then
710           write (iout,182)  'vary'
711           do i=1,mvaryy
712               j = niii+1-i
713               m = iipt(j)
714               local = (psav(j) .le. ppp1(j)+ppp3(j)*0.001) .or.
715      &                (psav(j) .ge. ppp2(j)-ppp3(j)*0.001)
716  c*           local = local .and. (iii2(j).ne.1)
717               if (local)  lbdry=.true.
718               if (m.le.mlmnts) then
719                   diefcn(m) = psav(j)
720                   if (local) then
721                       write (iout,183)  psav(j), m, '(n+ik)', blank
722                   else
723                       write (iout,183)  psav(j), m, '(n+ik)'
724                   end if
725               else
726                   m = m-mlmnts
727                   widths(m) = psav(j)
728                   if (local) then
729                       write (iout,183)  psav(j), m, '(z   )', blank
730                   else
731                       write (iout,183)  psav(j), m, '(z   )'
732                   end if
733               end if
734           end do
735       end if
736
737       if (lbdry )  write (iout,184)
738       if (lcoupl)  write (iout,133)
739
```

```
740       return
741
742   111 format (/' Scan a grid of model parameters.'
743      &        /' Grid info:',  4x,  'initial,',  6x,  'final,',
744      &                                            5x, 'increment' )
745   112 format ( 1x,  i4,  ')',  3x,  3f13.4,
746      &         '   " for:', i5, ',  ', a6)
747   113 format (/' oops,   inconsistent data input')
748   114 format (/' oops,   number of model parameters in the grid: '
749      &        /'         froz " ', i4
750      &        /'         use alternate option:     ichoic " 3')
751   115 format (/' oops,   number of model parameters in the grid: '
752      &        /'         vary " ', i4  )
753
754   121 format (/' Range of do-loops:   ', a4)
755   122 format ( 1x, i4, ')', i7, 3x, 'grid points for:', i6, ',  ', a6)
756
757   131 format (/' Couplings between samples involve: ', i3,
758      &         '   distinct model parameters')
759   132 format (9x, i4, ',', 3x, a6)
760   133 format (/' Note:    The existence of coupling between samples'
761      &        /'          due to the  VARY  model parameters -- '
762      &   /' induces an unusual interpretation unto the residual  |g|,'
763      &   /' because, while providing coupling,  they are NOT '
764      &   /' necessarily of similar value on different samples.' /)
765
766   141 format (/' Attempt to read breakpoint information')
767   142 format (1x, 3i5,      1p1e15.6,    25x, ' " mfrozz, mvaryy, ',
768      &                                              'ktu, bmin')
769   143 format (1x, 2i5,  5x, 1p2e15.6, i5, 5x, ' " i, ip, p, psav, m',
770      &                                                    a16 )
771   144 format (1x,  i5, 10x, 1p1e15.6, i5, 5x, ' " i,        psav, m',
772      &                                                    a16 )
773
774   146 format (/' The attempted restart was NOT acceptable.')
775   147 format (/' The attempted restart was     acceptable.')
776   148 format (/' Note:   NO attempt was made to restart.')
777
778   151 format (/ 1x, 17('====')  )
779   152 format (  4x,  '#',  i7,  19x, '" grid:   froz' )
780   153 format (  1x, i4, ')', 2x, i4, 1x, 1p1e15.6,
781      &         '   " model parameter for:', i5, ',  ', a6)
782
783   161 format (/' oops,   mvary =', i4, ' <---------  Note, local'
784      &        /'         sampl =', i4 )
785
786   171 format (  5x, 16('----')
787      &         / 8x,  'sample',        17x, '" grid:   vary')
788   172 format (  8x, i4, 1x, 1p1e15.6,  3x, '" |g|')
789   173 format ( 13x,         1p1e15.6,
790      &         '   " for:', i5, ',  ', a6, a1, 6x, '" boundary')
791   174 format ( 13x, 1p1e15.6, '   " |g| " summed over samples',
792      &                    a1,  ' boundary')
```

```
793
794     181 format (/' number of FROZ grid points scanned, ktu  = ', i10
795        &        /'       population along the minimum, ktum = ', i10
796        &        /'                    norm of residual, |g|  = ',
797        &                                1pe13.6,  '   (degrees) ')
798     182 format (/' Model parameter value along the minimum:', 4x, a4)
799     183 format (   13x, 1p1e15.6, '   ' for:', i5, ',  ', a6,
800        &                        a1,  ' ' boundary')
801     184 format (/' Note that the minimum point is near a boundary.')
802
803
804     200 format ( ' wall clock:    time = ', a8, 1x, ' ' hh:mm:ss '
805        &       / '                date = ', a9,     ' ' dd-mmm-yy' )
806
807         end
```

6.2.9 ZOOM.FOR

```
      subroutine zoom                  ! unconstrained optimization
      include   'iounit.'
      include   'defnit.'
      include   'filmmm.'
      include   'wstack.'
      include   'cgnxx1.'

      data       small  / 1.0E-5 /

      raddeg = 180.0 / 3.14159265
      llnorm = .false.                                ! ASMBL
      call asmbl                                      ! ia,ja,aa,bb
      call norm (meqns, bb, bn, 1)                    ! residual
      call scaljj (meqns,mvary,ia,ja,aa,xx, aats,w,2) ! scale columns, A

      if (bn .eq. 0.0) return
      bn  = bn*raddeg                                 ! degrees <-- radians
      bn1 = bn                                        ! initial
      bn2 = bn                                        ! last
      loop = 0
      niter = mvary*4                                 ! number of iterations
      write (iout,101)
      write (iout,102)  loop, bn

    1 itry = 0                                        ! initialize
      loop = loop+1
      do i=1,mvary                                    ! initialize
          xx(i) = 0.0                                 !     Newton step
      end do
      call cgnl (meqns,mvary, ia,ja,aa,bb,xx,
     &           niter,        u,v,w,  xw,se)
      do i=1,mvary                                    ! account for scaling
          xx(i) = xx(i)/aats(i)                       !   columns in SCALJJ
          se(i) = se(i)/aats(i)
      end do

    2 do i=1,mvary                                    ! update
          j = iptu(i)
          h = xx(i)*0.2                               ! step length
          s = se(i)*0.2                               ! estimated std dev

          if (j.le.mlmnts) then                       ! n+ik
              f = diefcn(j)
              g = uncerl(j)
c*            write (iout,121)  i,j,f,h
              ha = amax1 (s, 0.001, abs(f*0.2))
              ha = amin1 (g, abs(h), ha)
              if (h.lt.0.0)  ha=-ha
              aat(i) = f                              ! retain
              diefcn(j) = amax1 (0.0, f+ha)
```

```
51            else                                      ! z
52                j = j-mlmnts
53                f = widths(j)
54                g = uncerz(j)
55  c*            write (iout,122)  i,j,f,h
56                ha = amax1 (s, 0.01, f*0.2)
57                ha = amin1 (g, abs(h), ha)
58                if (h.lt.0.0) ha=-ha
59                aat(i) = f                            ! retain
60                widths(j) = amax1 (0.0, f+ha)
61            end if
62        end do
63
64        call asmbl                                    ! ia,ja,aa,bb
65        call norm (meqns, bb, bn, 1)                  ! residual
66        call scaljj (meqns,mvary,ia,ja,aa,xx, aats,w,2) ! scale columns, A
67        bn = bn*raddeg                                ! degrees <--- radians
68        total = bn /bn1                               ! total    reduction
69        relat = bn /bn2                               ! relative reduction
70
71        if (total .le. 1.0E-5) then                   ! convergence
72            bn2 = bn                                  !     retain
73            goto 4                                    !     escape
74        end if
75
76        if (relat .le. 0.999) then                    ! converging
77            bn2 = bn                                  !     retain
78            goto 3                                    !     iterate
79        end if
80
81        do i=1,mvary                                  ! reset
82            j = iptu(i)
83            if (j.le.mlmnts) then
84                diefcn(j) = aat(i)
85            else
86                j = j-mlmnts
87                widths(j) = aat(i)
88            end if
89        end do
90
91        if (itry.lt.3) then                           ! try again
92            itry = itry+1
93            do i=1,mvary
94                xx(i) = xx(i)*0.5                     ! reduce stepsize
95            end do
96            goto 2
97        end if
98
99        if (loop .le. 3) then                         ! convenience
100           write (iout,104)
101           if (relat .le. 1.0) then                   ! marginal
102               write (iout,105)
103           else                                       ! divergence
```

```
104               write (iout,106)
105            end if
106         end if
107         goto 5                                    ! escape
108
109       3 write (iout,103)  loop, relat, total, bn
110         goto 1
111       4 write (iout,103)  loop, relat, total, bn
112       5 continue
113
114   c     Output results of iterations involving least squares.
115
116         write (iout,111)                          !  results
117         do i=1,mvary
118            j = iptu(i)
119            if (j.le.mlmnts) then                  !  n+ik
120               write (iout,112)  i, diefcn(j), se(i), j
121            else                                   !  z
122               j = j-mlmnts
123               write (iout,113)  i, widths(j), se(i), j
124            end if
125         end do
126         write (iout,114)  bn1, bn2                !  compare
127
128         return
129
130     101 format (/' zoom:   loop,    ratio of reduction,     |g|'
131        &        /'                  (rel)       (total)         ')
132     102 format ( 8x, i5, 24x, 1p1e12.3, '    (degrees)'  )
133     103 format ( 8x, i5,      1p3e12.3)
134     104 format ( 8x, 'stepsize reduction attempted.')
135     105 format ( 8x, '... slow convergence.')
136     106 format ( 8x, '... divergence.')
137
138     111 format (/' model parameter value along the minimum: ')
139     112 format (1x, i4, ')', f15.4, f13.5, '  for:',
140        &                    i4, ',  (n+ik), estimated uncertainty')
141     113 format (1x, i4, ')', f15.4, f13.5, '  for:',
142        &                    i4, ',  (z   ), estimated uncertainty')
143     114 format (/' initial |g| =', 1p1e13.5, '    (degrees)'
144        &        /'   final |g| =',    e13.5)
145
146     121 format (20x, i5,')',i5, 1p2e15.5, ',    (n+ik),  d(n+ik)')
147     122 format (20x, i5,')',i5, 1p2e15.5, ',    (z   ),  d(z   )')
148
149         end
```

6.2.10 ZOOM2.FOR

```
      subroutine zoom2 (ppp1, ppp2, ppp3)        ! constrained optimization
      real       ppp1(1), ppp2(1), ppp3(1)       ! used by:   SCAN2

      include   'iounit.'
      include   'defnit.'
      include   'filmmm.'
      include   'wstack.'
      include   'cgnxx1.'

      data       small / 1.0E-5 /

      raddeg = 180.0 / 3.14159265                ! degrees <--- radians
      llnorm = .false.                           ! ASMBL
      call asmbl                                 ! ia,ja,aa,bb
      call norm (meqns, bb, bn, 1)               ! residual
      call scaljj (meqns,mvary,ia,ja,aa,xx, aats,w,2) ! scale columns, A

      if (bn .eq. 0.0) return
      bn  = bn*raddeg                            ! degrees <--- radians
      bn1 = bn                                   ! initial
      bn2 = bn                                   ! last
      loop = 0
      niter = mvary*4                            ! number of iterations
c     write (iout,101)
c     write (iout,102)  loop, bn

    1 itry = 0                                   ! initialize
      loop = loop+1
      do i=1,mvary                               ! initialize
          xx(i) = 0.0                            !     Newton step
      end do
      call cgnl (meqns,mvary, ia,ja,aa,bb,xx,
     &           niter,        u,v,w,  xw,se)
      do i=1,mvary                               ! account for scaling
          xx(i) = xx(i)/aats(i)                  !   columns in SCALJJ
          se(i) = se(i)/aats(i)
      end do

    2 do i=1,mvary                               ! update
          j = iptu(i)
          h = xx(i)*0.2                          ! step length
          s = se(i)*0.2                          ! estimated std dev

          if (j.le.mlmnts) then                  ! n+ik
              f = diefcn(j)
              g = uncerl(j)
c*            write (iout,121)  i,j,f,h
              ha = amax1 (s, 0.001, abs(f*0.2))
              ha = amin1 (g, abs(h), ha, ppp3(i))
              if (h.lt.0.0)  ha=-ha
```

```
51                aat(i) = f                          ! retain
52                diefcn(j) = amax1 (0.0, f+ha)
53
54                if (diefcn(j) .lt. ppp1(i)) then    ! ensure compactness
55                    diefcn(j) = ppp1(i)
56                else if (diefcn(j) .gt. ppp2(i)) then
57                    diefcn(j) = ppp2(i)
58                end if
59            else                                    ! z
60                j = j-mlmnts
61                f = widths(j)
62                g = uncerz(j)
63  c*            write (iout,122)  i,j,f,h
64                ha = amax1 (s, 0.01, f*0.2)
65                ha = amin1 (g, abs(h), ha, ppp3(i))
66                if (h.lt.0.0)  ha=-ha
67                aat(i) = f                          ! retain
68                widths(j) = amax1 (0.0, f+ha)
69
70                if (widths(j) .lt. ppp1(i)) then    ! ensure compactness
71                    widths(j) = ppp1(i)
72                else if (widths(j) .gt. ppp2(i)) then
73                    widths(j) = ppp2(i)
74                end if
75            end if
76        end do
77
78        call asmbl                                  ! ia,ja,aa,bb
79        call norm (meqns, bb, bn, 1)                ! residual
80        call scaljj (meqns,mvary,ia,ja,aa,xx, aats,w,2) ! scale columns, A
81        bn = bn*raddeg                              ! degrees
82        total = bn /bn1                             ! total    reduction
83        relat = bn /bn2                             ! relative reduction
84
85        if (total .le. 1.0E-5) then                 ! convergence
86            bn2 = bn                                !     retain
87            goto 4                                  !     escape
88        end if
89
90        if (relat .le. 0.999) then                  ! converging
91            bn2 = bn                                !     retain
92            goto 3                                  !     iterate
93        end if
94
95        do i=1,mvary                                ! reset
96            j = iptu(i)
97            if (j.le.mlmnts) then
98                diefcn(j) = aat(i)
99            else
100               j = j-mlmnts
101               widths(j) = aat(i)
102           end if
103       end do
```

```
104
105         if (itry.lt.3) then                          ! try again
106             itry = itry+1
107             do i=1,mvary
108                 xx(i) = xx(i)*0.5                    ! reduce stepsize
109             end do
110             goto 2
111         end if
112
113 c       if (loop .le. 3) then                        ! convenience
114 c           write (iout,104)
115 c           if (relat .le. 1.0) then                 ! marginal
116 c               write (iout,105)
117 c           else                                     ! divergence
118 c               write (iout,106)
119 c           end if
120 c       end if
121         goto 5                                       ! escape
122
123       3 continue
124 c       write (iout,103)  loop, relat, total, bn
125         goto 1
126       4 continue
127 c       write (iout,103)  loop, relat, total, bn
128       5 continue
129
130 c       write (iout,114)  bn1, bn2                   ! compare
131         return
132
133     101 format (/' zoom2:  loop,    ratio of reduction,     |g|'
134        &        /'                  (rel)       (total)         ')
135     102 format ( 8x, i5, 24x, 1p1e12.3, '   (degrees)'  )
136     103 format ( 8x, i5,      1p3e12.3)
137     104 format ( 8x, 'stepsize reduction attempted.')
138     105 format ( 8x, '... slow convergence.')
139     106 format ( 8x, '... divergence.')
140
141     111 format (/' model parameter value along the minimum: ')
142     114 format (/' initial |g| =', 1p1e13.5, '   (degrees)'
143        &        /'   final |g| =',    e13.5)
144
145     121 format (20x, 2i5, 1p2e15.5, ',    diefcn')
146     122 format (20x, 2i5, 1p2e15.5, ',    widths')
147         end
```

6.2.11 ZOOM3.FOR

```
      subroutine zoom3 (ppp1, ppp2, ppp3,
     &                  isx, iwsx, mwsx, iawsx, irawsx, nglx)

      real       ppp1(1), ppp2(1), ppp3(1)
      integer    isx, iwsx, mwsx, iawsx, irawsx, nglx

      include   'iounit.'
      include   'defnit.'
      include   'filmmm.'
      include   'wstack.'
      include   'cgnxx1.'

      data       small / 1.0E-5 /

c     This routine is called/used by:    SCAN3

      raddeg = 180.0 / 3.14159265                ! degrees <--- radians
      llnorm = .false.                           ! ASMBL
      call asmbl3 (isx, iwsx, mwsx, iawsx, irawsx, nglx)   ! ia,ja,aa,bb
      call norm (meqns, bb, bn, 1)               ! residual
      call scaljj (meqns,mvary,ia,ja,aa,xx, aats,w,2) ! scale columns, A

      if (bn .eq. 0.0) return
      bn  = bn*raddeg                            ! degrees <--- radians
      bn1 = bn                                   ! initial
      bn2 = bn                                   ! last
      loop = 0
      niter = mvary*4                            ! number of iterations
c     write (iout,101)
c     write (iout,102)  loop, bn

    1 itry = 0                                   ! initialize
      loop = loop+1
      do i=1,mvary                               ! initialize
          xx(i) = 0.0                            !     Newton step
      end do
      call cgnl (meqns,mvary, ia,ja,aa,bb,xx,
     &           niter,        u,v,w,  xw,se)
      do i=1,mvary                               ! account for scaling
          xx(i) = xx(i)/aats(i)                  !   columns in SCALJJ
          se(i) = se(i)/aats(i)
      end do

    2 do i=1,mvary                               ! update
          j = iptu(i)
          h = xx(i)*0.2                          ! step length
          s = se(i)*0.2                          ! estimated std dev

          if (j.le.mlmnts) then                  ! n+ik
              f = diefcn(j)
```

```
51                g = uncerl(j)
52  c*            write (iout,121)  i,j,f,h
53                ha = amax1 (s, 0.001, abs(f*0.2))
54                ha = amin1 (g, abs(h), ha, ppp3(i))
55                if (h.lt.0.0)  ha=-ha
56                aat(i) = f                          ! retain
57                diefcn(j) = amax1 (0.0, f+ha)
58
59                if (diefcn(j) .lt. ppp1(i)) then   ! ensure compactness
60                    diefcn(j) = ppp1(i)
61                else if (diefcn(j) .gt. ppp2(i)) then
62                    diefcn(j) = ppp2(i)
63                end if
64            else                                    ! z
65                j = j-mlmnts
66                f = widths(j)
67                g = uncerz(j)
68  c*            write (iout,122)  i,j,f,h
69                ha = amax1 (s, 0.01, f*0.2)
70                ha = amin1 (g, abs(h), ha, ppp3(i))
71                if (h.lt.0.0)  ha=-ha
72                aat(i) = f                          ! retain
73                widths(j) = amax1 (0.0, f+ha)
74
75                if (widths(j) .lt. ppp1(i)) then   ! ensure compactness
76                    widths(j) = ppp1(i)
77                else if (widths(j) .gt. ppp2(i)) then
78                    widths(j) = ppp2(i)
79                end if
80            end if
81        end do
82
83        call asmbl3 (isx, iwsx, mwsx, iawsx, irawsx, nglx)   ! ia,ja,aa,bb
84        call norm (meqns, bb, bn, 1)                ! residual
85        call scaljj (meqns,mvary,ia,ja,aa,xx, aats,w,2) ! scale columns, A
86        bn = bn*raddeg                              ! degrees
87        total = bn /bn1                             ! total    reduction
88        relat = bn /bn2                             ! relative reduction
89
90        if (total .le. 1.0E-5) then                 ! convergence
91            bn2 = bn                                !     retain
92            goto 4                                  !     escape
93        end if
94
95        if (relat .le. 0.999) then                  ! converging
96            bn2 = bn                                !     retain
97            goto 3                                  !     iterate
98        end if
99
100       do i=1,mvary                                ! reset
101           j = iptu(i)
102           if (j.le.mlmnts) then
103               diefcn(j) = aat(i)
```

```
104             else
105                 j = j-mlmnts
106                 widths(j) = aat(i)
107             end if
108         end do
109
110         if (itry.lt.3) then                          ! try again
111             itry = itry+1
112             do i=1,mvary
113                 xx(i) = xx(i)*0.5                    ! reduce stepsize
114             end do
115             goto 2
116         end if
117
118 c       if (loop .le. 3) then                        ! convenience
119 c           write (iout,104)
120 c           if (relat .le. 1.0) then                 ! marginal
121 c               write (iout,105)
122 c           else                                     ! divergence
123 c               write (iout,106)
124 c           end if
125 c       end if
126         goto 5                                       ! escape
127
128       3 continue
129 c       write (iout,103)  loop, relat, total, bn
130         goto 1
131       4 continue
132 c       write (iout,103)  loop, relat, total, bn
133       5 continue
134
135 c       write (iout,114)  bn1, bn2                   ! compare
136         return
137
138     101 format (/' zoom3:  loop,    ratio of reduction,     |g|'
139        &        /'                  (rel)       (total)         ')
140     102 format ( 8x, i5, 24x, 1p1e12.3, '   (degrees)'  )
141     103 format ( 8x, i5,      1p3e12.3)
142     104 format ( 8x, 'stepsize reduction attempted.')
143     105 format ( 8x, '... slow convergence.')
144     106 format ( 8x, '... divergence.')
145
146     111 format (/' model parameter value along the minimum: ')
147     114 format (/' initial |g| =', 1p1e13.5, '   (degrees)'
148        &        /'   final |g| =',    e13.5)
149
150     121 format (20x, 2i5, 1p2e15.5, ',    diefcn')
151     122 format (20x, 2i5, 1p2e15.5, ',    widths')
152         end
```

6.2.12 ASMBL.FOR

```
      subroutine asmbl
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      include  'filmss.'
      include  'seamx1.'

      logical   firstv
      real      a(nrows*2), b(2), c(2)

c     The sparse matrix format for the model parameters is of the form:
c         [(n/k)_(1), ..., (n/k)_(mlmnts)]  ^ diefcn (w)
c         [    z_(1), ...,     z_(mfilms)]  ^ widths


      call arrang
      mm = mlmnts+mfilmm                          ! model parameters
      ii = 1                                      ! IA
      ia(1) = 1
      jj = 0                                      ! JA

      iws = 0
      mws = 0
      iaws = 0
      iraws = 0
      ngl = 0                                     ! measured data
      do is=1,msampl
          mfilm = nnfilm(is)                      ! FILMSS
          mwave = nnwave(is)
          mfilms = mfilm+1                        ! films/substrate
          nrow = mfilm*3+2
          do iw=1,mwave
              iws = iws+1
              iwave = iiwave(iws)
              mbien = nnbent(iws)
              qq  = waveqq(iwave)                 ! FILMSS
              do i=1,nrow                         ! initialize local pointers
                  iptx(i) = 0                     ! vary --> unique
                  ipty(i) = 0                     ! vary --> full
              end do
              iv = 0                              ! local,full, non-unique
              kv = 0                              ! local,vary, non-unique
              mv = 0                              ! local,vary,   compress

              do m=1,mfilms                       ! films/substrate
                  if (m.ne.mfilms) then           ! films ^ z
                      mws = mws+1
                      iz = iifilm(mws)
                      zzz(m) = widths(iz)         ! FILMSS
                      iv = iv+1                   ! local,full
```

```
51                      if (lvaryz(iz).eq.1) then  ! vary
52                          j = mlmnts+iz
53                          i = iptv(j)            !    nonlocal, compress
54                          kv = kv+1              !    local, non-unique
55                          iptx(kv) = i           !    local ' nonlocal
56                          ipty(kv) = iv          !    local
57                          if (iptw(i).eq.0) then !    compress,unique
58                              mv = mv+1          !        local counter
59                              iptw(i) = mv       !        local
60                              jj = jj+1          !        within row A
61                              ja(jj) = i         !        column
62                          end if
63                      end if
64                  end if
65                  do ink=1,2                     ! n+ik
66                      mws = mws+1
67                      nk = iifilm(mws)
68                      c(ink) = diefcn(nk)        ! n,k
69                      iv = iv+1                  ! local, full
70                      if (lvaryl(nk).eq.1) then  ! vary
71                          i = iptv(nk)           !    nonlocal, compress
72                          kv = kv+1              !    local, non-unique
73                          iptx(kv) = i           !    local ' nonlocal
74                          ipty(kv) = iv          !    local
75                          if (iptw(i).eq.0) then !    compress, unique
76                              mv = mv+1          !        local counter
77                              iptw(i) = mv       !        local
78                              jj = jj+1          !        within row A
79                              ja(jj) = i         !        column
80                          end if
81                      end if
82                  end do
83                  die(m) = cmplx (c(1), c(2)) **2          ! FILMSS
84              end do          ! mfilms
85
86              if (mv.eq.0) then
87                  write (iout,100) is,iw,iwave
88                  stop
89              end if
90
91              firstv = .true.
92
93              do mbn=1,mbien                     ! ambients
94                  iaws = iaws+1
95                  imbien = iibent(iaws)
96                  mrpeat = nnpeat(iaws)
97                  air = ambent(imbien)           ! FILMSS
98                  do irpeat=1,mrpeat
99                      iraws = iraws+1
100                     mangl = mangle(iraws)
101                     do iangl=1,mangl           ! incident angles
102                         ngl = ngl+1            ! measured data
103                         angl = angles(ngl)
```

```
104                         call scatr (qq,angl, b,a,c)
105
106  c*                 write (iout,101) ngl, psiiis(ngl), deltas(ngl), b
107
108                         bb(ii  ) = psiiis(ngl) - b(1)    !   psi
109                         bb(ii+1) = deltas(ngl) - b(2)    ! delta
110
111                         ia(ii+1) = ia(ii  )+mv           !   psi
112                         ia(ii+2) = ia(ii+1)+mv           ! delta
113
114                     if (mv.ne.0) then          ! <==================
115                         if (firstv) then       ! psi completed already
116                             firstv = .false.
117                             do j=1,mv                    ! unique
118                                 jj = jj+1
119                                 ja(jj) = ja(jj-mv)       ! delta
120                             end do
121                         else
122                             do j=1,mv                    ! unique
123                                 jj = jj+1
124                                 ja(jj   ) = ja(jj-mv)    !   psi
125                                 ja(jj+mv) = ja(jj   )    ! delta
126                             end do
127                             jj = jj+mv
128                         end if
129
130                         j1 = ia(ii  )
131                         j2 = ia(ii+1)-1
132                         do j=j1,j2             ! compress
133                             aa(j   ) = 0.0     !     psi
134                             aa(j+mv) = 0.0     !   delta
135                         end do
136
137                         do k=1,kv              !    local, vary, non-unique
138                             iv  = ipty(k)      !    local, full
139                             iv2 = iv+iv        ! delta ' a ' local,full
140                             iv1 = iv2-1        !   psi ' a ' local,full
141
142                             i   = iptx(k)      ! nonlocal, compress
143                             imv = iptw(i)      !    local, compress --> ja,aa
144                             jj1 = j1-1+imv     !   psi ' aa
145                             jj2 = jj1 + mv     ! delta ' aa
146
147                             aa(jj1) = aa(jj1) + a(iv1)   !   psi'
148                             aa(jj2) = aa(jj2) + a(iv2)   ! delta'
149                         end do                           ! two rows in A
150                     end if                     ! <==================
151
152  c                      Renormalize rows in the matrix, least square.
153                         if (llnorm) then
154                             bb(ii  ) = bb(ii  ) /psiiiu(ngl)         !   psi
155                             bb(ii+1) = bb(ii+1) /deltau(ngl)         ! delta
156
```

```
157                             if (mv.ne.0) then  ! <==================
158                             do j=1,mv
159                                 aa(j1-1+j) = aa(j1-1+j) /psiiiu(ngl) !   psi
160                                 aa(j2  +j) = aa(j2  +j) /deltau(ngl) ! delta
161                             end do
162                             end if             ! <==================
163                         end if
164
165                         ii = ii+2
166                     end do      ! angle
167                 end do         ! repeat
168             end do            ! ambient
169
170             if (mv.ne.0) then                  ! <==================
171                 do k=1,kv                      ! re-initialize
172                     i = iptx(k)                !    unique-ness
173                     iptw(i) = 0                !    indicator
174                 end do
175             end if                             ! <==================
176
177         end do        ! wave
178     end do           ! sample
179     meqns = ii-1
180
181     if (jj .ne. ia(ii)-1) then
182         write (iout,102)  ii, jj, ia(ii)
183         stop
184     end if
185     if (jj .gt. nnjaaa) then
186         write (iout,104)
187         stop
188     end if
189     return
190
191 100 format (' oops,   there is NO varying model parameter'
192    &       /'         for the case involving:   sample = ', i3
193    &       /'         where the ', i3, '-th wave = ', i3)
194 101 format (' asmbl, ', i5, 1p2e14.4, ' ' ngl, psi, delta'
195    &       /'        ', 5x,   2e14.4)
196 102 format (' asmbl,   inconsistent format of sparse matrix, '
197    &       /'          ii = ', i10
198    &       /'          jj = ', i10, ' =/= ', i10, ' = ia(ii)-1 ')
199 104 format (' asmbl,   array allocation for the sparse matrix '
200    &       /'          has been exceeded. '
201    &       /'          aa,ja <---- nnjaaa (DEFNIT.)' )
202
203      end
```

6.2.13 ASMBL3.FOR

```
1  c     called by:   ZOOM3
2
3        subroutine asmbl3 (isx, iwsx, mwsx, iawsx, irawsx, nglx)
4
5        include  'iounit.'
6        include  'defnit.'
7        include  'filmmm.'
8        include  'filmss.'
9        include  'seamx1.'
10
11       logical   firstv
12       real      a(nrows*2), b(2), c(2)
13
14 c     The sparse matrix format for the model parameters is of the form:
15 c         [(n/k)_(1), ..., (n/k)_(mlmnts)]  ' diefcn (w)
16 c         [    z_(1), ...,     z_(mfilms)]  ' widths
17
18
19 c*   call arrang
20       mm = mlmnts+mfilmm                          ! model parameters
21       ii = 1                                      ! IA
22       ia(1) = 1
23       jj = 0                                      ! JA
24
25       is = isx
26       iws = iwsx     ! 0
27       mws = mwsx     ! 0
28       iaws = iawsx   ! 0
29       iraws = irawsx ! 0
30       ngl = nglx     ! 0                          ! measured data
31
32 c*   do is=1,msampl
33           mfilm = nnfilm(is)                      ! FILMSS
34           mwave = nnwave(is)
35           mfilms = mfilm+1                        ! films/substrate
36           nrow = mfilm*3+2
37           do iw=1,mwave
38               iws = iws+1
39               iwave = iiwave(iws)
40               mbien = nnbent(iws)
41               qq  = waveqq(iwave)                 ! FILMSS
42               do i=1,nrow                         ! initialize local pointers
43                   iptx(i) = 0                     ! vary --> unique
44                   ipty(i) = 0                     ! vary --> full
45               end do
46               iv = 0                              ! local,full, non-unique
47               kv = 0                              ! local,vary, non-unique
48               mv = 0                              ! local,vary,   compress
49
50               do m=1,mfilms                       ! films/substrate
```

```
51                      if (m.ne.mfilms) then          ! films ' z
52                          mws = mws+1
53                          iz = iifilm(mws)
54                          zzz(m) = widths(iz)        ! FILMSS
55                          iv = iv+1                  ! local,full
56                          if (lvaryz(iz).eq.1) then  ! vary
57                              j = mlmnts+iz
58                              i = iptv(j)            !    nonlocal, compress
59                              kv = kv+1              !    local, non-unique
60                              iptx(kv) = i           !    local ' nonlocal
61                              ipty(kv) = iv          !    local
62                              if (iptw(i).eq.0) then !    compress,unique
63                                  mv = mv+1          !        local counter
64                                  iptw(i) = mv       !        local
65                                  jj = jj+1          !        within row A
66                                  ja(jj) = i         !        column
67                              end if
68                          end if
69                      end if
70                      do ink=1,2                     ! n+ik
71                          mws = mws+1
72                          nk = iifilm(mws)
73                          c(ink) = diefcn(nk)        ! n,k
74                          iv = iv+1                  ! local, full
75                          if (lvaryl(nk).eq.1) then  ! vary
76                              i = iptv(nk)           !    nonlocal, compress
77                              kv = kv+1              !    local, non-unique
78                              iptx(kv) = i           !    local ' nonlocal
79                              ipty(kv) = iv          !    local
80                              if (iptw(i).eq.0) then !    compress, unique
81                                  mv = mv+1          !        local counter
82                                  iptw(i) = mv       !        local
83                                  jj = jj+1          !        within row A
84                                  ja(jj) = i         !        column
85                              end if
86                          end if
87                      end do
88                      die(m) = cmplx (c(1), c(2)) **2          ! FILMSS
89                  end do          ! mfilms
90
91  c*              if (mv.eq.0) then
92  c*                  write (iout,100)  is, iw, iwave
93  c*                  stop
94  c*              end if
95
96                  firstv = .true.
97
98                  do mbn=1,mbien                     ! ambients
99                      iaws = iaws+1
100                     imbien = iibent(iaws)
101                     mrpeat = nnpeat(iaws)
102                     air = ambent(imbien)           ! FILMSS
103                     do irpeat=1,mrpeat
```

```
104                     iraws = iraws+1
105                     mangl = mangle(iraws)
106                     do iangl=1,mangl          ! incident angles
107                         ngl = ngl+1           ! measured data
108                         angl = angles(ngl)
109                         call scatr (qq,angl, b,a,c)
110
111 c*                  write (iout,101) ngl, psiiis(ngl), deltas(ngl), b
112
113                         bb(ii  ) = psiiis(ngl) - b(1)    !   psi
114                         bb(ii+1) = deltas(ngl) - b(2)    ! delta
115
116                         ia(ii+1) = ia(ii  )+mv           !   psi
117                         ia(ii+2) = ia(ii+1)+mv           ! delta
118
119                     if (mv.ne.0) then         ! <==================
120                         if (firstv) then      ! psi completed already
121                             firstv = .false.
122                             do j=1,mv                    ! unique
123                                 jj = jj+1
124                                 ja(jj) = ja(jj-mv)       ! delta
125                             end do
126                         else
127                             do j=1,mv                    ! unique
128                                 jj = jj+1
129                                 ja(jj   ) = ja(jj-mv)    !   psi
130                                 ja(jj+mv) = ja(jj   )    ! delta
131                             end do
132                             jj = jj+mv
133                         end if
134
135                         j1 = ia(ii  )
136                         j2 = ia(ii+1)-1
137                         do j=j1,j2            ! compress
138                             aa(j   ) = 0.0    !     psi
139                             aa(j+mv) = 0.0    !   delta
140                         end do
141
142                         do k=1,kv             !    local, vary, non-unique
143                             iv  = ipty(k)     !    local, full
144                             iv2 = iv+iv       ! delta ' a ' local,full
145                             iv1 = iv2-1       !   psi ' a ' local,full
146
147                             i   = iptx(k)     ! nonlocal, compress
148                             imv = iptw(i)     !    local, compress --> ja,aa
149                             jj1 = j1-1+imv    !   psi ' aa
150                             jj2 = jj1 + mv    ! delta ' aa
151
152                             aa(jj1) = aa(jj1) + a(iv1)   !   psi'
153                             aa(jj2) = aa(jj2) + a(iv2)   ! delta'
154                         end do                           ! two rows in A
155                     end if                    ! <==================
156
```

```
c                           Renormalize rows in the matrix, least square.
                            if (llnorm) then
                                bb(ii  ) = bb(ii  ) /psiiiu(ngl)         !   psi
                                bb(ii+1) = bb(ii+1) /deltau(ngl)         ! delta

                                if (mv.ne.0) then  ! <------------------
                                do j=1,mv
                                    aa(j1-1+j) = aa(j1-1+j) /psiiiu(ngl) !   psi
                                    aa(j2  +j) = aa(j2  +j) /deltau(ngl) ! delta
                                end do
                                end if             ! <------------------
                            end if

                            ii = ii+2
                        end do           ! angle
                    end do               ! repeat
                end do                   ! ambient

                if (kv.ne.0) then
                    do k=1,kv                       ! re-initialize
                        i = iptx(k)                 !    unique-ness
                        iptw(i) = 0                 !    indicator
                    end do
                end if

            end do        ! wave
c*      end do            ! sample
        meqns = ii-1

        if (jj .ne. ia(ii)-1) then
            write (iout,102)  ii, jj, ia(ii)
            stop
        end if
        if (jj .gt. nnjaaa) then
            write (iout,104)
            stop
        end if

        return

  100 format (' oops,   there is NO varying model parameter'
     &       /'         for the case involving:   sample = ', i3
     &       /'         where the ', i3, '-th wave = ', i3)
  101 format (' asmbl, ', i5, 1p2e14.4, ' ' ngl, psi, delta'
     &       /'       ', 5x,   2e14.4)
  102 format (' asmbl,   inconsistent format of sparse matrix, '
     &       /'          ii = ', i10
     &       /'          jj = ', i10, ' =/= ', i10, ' = ia(ii)-1 ')
  104 format (' asmbl,   array allocation for the sparse matrix '
     &       /'          has been exceeded. '
     &       /'          aa,ja <---- nnjaaa (DEFNIT.)' )

        end
```

6.2.14 ASMBLX.FOR

```
      subroutine  asmblx                          ! called by:   SEAMA
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      include  'filmss.'
      include  'seamx1.'
      logical   firstv, firstu
      real      a(nrows*2), b(2), c(2)

c     Construct sparse matrix associated with model experiment.
c     Configuration:       angle,repeat,ambient,wave,sample.
c     ------------------------------------------------------
c     Ordering of indices in IPTU:    [vary-->     <--froz]

      call arrang
      mm = mlmnts+mfilmm                          ! model parameters

      ii = 1
      ia(1) = 1                                   ! vary
      jj = 0

      iaa(1) = 1                                  ! frozen
      jja = 0

      iws = 0                             !                    wave,sampl
      mws = 0                             !              nfilms,wave,sampl
      iaws = 0                            !             ambient,wave,sampl
      iraws = 0                           !       repeat,ambient,wave,sampl
      ngl = 0                             ! ndegr,repeat,ambient,wave,sampl
      do is=1,msampl
          mfilm = nnfilm(is)                      ! FILMSS
          mwave = nnwave(is)
          mfilms = mfilm+1                        ! film/substrate
          nrow = mfilm*3+2                        ! (z,n,k) (n,k)
          do iw=1,mwave
              iws = iws+1
              iwave = iiwave(iws)
              mbien = nnbent(iws)
              qq = waveqq(iwave)                  ! FILMSS

              do i=1,nrow
                  iptx(i) = 0                     ! vary ---> unique
                  ipty(i) = 0                     ! vary ---> full
                  kptx(i) = 0                     ! froz ---> unique
                  kpty(i) = 0                     ! froz ---> full
              end do
              iv = 0                              ! local, full, non-unique
              kv = 0                              ! local, vary, non-unique
              mv = 0                              ! local, vary,   compress
              ku = 0                              ! local, froz, non-unique
```

```
51                  mu = 0                              ! local, froz,   compress
52
53                  do m=1,mfilms                       ! films/substrate
54                      if (m.ne.mfilms) then           ! films ' z
55                          mws = mws+1
56                          iz = iifilm(mws)
57                          zzz(m) = widths(iz)         ! FILMSS
58                          iv = iv+1                   !    local, full
59                          j = mlmnts+iz               ! nonlocal, full
60                          if (lvaryz(iz).eq.1) then   ! vary
61                              i = iptv(j)             !    nonlocal, compress
62                              kv = kv+1               !    local, non-unique
63                              iptx(kv) = i            !    local ' nonlocal
64                              ipty(kv) = iv           !    local, full
65                              if (iptw(i).eq.0) then  !    compress, unique
66                                  mv = mv+1           !    local counter
67                                  iptw(i) = mv        !    local
68                                  jj = jj+1
69                                  ja(jj) = i
70                              end if
71                          else                        ! frozen
72                              i = iptv(j)             !    nonlocal, compress (backwards)
73                              ku = ku+1               !    local, non-unique
74                              kptx(ku) = i            !    local ' nonlocal   (backwards)
75                              kpty(ku) = iv           !    local, full
76                              if (iptw(i).eq.0) then  !    compress, unique
77                                  mu = mu+1           !    local counter
78                                  iptw(i) = mu        !    local
79                                  jja = jja+1
80                                  jaa(jja) = i        !    backwards, mm+1-i=k
81                              end if
82                          end if
83                      end if                          ! z
84                      do ink=1,2                      ! n+ik
85                          mws = mws+1
86                          nk = iifilm(mws)
87                          c(ink) = diefcn(nk)         ! n,k
88                          iv = iv+1                   ! local, full, non-unique
89                          if (lvaryl(nk).eq.1) then   ! vary
90                              i = iptv(nk)            !    nonlocal, compress
91                              kv = kv+1               !    local, non-unique
92                              iptx(kv) = i
93                              ipty(kv) = iv
94                              if (iptw(i).eq.0) then  !    compress, unique
95                                  mv = mv+1           !    local counter
96                                  iptw(i) = mv        !    local
97                                  jj = jj+1
98                                  ja(jj) = i
99                              end if
100                         else                        ! frozen
101                             i = iptv(nk)            !    nonlocal, compress (backwards)
102                             ku = ku+1               !    local, non-unique
103                             kptx(ku) = i            !    (backwards)
```

```
                        kpty(ku) = iv
                        if (iptw(i).eq.0) then !    compress, unique
                            mu = mu+1          !    local counter
                            iptw(i) = mu       !    local
                            jja = jja+1
                            jaa(jja) = i       !    backwards, mm+1-i=k
                        end if
                    end if
                end do     ! n+ik
                die(m) = cmplx (c(1),c(2)) **2 ! FILMSS
            end do         ! mfilms

            if (mv.eq.0) then
                write (iout,100)  is,iw,iwave
                stop
            end if

            firstv = .true.
            firstu = .true.

            do mbn=1,mbien                     ! ambients
                iaws = iaws+1
                imbien = iibent(iaws)
                mrpeat = nnpeat(iaws)
                air = ambent(imbien)           ! FILMSS
                do irpeat=1,mrpeat             ! repeats
                    iraws = iraws+1
                    mangl = mangle(iraws)
                    do iangl=1,mangl           ! angles of incidence
                        ngl = ngl+1            ! measurement data
                        angl = angles(ngl)
                        angu = angleu(ngl)

                        call scatr (qq,angl, b,a,c)

                        bb(ii  ) = psiiis(ngl) - b(1)    ! deviation
                        bb(ii+1) = deltas(ngl) - b(2)

                        cc(ii  ) = c(1)                  ! d/d incident angle
                        cc(ii+1) = c(2)
c                       ------------------------------------------
                        ia(ii+1) = ia(ii)+mv             ! vary
                        ia(ii+2) = ia(ii)+mv+mv
                        if (mv.ne.0) then
                            if (firstv) then
                                firstv = .false.
                                do j=1,mv
                                    jj = jj+1
                                    ja(jj) = ja(jj-mv)
                                end do
                            else
                                do j=1,mv
                                    jj = jj+1
```

```
157                                     ja(jj   ) = ja(jj-mv)
158                                     ja(jj+mv) = ja(jj   )
159                                 end do
160                                 jj = jj+mv
161                             end if
162
163                             j1 = ia(ii  )
164                             j2 = ia(ii+1)-1
165                             do j=j1,j2          ! compress, initialize
166                                 aa(j   ) = 0.0 !   psi
167                                 aa(j+mv) = 0.0 ! delta
168                             end do
169
170                             do k=1,kv           ! local, non-unique
171                                 iv  = ipty(k)  ! local full
172                                 iv2 = iv+iv    ! delta ' a ' local,full
173                                 iv1 = iv2-1    !   psi ' a ' local,full
174
175                                 i   = iptx(k)  ! nonlocal, compress
176                                 imv = iptw(i)  !    local, compress --> ja,aa
177                                 jj1 = j1-1+imv !   psi ' aa
178                                 jj2 = jj1 + mv ! delta ' aa
179
180                                 aa(jj1) = aa(jj1) + a(iv1)    !   psi'
181                                 aa(jj2) = aa(jj2) + a(iv2)    ! delta'
182                             end do
183                         end if
184 c                       ------------------------------------------
185                         iaa(ii+1) = iaa(ii)+mu          ! frozen
186                         iaa(ii+2) = iaa(ii)+mu+mu
187                         if (mu.ne.0) then
188                             if (firstu) then
189                                 firstu = .false.
190                                 do j=1,mu
191                                     jja = jja+1
192                                     jaa(jja) = jaa(jja-mu)
193                                 end do
194                             else
195                                 do j=1,mu
196                                     jja = jja+1
197                                     jaa(jja   ) = jaa(jja-mu)
198                                     jaa(jja+mu) = jaa(jja   )
199                                 end do
200                                 jja = jja+mu
201                             end if
202
203                             j1 = iaa(ii  )
204                             j2 = iaa(ii+1)-1
205                             do j=j1,j2              ! compress, initialize
206                                 aaa(j   ) = 0.0
207                                 aaa(j+mu) = 0.0
208                             end do
209
```

```
210                             do k=1,ku               ! compress
211                                 iv  = kpty(k)       ! a " local Jacobian
212                                 iv2 = iv+iv         ! delta
213                                 iv1 = iv2-1         ! psi
214
215                                 i = kptx(k)         ! backwards
216                                 imu = iptw(i)       ! local, compress
217                                 jj1 = j1-1+imu
218                                 jj2 = jj1 + mu
219
220                                 aaa(jj1) = aaa(jj1) + a(iv1)
221                                 aaa(jj2) = aaa(jj2) + a(iv2)
222                             end do
223                         end if       ! frozen
224 c                       ------------------------------------------
225                         if (llnorm) then
226                             bb(ii  ) = bb(ii  ) /psiiiu(ngl)
227                             bb(ii+1) = bb(ii+1) /deltau(ngl)
228                             cc(ii  ) = cc(ii  ) /psiiiu(ngl)
229                             cc(ii+1) = cc(ii+1) /deltau(ngl)
230
231                             j1 = ia(ii  )
232                             j2 = ia(ii+1)-1
233                             do j=j1,j2
234                                 aa(j   ) = aa(j   ) /psiiiu(ngl)
235                                 aa(j+mv) = aa(j+mv) /deltau(ngl)
236                             end do
237
238                             if (mu.ne.0) then           ! frozen
239                                 j1 = iaa(ii  )
240                                 j2 = iaa(ii+1)-1
241                                 do j=j1,j2
242                                     aaa(j   ) = aaa(j   )/psiiiu(ngl)
243                                     aaa(j+mu) = aaa(j+mu)/deltau(ngl)
244                                 end do
245                             end if
246                         end if       ! renormalize
247
248                         ii = ii+2
249                     end do           ! angles
250                 end do               ! repeat
251             end do                   ! ambient
252
253             if (kv.ne.0) then                ! vary
254                 do k=1,kv                    !    reset
255                     i = iptx(k)              !    unique-ness
256                     iptw(i) = 0              !    indicator
257                 end do
258             end if
259             if (ku.ne.0) then                ! frozen
260                 do k=1,ku                    !    reset
261                     i = kptx(k)              !    unique-ness
262                     iptw(i) = 0              !    indicator
```

```
263                     end do
264                 end if
265
266             end do          ! wave
267         end do              ! sample
268         meqns = ii-1
269
270         if (jj .ne. ia(ii)-1) then
271             write (iout,102)  ii, jj, ia(ii)
272             stop
273         end if
274         if (jja .ne. iaa(ii)-1) then
275             write (iout,103)  ii, jja, iaa(ii)
276             stop
277         end if
278         if (jj.gt.nnjaaa .or. jja.gt.nnjaaa) then
279             write (iout,104)
280             stop
281         end if
282
283         return
284
285     100 format (/' asmblx,  there is NO varying model parameter '
286        &        /'           for the case involving:    sample = ', i3
287        &        /'           where the ', i3, '-th wave = ', i3 )
288     102 format (/' asmblx,  inconsistent format of sparse matrix, ',
289        &                                          'aa " vary'
290        &        /'           ii = ', i10
291        &        /'           jj = ', i10, ' =/= ', i10, ' =/= ia(ii)-1')
292     103 format (/' asmblx,  inconsistent format of sparse matrix, '
293        &                                          'aaa " froz'
294        &        /'           ii = ', i10
295        &        /'           jja= ', i10, ' =/= ', i10, ' =/= iaa(ii)-1')
296     104 format (/' asmblx,  array allocation for the sparse matrix '
297        &        /'           has been exceeded.'
298        &        /'            aa,ja  <---- nnjaaa (DEFNIT.)'
299        &        /'           aaa,jaa <---- nnjaaa (DEFNIT.)' )
300
301         end
```

6.2.15 SCATR.FOR

This subroutine calculates the ellipsometric angles (Δ, ψ) and the associated partial derivatives.

```
      subroutine scatr (qq, angl, b,a,c)
      include  'defnit.'
      include  'filmss.'
      include  'rstack.'

      real      a(1), b(2), c(2)
      data  pi / 3.1415926 /


      call forwrd (qq,angl, Rs,Rp, dRs,dRp, dRsa,dRpa)  ! angl ~radians

      nrow = mfilm*3+2                 ! (z,n,k)    (,n,k)
      sx =  real (Rs)
      sy = aimag (Rs)
      px =  real (Rp)
      py = aimag (Rp)
      call polar (sx,sy,sr,sa,1)
      call polar (px,py,pr,pa,1)

      if (sr.eq.0.0) then
          if (pr.eq.0.0) then
              psi = 0.0
          else
              psi = 0.5*pi
          end if
      else if (pr.eq.0.0) then         ! sr > 0.0
          psi = 0.0
      else if (pr.le.sr) then
          psi = atan (pr/sr)
      else
          psi = 0.5*pi - atan(sr/pr)
      end if

      delta = pa-sa
    1 if (delta.lt.0.0) then
          delta = delta+2.0
          goto 1
      else if (delta.ge.2.0) then
          delta = delta-2.0
          goto 1
      end if
      delta = delta*pi

      b(1) = psi                   ! radians
      b(2) = delta                 ! radians
      k = 0
```

```
47
48      do n=1,nrow                  ! d(psi,delta) /d(e)   ' Jacobian
49          sxd =  real (dRs(n))
50          syd = aimag (dRs(n))
51          pxd =  real (dRp(n))
52          pyd = aimag (dRp(n))
53          ssrd = (sx*sxd+sy*syd)/sr                    !  |Rs|'
54          pprd = (px*pxd+py*pyd)/pr                    !  |Rp|'
55          sdel = (sx*syd-sy*sxd)/(sr*sr)               !  delta(s)'
56          pdel = (px*pyd-py*pxd)/(pr*pr)               !  delta(p)'
57
58          ddel = pdel-sdel                             !  delta'
59          dpsi = (sr*pprd-pr*ssrd)/(sr*sr+pr*pr)       !  psi'
60
61          k = k+1
62          a(k) = dpsi
63          k = k+1
64          a(k) = ddel
65      end do
66
67      sxd =  real (dRsa)
68      syd = aimag (dRsa)
69      pxd =  real (dRpa)
70      pyd = aimag (dRpa)
71      ssrd = (sx*sxd+sy*syd)/sr                        !  |Rs|'
72      pprd = (px*pxd+py*pyd)/pr                        !  |Rp|'
73      sdel = (sx*syd-sy*sxd)/(sr*sr)                   !  delta(s)'
74      pdel = (px*pyd-py*pxd)/(pr*pr)                   !  delta(p)'
75
76      ddel = pdel-sdel                                 !  delta'
77      dpsi = (sr*pprd-pr*ssrd)/(sr*sr+pr*pr)           !  psi'
78
79      c(1) = dpsi            ! d(psi  ) /d(angle),   Jacobian
80      c(2) = ddel            ! d(delta) /d(angle),   Jacobian
81
82      return
83      end
```

6.2.16 FORWRD.FOR

This subroutine calculates the reflection coefficients (R_s, R_p) and the associated partial derivatives.

```
      subroutine forwrd (qq,angl,  Rs,Rp,  dRs,dRp,  dRsa,dRpa)

      real     qq, angl           ! incident angle  ' radians
      complex  Rs,Rp,  dRs(1),dRp(1),  dRsa,dRpa

      complex  half,one,two,four,  eta0,eta0a,  cta0,cta0a
      complex  top,bot,ss,pp,  ssz,ppz,sse,ppe,ssa,ppa,  pp1,pp2,pp3
      include  'defnit.'          ! nfilms, nrows
      include  'filmss.'
      complex  ys(nfilms+1), eta(nfilms+1)
      complex  yp(nfilms+1), cta(nfilms+1)
      complex  ee(nfilms), em(nfilms), ep(nfilms), zq(nfilms)

c     Solve:   the direct/forward scattering problem.
c   Discern:   the reflection coefficient and Jacobian.
c     Given:   dielectric function of films (isotropic,homogeneous,uniform)
c              incident angle:    angl (radians)
c      Find:     Rs, Rp.      ------> (psi, delta)
c               dRs,dRp.      <------ partial wrt:   dielectric function
c                             <------ partial wrt:   n & k         ' (n+ik)
c              dRsa,dRpa      <------ partial wrt:  -[n(air)*sin(angle)]**2
c                             <------ partial wrt:  incident angle in radians
c     ... neglected here:     <------ partial wrt:   q

      sa = sin (angl)
      ca = cos (angl)
      as = air*sa
      ac = air*ca
      as2 = as*as                                  ! air
      eta0  = cmplx ( ac, 0.0)                     ! air   TE
      eta0a = cmplx (-as, 0.0)                     ! air   d(eta)/d(angl)
      cta0  = cmplx ( ca/air, 0.0)                 ! air   TM
      cta0a = cmplx (-sa/air, 0.0)                 ! air   d(cta)/d(angl)
      mfilms = mfilm+1                             ! films, substrate

      do i=1,mfilms                                ! films, substrate
          eta(i) = sqrtt (die(i)-cmplx(as2,0.0)) ! TE
          cta(i) = eta(i)/die(i)                   ! TM
      end do
      half = cmplx (0.5, 0.0)
      one  = cmplx (1.0, 0.0)
      two  = cmplx (2.0, 0.0)
      four = cmplx (4.0, 0.0)

c     Determine the reflection coefficients in air.
c     Method for TE:   admittance = Y = - Hx/Ey,       Rs uses E field.
```

```
47  c                TM:    impedance = Z =   Ex/Hy,       Rp uses H field.
48
49        ys(mfilms) = eta(mfilms)                                ! substrate
50        yp(mfilms) = cta(mfilms)
51        if (mfilm.ne.0) then
52            do i=mfilm,1,-1                                     ! backwards
53                x = zzz(i)*qq
54                zq(i) = cmplx (0.0, x)                          ! izq
55                ss =    cmplx (0.0, x+x) * eta(i)               ! izqn2
56                x  = exp (real (ss))
57                y  =     aimag (ss)
58                ss = cmplx (x*cos(y), x*sin(y))                 ! exp (izqn2)
59                ee(i) = ss
60                em(i) = one-ss
61                ep(i) = one+ss
62                ys(i) = eta(i)* ((em(i)*eta(i)+ep(i)*ys(i+1)) /
63       &                        (ep(i)*eta(i)+em(i)*ys(i+1)) )
64                yp(i) = cta(i)* ((em(i)*cta(i)+ep(i)*yp(i+1)) /
65       &                        (ep(i)*cta(i)+em(i)*yp(i+1)) )
66            end do
67        end if
68        Rs = (eta0-ys(1)) /(eta0+ys(1))             ! air
69        Rp = (cta0-yp(1)) /(cta0+yp(1))             ! air
70  c*    Rp = -Rp                                    ! E <---- H
71
72  c     Jacobian
73
74        do i=mfilms,1,-1                            ! backwards
75            if (i.eq.mfilms) then                   ! source substrate,    Y'
76                sse = half /eta(i)                  ! d(eta) /d(e)
77                ssa = sse                           ! d(eta) /d(-(air*sin)**2)
78                ppe = (cmplx (2.0*as2, 0.0) - die(i))
79       &              * half / (eta(i)* die(i)*die(i))      ! d(cta)/d(e)
80                ppa = half / (eta(i)*die(i))                ! d(cta)/d(-as**2)
81            else                                            ! source film
82                bot = ep(i)*eta(i) + em(i)*ys(i+1)          ! denominator,  TE
83                top = em(i)*eta(i) + ep(i)*ys(i+1)          ! numerator
84                ss  = em(i) + zq(i)*ee(i)*(ys(i+1)-eta(i))
85       &                    +  ep(i)*half*(ys(i+1)/eta(i))
86                pp  = ep(i)*half - zq(i)*ee(i)*(ys(i+1)-eta(i))
87
88                sse = (ss/bot) - (pp*top)/(bot*bot)
89                ssa = (ss + ep(i)*eta(i)*ssa) /bot -
90       &              (pp + em(i)*eta(i)*ssa)*top /(bot*bot)
91                ssz = two*ee(i)* (eta(i)**2) *              ! d/d (izq2)
92       &              (ys(i+1)-eta(i)) * (ys(i+1)+eta(i)) /(bot*bot)
93
94                bot = ep(i)*eta(i) + em(i)*yp(i+1)*die(i)   ! denominator,  TM
95                top = em(i)*eta(i) + ep(i)*yp(i+1)*die(i)   ! numerator
96                pp  = (cmplx (2.0*as2, 0.0) - die(i))
97       &              * half / (eta(i)* die(i)*die(i))      ! d(cta)/d(e)
98
99                pp1 = pp*top/bot
```

```
100               pp2 = em(i)*half/die(i) + ep(i)*cta(i)*yp(i+1) +
101      &              zq(i)*ee(i)*(yp(i+1)-cta(i))
102               pp3 = ep(i)*half/die(i) + em(i)*cta(i)*yp(i+1) -
103      &              zq(i)*ee(i)*(yp(i+1)-cta(i))
104               ppe = pp1 + pp2/bot - pp3*top/(bot*bot)
105
106               pp1 = half*top /(bot*eta(i)*die(i))
107               pp2 = em(i)*half/die(i) + ep(i)*eta(i)*ppa +
108      &              zq(i)*ee(i)*(yp(i+1)-cta(i))
109               pp3 = ep(i)*half/die(i) + em(i)*eta(i)*ppa -
110      &              zq(i)*ee(i)*(yp(i+1)-cta(i))
111               ppa = pp1 + pp2/bot - pp3*top/(bot*bot)
112
113               pp  = (eta(i)-die(i)*yp(i+1)) * (eta(i)+die(i)*yp(i+1))
114               ppz = -two*ee(i)*eta(i)*cta(i)* (pp/(bot*bot))
115            end if
116
117            if (i.ne.1) then                        ! transport,  Y'(e)
118               k = i-1
119               do j=k,1,-1                                    ! backwards
120                  sse = sse*four*ee(j)* ((eta(j)**2) /
121      &                    ((ep(j)*eta(j)+em(j)*ys(j+1))**2))
122                  ppe = ppe*four*ee(j)* ((cta(j)**2) /
123      &                    ((ep(j)*cta(j)+em(j)*yp(j+1))**2))
124               end do
125            end if
126            sse = -sse*two*eta0 /((eta0+ys(1))**2)          ! R'(e)
127            ppe = -ppe*two*cta0 /((cta0+yp(1))**2)
128
129            if (i.ne.mfilms) then
130               if (i.ne.1) then                     ! transport,  Y'(izq2)
131                  k = i-1
132                  do j=k,1,-1                                ! backward
133                     ssz = ssz*four*ee(j)* ((eta(j)**2) /
134      &                       ((ep(j)*eta(j)+em(j)*ys(j+1))**2))
135                     ppz = ppz*four*ee(j)* ((cta(j)**2) /
136      &                       ((ep(j)*cta(j)+em(j)*yp(j+1))**2))
137                  end do
138               end if
139               ssz = -ssz*two*eta0 /((eta0+ys(1))**2)       ! R'(izq2)
140               ppz = -ppz*two*cta0 /((cta0+yp(1))**2)
141            end if
142
143   c        The format of the Jacobian vector has the form:
144   c         air / film#1 / film#2 / ... / mfilm / substrate
145   c               [z,n,k,   z,n,k,   ...   z,n,k,    n,k]
146
147            k = 3*(i-1)                  ! position within Jacobian
148            if (i.eq.mfilms) then        ! substrate,   ( ,n,k)
149               k = k-1
150            else                         ! film,        (z,n,k)
151               ss = cmplx (0.0, qq*2.0)                 ! d(izq2) /dz
152   c*          ppz = -ppz                               ! E <---- H
```

```
153                 dRs(k+1) = ssz *ss                      ! R'(z)
154                 dRp(k+1) = ppz *ss                      ! R'(z)
155               end if
156               ss = sqrtt (die(i))                       ! n+ik
157   c*          ppe = -ppe                                ! E <---- H
158               dRs(k+2) = sse *ss*cmplx (2.0, 0.0)       ! R'(n)
159               dRs(k+3) = sse *ss*cmplx (0.0, 2.0)       ! R'(k)
160               dRp(k+2) = ppe *ss*cmplx (2.0, 0.0)       ! R'(n)
161               dRp(k+3) = ppe *ss*cmplx (0.0, 2.0)       ! R'(k)
162           end do
163
164   c       Since the angular partials  involved:     dR/d (-(air*sin)**2)
165   c       and we want partials wrt angle, i.e.:     dR/d (angl)     angl~radians
166
167           ss = cmplx (-air*air*sin(angl+angl), 0.0) ! d (-(air*sin)**2) /d(angl)
168           ssa = ssa*ss                              ! d(Ys) /d(angl)
169           ppa = ppa*ss                              ! d(Yp) /d(angl)
170
171           ssa = two* (eta0a*ys(1)-eta0*ssa) /((eta0+ys(1))**2)      ! R'(a)
172           ppa = two* (cta0a*yp(1)-cta0*ppa) /((cta0+yp(1))**2)
173   c*      ppa = -ppa                                ! E <---- H
174           dRsa = ssa                                ! R'(a)
175           dRpa = ppa
176
177   c       Since the TM calculation utilizes H fields, i.e.,
178   c            Rp ~ H(reflected)/H(incident),
179   c       and that convention uses E fields, i.e.,
180   c            Rp ~ E(x,reflected) / E(x,incident),
181   c       one may induce the minus sign (-) onto Rp above.
182
183           return
184           end
```

6.2.17 STAT22.FOR

```
      subroutine stat22 (n,b)
      real       b(n)
      include   'iounit.'
      logical    llzero
      data  pi / 3.1415926536 /

c     Perform:    simple statistics of deviations from the model.
c     Discern:    (mean, standard deviation)     of deviations.
c     Assume:      b(1) ' d(psi  ) ' deviation ' experiment-model
c                  b(2) ' d(delta) ' deviation ' experiment-model

      raddeg = 180.0 /pi                    ! degrees <--- radians
      nh = n/2                              ! psi, delta
      h = float (nh)

      write (iout,111)
      if (n.le.1 .or. n.ne.nh+nh) then
          write (iout,112)  n
          stop
      end if

      a1 = 0.0
      a2 = 0.0
      do i=1,n,2                            ! mean deviation
          a1 = a1 + b(i  )                  !     psi
          a2 = a2 + b(i+1)                  !     delta
      end do
      a1 = a1 /h                            !     psi
      a2 = a2 /h                            !     delta

      llzero = .false.                      ! zero variance
      s1 = 0.0
      s2 = 0.0
      s3 = 0.0
      do i=1,n,2                            ! variance of deviations
          s1 = s1 + (b(i  )-a1)**2          !     <psi  |  psi>
          s2 = s2 + (b(i+1)-a2)**2          !     <delta|delta>
          s3 = s3 + (b(i+1)-a2)*(b(i)-a1)   !     <delta|  psi>
      end do
      s1 = sqrt (s1 /h)                     ! standard deviation
      s2 = sqrt (s2 /h)
      s3 =       s3 /h                      ! covariance

      if (s1.eq.0.0 .or. s2.eq.0.0) then    ! exact fit ' no scatter
          llzero = .true.
      else
          s3 = s3 /(s1*s2)                  ! correlation coefficient
      end if

      a1 = a1 *raddeg                       ! degrees
```

```
51        a2 = a2 *raddeg
52        s1 = s1 *raddeg
53        s2 = s2 *raddeg
54
55        write (iout,113)  a1,s1,  a2,s2,  s3
56        if (llzero)  write (iout,114)
57
58        return
59
60    111 format (/  1x, 15('----')
61       &        /' Statistics of deviations ' experiment-model ' g'
62       &       //'    where:    g ' column array of length ' 2M   '
63       &        /'      let:   () ' (psi or delta) ' (1 or 2)     '
64       &       //'        mean () = m() = <g()> = (1/M) sum: g()  '
65       &        /'    variance () = < [g( )-m( )]**2 >            '
66       &        /'  covariance    = < [g(1)-m(1)]*[g(2)-m(2)] >   '
67       &        /'        std dev = sqrt (variance)               '
68       &        /'  correlat coef = covariance / [std dev (psi) * ',
69       &                                        'std dev (delta)]' )
70
71    112 format (/' stat22,   ... oops ' inconsistency, n=', i5)
72
73    113 format (/  20x, ' mean,',  4x, ' std dev     (degrees)'
74       &        /   5x, '  psi:',  4x, f10.3, 3x, f10.3
75       &        /   5x, 'delta:',  4x, f10.3, 3x, f10.3
76       &        /   5x,     6x,    4x, f10.3, 1x,
77       &              '' correlation coefficient ' <psi|delta>    ')
78    114 format (   26x, '' UNnormalized, because atleast one of the '
79       &        /  26x, '                standard deviations vanish.'/)
80
81        end
```

6.2.18 CORLAT.FOR

```
      subroutine corlat
      include   'iounit.'
      include   'defnit.'
      include   'filmmm.'
      include   'wstack.'
      include   'cgnxx1.'

      dimension  det(2), inert(3)                ! LINPACK

      raddeg = 180.0 / 3.14159265

c     Discern variance of model parameters, normalized,  A(T)*A.

      llnorm = .false.
      call asmbl                                 ! ia,ja,aa,bb
      call stat22 (meqns, bb)                    ! mean, std dev.
      call pltdat (2)                            ! plot deviations of fit

      kv = 0
      do jv=1,mvary                              ! A(T)*A,        parameters
          do iv=1,jv                             ! upper triangle + diagonal
              sum1 = 0.0
              sum2 = 0.0
              do i=1,meqns,2
                  j1 = ia(i  )
                  j2 = ia(i+1)-1
                  mv = j2-j1+1
                  s1 = 0.0
                  s2 = 0.0
                  s3 = 0.0
                  s4 = 0.0
                  do j=j1,j2                     ! within a row of A
                      jaj = ja(j)                ! column
                      if (jaj.eq.iv) then
                          s1 = aa(j   )
                          s3 = aa(j+mv)
                      end if
                      if (jaj.eq.jv) then
                          s2 = aa(j   )
                          s4 = aa(j+mv)
                      end if
                  end do
                  sum1 = sum1 + s1*s2            !   psi ' dot product
                  sum2 = sum2 + s3*s4            ! delta
              end do
              sum = sum1+sum2
              sum = sum /float (meqns)           ! normalization
              kv = kv+1                          ! packed format ' storage
              aat (kv) = sum                     ! upper triangle + diagonal
              if (jv.eq.iv) then                 ! diagonal
```

```
51                  xx(jv) = sqrt (sum)
52              end if
53          end do
54      end do
55      kv = 0
56      do jv=1,mvary                                   ! renormalize
57          do iv=1,jv
58              kv = kv+1
59              aat(kv) = aat(kv) / (xx(jv) * xx(iv))
60          end do
61      end do
62
63      write (iout,111)
64      k2 = 0
65      do i=1,mvary
66         k1 = k2+1
67         k2 = k2+i
68         write (iout,114)  i, (aat(k),k=k1,k2)
69      end do
70      write (iout,120)
71      write (iout,121)  (xx(i), i=1,mvary)        ! normalization coefficients
72
73  c   LINPACK, Chapter 5, Solving symmetric indefinite matrices.
74      call sspco (aat, mvary, ipvt, rcond, w)     ! UD*T(U)  decomposition
75      write (iout,116)  rcond
76
77      if (.true.) return      ! <-----------------------------------
78
79  c*  if (1.0 .eq. 1.0+rcond     ) then           ! singular matrix
80      if (1.0 .eq. 1.0+rcond*0.01) then           ! ill-conditioning
81          call sspdi (aat, mvary, ipvt, det, inert, w, 111)
82          write (iout,117)  det, inert
83          k2 = 0
84          do i=1,mvary
85             k1 = k2+1
86             k2 = k2+i
87             write (iout,112)  i, (aat(k), k=k1,k2)
88          end do
89      end if
90
91  c   ======================================================
92  c   Alternate representation of correlation,  normalized.
93
94      do jv=1,mvary                                   ! mean ~  <A(,j)>
95          sum = 0.0
96          do i=1,meqns,2
97              j1 = ia(i  )
98              j2 = ia(i+1)-1
99              mv = j2-j1+1
100             do j=j1,j2                              ! row of A
101                 jaj = ja(j)                         ! column
102                 if (jaj.eq.jv) then
103                     sum = sum + aa(j)+aa(j+mv)
```

```
104                 end if
105             end do
106         end do   ! meqns
107         sum = sum/float (meqns)
108         v(jv) = sum
109     end do       ! mvary
110
111     kv = 0                                      ! covariance
112     do jv=1,mvary                               ! <(x-<x>)*(y-<y>)>
113         do iv=1,jv
114             sum = 0.0
115             do i=1,meqns,2
116                 j1 = ia(i  )
117                 j2 = ia(i+1)-1
118                 mv = j2-j1+1
119                 s1 = 0.0
120                 s2 = 0.0
121                 s3 = 0.0
122                 s4 = 0.0
123                 do j=j1,j2                      ! row
124                     jaj = ja(j)                 ! column
125                     if (jaj.eq.jv) then
126                         s1 = aa(j   )-v(jv)
127                         s3 = aa(j+mv)-v(jv)
128                     end if
129                     if (jaj.eq.iv) then
130                         s2 = aa(j   )-v(iv)
131                         s4 = aa(j+mv)-v(iv)
132                     end if
133                 end do
134                 sum = sum + s1*s2 + s3*s4
135             end do    ! meqns
136             sum = sum/float (meqns-1)           ! <(x-<x>)*(y-<y>)>
137             kv = kv+1
138             aat(kv) = sum
139             if (iv.eq.jv) then                  ! diagonal
140                 xx(jv) = sqrt (sum)
141             end if
142         end do
143     end do
144     kv = 0                                      ! renormalize
145     do jv=1,mvary                               ! correlation
146         do iv=1,jv
147             kv = kv+1
148             if (xx(iv).eq.0.0  .or.  xx(jv).eq.0.0) then
149                 aat(kv) = -2.0
150             else
151                 aat(kv) = aat(kv) / (xx(iv)*xx(jv))
152             end if
153         end do
154     end do
155
156     write (iout,118)                            ! Print out results
```

```
157         k2 = 0
158         do i=1,mvary
159            k1 = k2+1
160            k2 = k2+i
161            write (iout,114)  i, (aat(k),k=k1,k2)
162         end do
163         write (iout,120)
164         write (iout,121)  (xx(i), i=1,mvary)       ! normalization coefficients
165
166         return
167
168      111 format (/' J(T)*J:        (renormalized for correlation)')
169      112 format (1x, i4, ')', 1x, 1p10e10.2, : /(7x, 10e10.2))
170      114 format (1x, i4, ')', 1x,   10f10.5, : /(7x, 10f10.5))
171      116 format (/' rcond=', 1pe12.3, ',  condition number')
172      117 format (/' J(T)*J:        (renormalized for correlation)'
173         &         /'     Determinant:    ', f8.4, ' E ', f8.4
174         &         /'     Inertia:       (', 3i4,
175         &                    '),    number of (+,-,0) eigenvalues'
176         &         /'     Inverse:    upper+diagonal  matrix')
177
178      118 format (/' J(T)*J:      ' <(x-<x>)*(y-<y>)> ')
179      120 format (/' Normalization coefficients:     sqrt [J(T)*J](i,i)')
180      121 format ( 1x, 1p10e10.2)
181         end
```

6.2.19 SEAMA.FOR

```
      subroutine  seama
      include    'iounit.'
      include    'defnit.'
      include    'filmmm.'
      include    'seamx1.'
      include    'wstack.'
      include    'cgnxx1.'              !  w,xw    ' convenience

      real        det(2),  ss(4)
      integer     inert(3)
      data        pi / 3.14159265 /

c     ---------------------------------------------------------------
c     Sensitivity analysis for multiple:  angle,ambient,wave,sample.
c     Ordering of indices in IPTU:     [vary-->      <--froz]

c     The matrix equation is:      g = AV*v + AU*u + AT*t
c     where:   v ' vary,
c              u ' frozen,
c              t ' d(phi ' angle of incidence)
c              g ' deviations ' experiment - model
c     ---------------------------------------------------------------

      raddeg = pi/180.0                    ! radians <-- degrees
      mm = mlmnts+mfilmm                   ! model parameters
      llnorm = .false.
      call asmblx

c     Renormalize the columns of AA,  and retain factors in AATS.
      call scaljj (meqns,mvary, ia,ja,aa,xx, aats,w,2) ! aa,aats

c     Formulate the "Normal" equations.
c     Note that:     J(v,T)*J(v)  ' symmetric matrix.
c     Construct:     upper triangle,  i<= j.

      do jv=1,mvary
          do iv=1,jv                              !  upper triangle
              s = 0.0
              do i=1,meqns,2
                  j1 = ia(i  )
                  j2 = ia(i+1)-1
                  mv = j2-j1+1
                  s1 = 0.0
                  s2 = 0.0
                  s3 = 0.0
                  s4 = 0.0
                  do j=j1,j2
                      jaj = ja(j)              ! column
                      if (jaj.eq.iv) then
                          s1 = aa(j   )        ! psi
```

```
51                          s3 = aa(j+mv)          ! delta
52                      end if
53                      if (jaj.eq.jv) then
54                          s2 = aa(j   )          ! psi
55                          s4 = aa(j+mv)          ! delta
56                      end if
57                  end do
58                  s = s + s1*s2 + s3*s4
59              end do
60              ij = iindex (3,mvary, iv,jv)       ! column-wise
61              aat(ij) = s                        ! upper triangle
62          end do              ! rows
63      end do                  ! columns
64
65      gggpsi = 0.0                               ! maximum magnitude
66      gggdel = 0.0
67      gggang = 0.0
68      do ii=1,meqns,2
69          ngl = (ii+1)/2
70          gggpsi = amax1 (gggpsi, abs (   bb(ii  )))
71          gggdel = amax1 (gggdel, abs (   bb(ii+1)))
72          gggang = amax1 (gggang, abs (angleu(ngl)))  ! radians
73      end do
74      ggpsi = 0.0                                ! estimate variances
75      ggdel = 0.0
76      ggang = 0.0
77      do ii=1,meqns,2                              ! rescale
78          ngl = (ii+1)/2                           ! index uncertainties
79          ggpsi = ggpsi + (   bb(ii  )/gggpsi)**2  ! psi      deviations
80          ggdel = ggdel + (   bb(ii+1)/gggdel)**2  ! delta    deviations
81          ggang = ggang + (angleu(ngl)/gggang)**2  ! anglei   radians
82      end do
83      ggpsi = ggpsi* gggpsi**2
84      ggdel = ggdel* gggdel**2
85      ggang = ggang* gggang**2
86
87      ggvar = (ggpsi+ggdel) /float (meqns-mvary) ! <gg>     variance
88      ggang = ggang /float (meqns/2)             ! <aa>
89      ggvars = sqrt (ggvar)                      ! standard deviation
90      ggangs = sqrt (ggang)
91
92      write (iout,111)
93      write (iout,126)  ggvar,ggvars,  ggang,ggangs
94      write (iout,112)
95      write (iout,121)
96  c   write (iout,112)
97
98      call sspco (aat,mvary,ipvt,rcond, xx)      ! UD*U(T)
99
100 c    if (1.0 .eq. 1.0+rcond     ) then
101     if (1.0 .eq. 1.0+rcond*0.01) then          ! ill-conditioned
102         write (iout,101)
103         return
```

```
104         end if
105
106         call sspdi (aat,mvary,ipvt,det,inert,xx,1) ! inverse
107
108         cc1xxx = 0.0                               !   psi,  max magnitude
109         cc2xxx = 0.0                               ! delta,  max magnitude
110         do ii=1,meqns,2
111             cc1xxx = amax1 (cc1xxx, abs (cc(ii  )))
112             cc2xxx = amax1 (cc2xxx, abs (cc(ii+1)))
113         end do
114
115         do jv=1,mvary                              ! column
116             bbb1xx = 0.0                           !   psi, max magnitude
117             bbb2xx = 0.0                           ! delta, max magnitude
118             do ii=1,meqns,2                        ! [A(T)*A]**-1 *A(T)
119                 j1 = ia(ii  )
120                 j2 = ia(ii+1)-1
121                 mv = j2-j1+1
122                 s1 = 0.0
123                 s2 = 0.0
124                 do jj=j1,j2                        ! row
125                     jaj = ja(jj)                   ! column
126                     k  = iindex (3,mvary, jv,jaj)
127                     s1 = s1 + aat(k)*aa(jj   )     ! psi
128                     s2 = s2 + aat(k)*aa(jj+mv)     ! delta
129                 end do
130                 bbb(ii  ) = s1                     ! psi
131                 bbb(ii+1) = s2                     ! delta
132                 bbb1xx = amax1 (bbb1xx, abs (s1))  ! maximum
133                 bbb2xx = amax1 (bbb2xx, abs (s2))
134             end do
135
136             write (iout,112)
137
138             do iv=1,jv                             ! row
139                 bb1xxx = 0.0
140                 bb2xxx = 0.0
141                 do ii=1,meqns,2                    ! [A(T)*A]**-1 *A(T)
142                     j1 = ia(ii  )
143                     j2 = ia(ii+1)-1
144                     mv = j2-j1+1
145                     s1 = 0.0
146                     s2 = 0.0
147                     do jj=j1,j2                    ! row
148                         jaj = ja(jj)               ! column
149                         k  = iindex (3,mvary, iv,jaj)
150                         s1 = s1 + aat(k)*aa(jj   ) ! psi
151                         s2 = s2 + aat(k)*aa(jj+mv) ! delta
152                     end do
153                     bb(ii  ) = s1                  ! psi     Note:
154                     bb(ii+1) = s2                  ! delta   overwriting
155                     bb1xxx = amax1 (bb1xxx, abs (s1))
156                     bb2xxx = amax1 (bb2xxx, abs (s2))
```

```
157                 end do
158
159                 do j=1,4                              ! convenience
160                     ss(j) = 0.0                       ! initialize
161                 end do
162                 do ii=1,meqns,2
163                     ss(1) = ss(1) + (bbb(ii  )/bbb1xx)
164       &                           * ( bb(ii  )/bb1xxx)       ! psi
165                     ss(2) = ss(2) + (bbb(ii+1)/bbb2xx)
166       &                           * ( bb(ii+1)/bb2xxx)       ! delta
167                     ss(3) = ss(3) + (bbb(ii  )/bbb1xx)
168       &                           * ( bb(ii  )/bb1xxx)
169       &                           * ( cc(ii  )/cc1xxx)**2    ! psi
170                     ss(4) = ss(4) + (bbb(ii+1)/bbb2xx)
171       &                           * ( bb(ii+1)/bb2xxx)
172       &                           * ( cc(ii+1)/cc2xxx)**2    ! delta
173                 end do
174                 ss(1) = ss(1)*bbb1xx*bb1xxx
175                 ss(2) = ss(2)*bbb2xx*bb2xxx
176                 ss(3) = ss(3)*bbb1xx*bb1xxx* cc1xxx**2
177                 ss(4) = ss(4)*bbb2xx*bb2xxx* cc2xxx**2
178
179                 ss(1) = (ss(1) + ss(2))*ggvar             !  BggB
180                 ss(3) = (ss(3) + ss(4))*ggang             ! BJaaJB
181
182                 ss(1) = ss(1) /(aats(jv)*aats(iv))        ! scale
183                 ss(3) = ss(3) /(aats(jv)*aats(iv))
184
185                 ss(2) = sqrt (abs (ss(1)))                ! ' std dev
186                 ss(4) = sqrt (abs (ss(3)))
187
188                 write (iout,131)  jv,iv,  ss(1),ss(3),  ss(2),ss(4)
189             end do   ! iv
190
191             random = sqrt (ss(1) + ss(3))          ! diagonal, (j,j)
192             system = 0.0
193
194             if (mfroz.ne.0) then                   ! systematic errors
195                 do k=1,mfroz
196                     xxx(k) = 0.0                   ! initialize
197                 end do
198                 do ii=1,meqns,2
199                     j1 = iaa(ii  )
200                     j2 = iaa(ii+1)-1
201                     if (j1.le.j2) then
202                         mu = j2-j1+1
203                         do j=j1,j2
204                             km = jaa(j)            ! backwards
205                             k  = mm+1-km           ! column
206                             xxx(k) = xxx(k)
207       &                           + bbb(ii  )*aaa(j   )
208       &                           + bbb(ii+1)*aaa(j+mu)
209                         end do
```

```
210                     end if      ! row
211                 end do          ! meqns
212
213                 do k=1,mfroz
214                     km = mm+1-k                     ! backwards
215                     m  = iptu(km)
216                     if (m.le.mlmnts) then
217                         uncert = uncerl(m)
218                     else
219                         m = m-mlmnts
220                         uncert = uncerz(m)
221                     end if
222                     xxx(k) = abs (xxx(k)*uncert)    !  |BJu|
223                     xxx(k) = xxx(k) /aats(jv)       !  scale
224                     system = system + xxx(k)
225                 end do
226             end if        ! frozen
227
228 c           Output results ------------------------
229
230             write (iout,113)
231
232             total = random + system                 ! diagonal
233             m = iptu(jv)                            ! unique,nonlocal,full
234             if (m.le.mlmnts) then
235                 s1 = diefcn(m)
236                 s2 = uncerl(m)
237                 write (iout,122)  jv, total, s2, s1, m, '(n+ik)'
238                 write (iout,123)  random, ss(2), ss(4)
239             else
240                 m = m-mlmnts
241                 s1 = widths(m)
242                 s2 = uncerz(m)
243                 write (iout,122)  jv, total, s2, s1, m, '(z   )'
244                 write (iout,123)  random, ss(2), ss(4)
245             end if
246
247             if (mfroz.ne.0) then                    ! systematic errors
248                 write (iout,124)  system
249                 write (iout,110)
250                 do k=1,mfroz
251                     km = mm+1-k
252                     m = iptu(km)
253                     if (m.le.mlmnts) then
254                         write (iout,125)  k, xxx(k), m, '(n+ik)'
255                     else
256                         m = m-mlmnts
257                         write (iout,125)  k, xxx(k), m, '(z   )'
258                     end if
259                 end do
260             end if
261
262         end do          ! vary
```

```
263
264         write (iout,111)
265         return
266
267     100 format (/' seama,  insufficient allocation for array, AAT')
268     101 format (/' seama,  singular or ill-conditioned matrix ')
269     102 format (/' seama,  uncertainties')
270
271     110 format (' ')
272     111 format ( 1x, 17('===='))
273     112 format ( 1x, 17('----'))
274     113 format (13x, 14(' - -'))
275
276     121 format ( ' Discern:    Uncertainty      in model parameters  '
277        &    /'   where:    |v| = sqrt( BB(T) <gg> + (BJ)(BJ)(T) <aa>)'
278        &    /'                                  +  |BJ||u|,        '
279        &   //'             B ' [J(T)*J]**-1 *J(T),      (nonsquare J)'
280        &   //'    vary',  6x, 'total ', 4x, 'initial', 4x, 'parameter')
281
282     122 format ( 3x, i4,  ')',  2x,  1p2e11.2,  e15.6,
283        &                           ', for:',  i4,  ', ',  a)
284     123 format ( 10x,  1p3e11.2,  4x, 'random, |Bg|, |BJa|')
285     124 format ( 21x,  1p1e11.2,  4x, 'systematic ' total ')
286     125 format ( 10x,    6x, i4, ')',
287        &              1p1e11.2,  4x, 'systematic ' |BJu| '',i4,', ',a)
288
289     126 format (' <gg> ' ', 1p2e12.3, 4x, '(variance, std dev)'
290        &       /' <aa> ' ',   2e12.3, 4x, '(variance, std dev)')
291
292     131 format ( 10x, 2i5, 1p2e11.2, 4x, ' (j,i), |BggB|,  |BJaaJB|'
293        &       / 20x,        2e11.2, 4x, '        |Bg|,    |BJa|   ')
294         end
```

6.2.20 SEAMAX.FOR

```
c     ------------------------------------------------------------
c     Note:    This routine should be compared to:    SEAMA
c     Here:    Methods   2,3    are not available.
c              There are two estimates of errors, but
c              NO use is made of that retained in:    xw
c     ------------------------------------------------------------

      subroutine  seamax
      include    'iounit.'
      include    'defnit.'
      include    'filmmm.'
      include    'seamx1.'
      include    'seamx2.'
      include    'cgnxx1.'
      include    'wstack.'

      logical     square, lsout
      integer     inert(3),                ks(mrowss*2+nrowss)
      real        det(2), ss(4), sss(4,2), sk(mrowss*2+nrowss)

      data  pi / 3.14159265 /

c     Sensitivity/error analysis for multiple:   angle,ambient,wave,sample.
c     Ordering of indices in IPTU:     [vary-->      <--froz]
      call arrang

      raddeg = pi/180.0                          ! radians <-- degrees
      mm = mlmnts+mfilmm                         ! model parameters
      call seam2                                 ! construct matrix table

      lsout = maraws.le.2                        ! single/double angle
      if (lsout) then                            ! output header cards
          open (unit=isout, file='x.sout', status='unknown')
          write (isout,104)  msampl, mvary, ndegr
          write (isout,104)  (nnwave(is), is=1,msampl)
          iws = 0
          iaws = 0
          iraws = 0
          do is=1,msampl
              mwave = nnwave(is)
              do iw=1,mwave
                  iws = iws+1
                  mbien = nnbent(iws)
                  write (isout,104) mbien
                  do mbn=1,mbien
                      iaws = iaws+1
c*                    mrpeat = nnpeat(iaws)
c*                    do irpeat=1,mrpeat
                          iraws = iraws+1
                          istep = isteps(iraws)
```

```
51                              mangl = mangle(iraws)
52                              write (isout,104) istep,mangl
53  c*                      end do      ! repeat
54                      end do          ! ambient
55                  end do              ! wave
56              end do                  ! sample
57          end if
58          call poplat (xx, 1)                         ! initialize
59  c       ================================================================
60  c       Provide:   do-loop nesting to the necessary depth.
61  c       Utilize:   goto, if,      and three vectors.
62  c       (angle | ambient,wave,sample),     i.e., without repeats.
63  c       For each distinct set of (ambient,wave,sample),
64  c       scan a set of multiple angles,
65  c       where each set of multiple angles are of the form:
66  c                  do i(1)=1+(manglm-1)*step,  ndegr    , step
67  c                  do i(2)=1+(manglm-2)*step,  i(1)-step, step
68  c                  do i(3)=1+(manglm-3)*step,  i(2)-step, step
69  c                                  ...
70  c                  do i(manglm)=1,      i(manglm-1)-step, step
71
72          kt = 0                                      ! index counter
73          kts = 0                                     ! index singular events
74
75          j2 = 0                                      ! (angle,ambient,wave,sample)
76          k  = 0                                      ! (      ambient,wave,sample)
77        1 k = k+1
78          if (k.gt.mraws) goto 4                      ! work
79          mangl = mangle(k)                           ! multiple angle
80          istep = isteps(k)                           ! do-loop increment
81          j1 = j2+1
82          j2 = j2+mangl
83          iii2(j1) = ndegr                            ! limit outer do-variable
84          do j=j1,j2                                  ! multiple angle
85              iii1(j) = 1 + (j2-j)*istep              ! fixed, first do-parameter
86              iiii(j) = iii1(j) - istep               ! initialize   do-variable
87          end do
88
89          j = j1-1                                    ! index of:  nested do-variables.
90        2 j = j+1                                     ! index of:  j-th   nested do.
91          if (j.gt.j2) goto 1
92        3 iiii(j) = iiii(j) + istep                   ! update do-loop variable
93          if (iiii(j) .le. iii2(j)) then              ! test upper limit
94              if (j.ne.j2)  iii2(j+1)=iiii(j)-istep
95              goto 2
96          end if
97          iiii(j) = iii1(j) - istep                   ! reset inner do
98          j = j-1                                     ! backup one do-level
99          if (j.ge.j1) goto 3
100
101         if (k.eq.1) goto 6                          ! escape do-loop nest
102         k = k-1                                     ! backup
103         mangl = mangle(k)
```

```
104         istep = isteps(k)
105         j2 = j1-1
106         j1 = j1-mangl
107         goto 3
108       4 continue                              ! inner-most nested do.
109
110         j1save = j1
111         j2save = j2
112         if (j.ne.maraws+1) then
113             write (iout,107)  maraws, j
114             stop
115         end if
116
117   c     write (iout,102)  kt,  (iiii(j), j=1,maraws)
118   c     if (.true.) goto 5
119   c     ---------------------------------------------------------------
120   c     The matrix equation is:      b = AV*v + AU*u + AT*t
121   c     where:    v ' vary,
122   c               u ' frozen,
123   c               t ' d(phi ' angle of incidence)
124   c               b ' deviations ' experimental uncertainty
125   c     The method of solution for |v|,
126   c     depends on whether the matrix [AV] is square.
127
128         call seam3                                  ! model experiment, table
129
130         square = meqns.eq.mvary
131
132         if (.not.square) then                       ! renormalize columns
133             call scaljj (meqns,mvary, ia,ja,aa,xx, aats,w,2) ! aa,aats
134         end if
135
136         if (square) then                            ! non-symmetric matrix
137             mvaryy = mvary*mvary                    ! full, square
138             if (mvaryy .gt. naat) then              ! WSTACK, allocation
139                 write (iout,100)
140                 stop
141             end if
142             do i=1,mvaryy                           ! square matrix
143                 aat(i) = 0.0                        ! initialize
144             end do
145             do i=1,meqns,2                          ! LINPACK format
146                 j1 = ia(i  )
147                 j2 = ia(i+1)-1
148                 mv = j2-j1+1
149                 do j=j1,j2                          ! row
150                     jaj = ja(j)                     ! column
151                     kpsi = iindex (5,mvary, i  ,jaj)    ! stored column-wise
152                     kdel = iindex (5,mvary, i+1,jaj)
153                     aat(kpsi) = aa(j   )            ! psi
154                     aat(kdel) = aa(j+mv)            ! delta
155                 end do
156             end do
```

```
157           call sgeco (aat,mvary,mvary,ipvt,rcond, xx)      ! LU decomposition
158   c       if (1.0 .eq. 1.0+rcond     ) then                ! singular matrix
159           if (1.0 .eq. 1.0+rcond*0.01) then                ! ill-conditioned
160   c           write (iout,101)  (iiii(i), i=1,maraws)
161               kts = kts+1
162               do i=1,mvary
163                   xx(i) = -1.0                             ! truncate
164               end do
165           else
166               call sgedi (aat,mvary,mvary,ipvt,det, xx,1)  ! inverse
167               do i=1,mvary                                 ! row
168                   do j=1,2
169                       do kj=1,4
170                           sss(kj,j) = 0.0                  ! initialize
171                       end do
172                   end do
173                   do j=1,meqns,2                           ! (psi,delta)
174   c*                  uang = 0.002*raddeg                  ! radians, SEAM2
175
176                       kpsi = iindex (5,mvary, i,j  )
177                       kdel = iindex (5,mvary, i,j+1)
178
179                       ss(1) = aat(kpsi)*bb(j  )            !   psi
180                       ss(2) = aat(kdel)*bb(j+1)            ! delta
181                       ss(3) = aat(kpsi)*cc(j  )*uang       !   psi'
182                       ss(4) = aat(kdel)*cc(j+1)*uang       ! delta'
183                       do kj=1,4
184                           sss(kj,1) = sss(kj,1) + ss(kj)
185                           sss(kj,2) = sss(kj,2) + ss(kj)**2
186                       end do
187                   end do
188                   ss(1) =  abs (sss(1,1) + sss(2,1))       !    |Dg |
189         &                +  abs (sss(3,1) + sss(4,1))       !    |DJt|
190                   ss(2) = sqrt (sss(1,2) + sss(2,2))       ! rms(Dg )
191         &               + sqrt (sss(3,2) + sss(4,2))       ! rms(DJt)
192
193                   if (mfroz.ne.0) then                     ! frozen
194                       do k=1,mfroz
195                           xxx(k) = 0.0                     ! initialize
196                       end do
197                       do ii=1,meqns,2                      ! even
198                           j1 = iaa(ii  )
199                           j2 = iaa(ii+1)-1
200                           if (j1.le.j2) then
201                               mu = j2-j1+1
202                               do j=j1,j2                   ! row
203                                   km = jaa(j)              ! backwards
204                                   k  = mm+1-km             ! column
205                                   kpsi = iindex (5,mvary, i,ii  )
206                                   kdel = iindex (5,mvary, i,ii+1)
207                                   xxx(k) = xxx(k)
208         &                                 + aat(kpsi)*aaa(j   )
209         &                                 + aat(kdel)*aaa(j+mu)
```

```
                            end do
                        end if
                    end do          ! meqns
                    do k=1,mfroz
                        km = mm+1-k                         ! backwards
                        m  = iptu(km)
                        if (m.le.mlmnts) then
                            uncert = uncerl(m)
                        else
                            m = mlmnts-m
                            uncert = uncerz(m)
                        end if
                        uncert = uncert * abs (xxx(k))   !  |DJ||u|
                        ss(1) = ss(1) + uncert
                        ss(2) = ss(2) + uncert
                    end do
                end if     ! frozen

                xx(i) = ss(1)
                xw(i) = ss(2)
            end do         ! vary
        end if
        goto 5
    end if                                      ! square matrix
c   ------------------------------------------------------------------

    if (method.eq.1) then                       ! "Normal" equations
        do jv=1,mvary                           !  A(T)*A, symmetric
            do iv=1,jv                          !  upper triangle, i<=j
                s = 0.0
                do i=1,meqns,2
                    j1 = ia(i  )
                    j2 = ia(i+1)-1
                    mv = j2-j1+1
                    s1 = 0.0
                    s2 = 0.0
                    s3 = 0.0
                    s4 = 0.0
                    do j=j1,j2
                        jaj = ja(j)             ! column
                        if (jaj.eq.iv) then
                            s1 = aa(j   )
                            s3 = aa(j+mv)
                        end if
                        if (jaj.eq.jv) then
                            s2 = aa(j   )
                            s4 = aa(j+mv)
                        end if
                    end do
                    s = s + s1*s2 + s3*s4
                end do
                ij = iindex (3,mvary, iv,jv)            ! column-wise
                aat(ij) = s                             ! upper triangle
```

```
263               end do
264             end do
265             call sspco (aat,mvary,ipvt,rcond, xx)            ! UD*U(T)
266             if (1.0 .eq. 1.0+rcond*0.01) then                ! ill-conditioned
267   c             write (iout,101)  (iiii(i), i=1,maraws)
268                 kts = kts+1
269                 do i=1,mvary
270                     xx(i) = -1.0                             ! truncate
271                 end do
272             else
273                 call sspdi (aat,mvary,ipvt,det,inert,xx,1)   ! inverse
274                 do i=1,mvary
275                     do ii=1,meqns,2              ! [A(T)*A]**-1 *A(T)
276                         j1 = ia(ii  )
277                         j2 = ia(ii+1)-1
278                         mv = j2-j1+1
279                         s1 = 0.0
280                         s2 = 0.0
281                         do jj=j1,j2                          ! row
282                             jaj = ja(jj)                     ! column
283                             k  = iindex (3,mvary, i,jaj)
284                             s1 = s1 + aat(k)*aa(jj   )       ! psi
285                             s2 = s2 + aat(k)*aa(jj+mv)       ! delta
286                         end do
287                         bbb(ii  ) = s1                       ! psi
288                         bbb(ii+1) = s2                       ! delta
289                     end do
290                     do jj=1,2
291                         do kj=1,4
292                              sss(kj,jj) = 0.0                ! initialize
293                         end do
294                     end do
295                     do ii=1,meqns,2
296   c*                    uang = 0.002*raddeg                  ! radians, SEAM2
297                         ss(1) = bbb(ii  )*bb(ii  )
298                         ss(2) = bbb(ii+1)*bb(ii+1)
299                         ss(3) = bbb(ii  )*cc(ii  )*uang
300                         ss(4) = bbb(ii+1)*cc(ii+1)*uang
301                         do kj=1,4
302                             sss(kj,1) = sss(kj,1) + ss(kj)
303                             sss(kj,2) = sss(kj,2) + ss(kj)**2
304                         end do
305                     end do
306                     ss(1) =  abs (sss(1,1) + sss(2,1))       !    |Dg |
307         &                 +  abs (sss(3,1) + sss(4,1))       !    |DJt|
308                     ss(2) = sqrt (sss(1,2) + sss(2,2))       ! rms(Dg )
309         &                 + sqrt (sss(3,2) + sss(4,2))       ! rms(DJt)
310
311                     if (mfroz.ne.0) then
312                         do k=1,mfroz
313                             xxx(k) = 0.0
314                         end do
315                         do ii=1,meqns,2
```

```
316                             j1 = iaa(ii  )
317                             j2 = iaa(ii+1)-1
318                             if (j1.le.j2) then
319                                 mu = j2-j1+1
320                                 do j=j1,j2
321                                     km = jaa(j)              ! backwards
322                                     k  = mm+1-km             ! column
323                                     xxx(k) = xxx(k)
324       &                                    + bbb(ii  )*aaa(j   )
325       &                                    + bbb(ii+1)*aaa(j+mu)
326                                 end do
327                             end if
328                         end do
329                         do k=1,mfroz
330                             km = mm+1-k                      ! backwards
331                             m  = iptu(km)
332                             if (m.le.mlmnts) then
333                                 uncert = uncerl(m)
334                             else
335                                 m = m-mlmnts
336                                 uncert = uncerz(m)
337                             end if
338                             uncert = uncert * abs (xxx(k))   !  |DJ||u|
339                             ss(1) = ss(1) + uncert
340                             ss(2) = ss(2) + uncert
341                         end do
342                     end if     ! frozen
343
344                     xx(i) = ss(1)
345                     xw(i) = ss(2)
346                 end do         ! vary
347             end if
348             goto 5
349         end if
350 c       ------------------------------------------------------------------
351
352         if (method.eq.2) then           ! Brute force, forward problem
353             if (.true.) stop            ! Note:   xw ' not available
354
355             do i=1,mvary                ! initialize magnitudes of
356                 xx(i) = 0.0             !     stored uncertainties
357             end do
358
359 c           Simulate do-loops to scan possible sign-flips.
360 c           There is a sign-flip for each component in:  b,t,u.
361 c           For the case of systematic error in the angle of incidence,
362 c           i.e., d(phi) ' t,         then 't' is scalar, not vector.
363
364 c           Simulate:    do i1=1,2
365 c                         do i2=1,2
366 c                          do i3=1,2
367 c                           ...
368 c                            do i(meqns+1+mfroz)
```

```
369
370  c          This involves:   2**(meqns+1+mfroz)      sign permutations.
371  c          Unfortunately, this becomes unwieldly rather quickly.
372
373             ns = meqns+1+mfroz            ! depth of do-nest
374             do i=1,ns
375                 ks(i) = 0                 ! i1-i3,  do-variable, initialize
376             end do
377             ms = 0
378       21    ms = ms+1
379             if (ms.gt.ns) goto 23
380       22    ks(ms) = ks(ms) + 1           ! update do-variable
381             if (ks(ms).le.2) then         ! test uper limit
382                 if (ks(ms).eq.1) then     ! specify sign convention
383                     sk(ms) =  1.0
384                 else
385                     sk(ms) = -1.0
386                 end if
387                 goto 21
388             end if
389             ks(ms) = 0                    ! reset inner do-loop
390             ms = ms-1                     ! backup one level of do-s
391             if (ms.eq.0) goto 24          ! escape do-nest
392             goto 22
393       23    ms = ms-1                     ! level of deepest do
394  c          ------------------------------
395             uangsk = uang*sk(meqns+1)     ! systematic error
396
397             do i=1,meqns,2                ! form 'b' vector
398                 bbb(i  ) = bb(i  )*sk(i  ) - cc(i  )*uangsk       !   psi
399                 bbb(i+1) = bb(i+1)*sk(i+1) - cc(i+1)*uangsk       ! delta
400                 j1 = iaa(i  )             ! frozen
401                 j2 = iaa(i+1)-1
402                 if (j1.le.j2) then
403                     mu = j2-j1+1
404                     s1 = 0.0
405                     s2 = 0.0
406                     do j=j1,j2
407                         km = jaa(j)       ! backwards
408                         k  = mm+1-km
409                         m  = iptu(km)
410                         if (m.le.mlmnts) then
411                             uncert = uncerl(m)
412                         else
413                             m = m-mlmnts
414                             uncert = uncerz(m)
415                         end if
416                         uncert = uncert*sk(meqns+1+k)
417                         s1 = s1 + aaa(j   )*uncert
418                         s2 = s2 + aaa(j+mu)*uncert
419                     end do
420                     bbb(i  ) = bbb(i  ) - s1
421                     bbb(i+1) = bbb(i+1) - s2
```

```
422               end if
423           end do                          ! meqns
424           do i=1,mvary
425               xxx(i) = 0.0                ! initialize,  CGNL
426           end do
427
428           niter = mvary*4
429           call cgnl (meqns,mvary,  ia,ja,aa,bbb,xxx,
430      &                niter,           u,v,w,  xw,se)
431
432           do i=1,mvary                    ! retain maximum
433               xx(i) = amax1 (xx(i), abs (xxx(i)))
434           end do
435           goto 22
436      24   continue
437           goto 5
438       end if
439 c     ---------------------------------------------------------------
440
441       if (method.eq.3) then               ! singular value decomposition
442           write (iout,108)
443           stop
444 c*        goto 5
445       end if
446 c     ---------------------------------------------------------------
447     5 continue
448       if (.not.square) then
449           if (xx(1) .ne. -1.0) then                ! not singular
450               do i=1,mvary                         ! account for scaling
451                   xx(i) = xx(i)/aats(i)            ! of columns in SCALJJ
452                   xw(i) = xw(i)/aats(i)
453               end do
454           end if
455       end if
456
457
458 c     NOTE:    xw   is NOT used !  <================================
459
460
461       kt = kt+1
462       if (lsout) then
463           write (isout,102)  kt,  (iiii(k), k=1,maraws)
464           write (isout,103)       (  xx(k), k=1,mvary )
465       end if
466       call poplat (xx, 2)                      ! update
467
468       k = mraws          ! inner-most level:  (      ambient,wave,sample)
469       j = maraws         ! inner-most level:  (angle,ambient,wave,sample)
470       j1 = j1save
471       j2 = j2save
472       goto 3             ! loop back ' nested do,   iiii
473     6 continue           ! last line of nested-do,   iiii
474 c     ===============================================================
```

```
      write (iout,111)  kts, kt

      if (lsout)  close (unit=isout)
      call poplat (xx, 3)              ! plot
      call poplat (xx, 4)              ! correlation at minima

      write (iout,112)
      do i=1,mvary
          m = iptu(i)                  ! unique,nonlocal,full
          if (m.le.mlmnts) then
              s1 = diefcn(m)
              s2 = uncerl(m)
              write (iout,113)  i, xx(i), s2, s1, m, '(n+ik)'
          else
              m = m-mlmnts
              s1 = widths(m)
              s2 = uncerz(m)
              write (iout,113)  i, xx(i), s2, s1, m, '(z   )'
          end if
      end do

      return

  100 format (/' seamax,   insufficient allocation for array, AAT')
  101 format (' singular mtx, iiii: ', 20i4, : /(21x, 20i4))
  102 format (1x, i10, ',', 20i4, : /(12x, 20i4))
  103 format (1x, 1p10e12.4)
  104 format ( 1x, 20i4)
  105 format ( 1x,  3i4, 5x, f3.1)

  107 format (/' seamax,  maraws = ', i3, ',   j=', i3
     &        /'          should be equal.')
  108 format (/' seamax,  method=3, singular value decomposition, '
     &        /'          not available, ... yet.')

  111 format (/' seamax,  kts = ', i10, ' = singular events'
     &        /'          kt  = ', i10, ' = total    events')

  112 format (/' Discern:    uncertainty '
     &        /'    when:    not square,    D = [J(T)*J]**-1 *J(T)'
     &        /'                 square,    D =       J **-1      '
     &        /'    case:    1,     |Dg|  +  |DJa|  + |DJ||u|'
     &        /'             2,    ||Dg|| + ||DJa|| + |DJ||u|'
     &       //'    vary',  6x,  'case 1',   5x,  'case 2',
     &                     5x,  'initial',  4x, 'parameter' )

  113 format (3x, i4, ')',  2x,  1p2e11.2,  11x,  e15.6,
     &                          ', for:',  i4,  ',',  a)

       end
```

6.2.21 SEAM2.FOR

```
      subroutine  seam2                 ! construct table <---- asmblx
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      include  'filmss.'
      include  'seamx1.'
      include  'seamx2.'
      real      a(nrows*2), b(2), c(2)

c     Solve the forward scattering problem
c     on a grid of distinct incident angles
c     for each distinct wavelength incident on the sample.
c     Construct matrix of possible scatterings.

      mws = 0                           !               nfilms,wave,sampl
      iws = 0                           !                      wave,sampl
      iaws = 0                          !              ambient,wave,sampl
      iraws = 0                         !       repeat,ambient,wave,sampl
      maraws = 0                        ! mangl,repeat,ambient,wave,sampl
      ngl = 0                           ! ndegr,repeat,ambient,wave,sampl
      do is=1,msampl
          mfilm = nnfilm(is)                    ! FILMSS
          mwave = nnwave(is)
          mfilms = mfilm+1                      ! film/substrate
          nrow = mfilm*3+2                      ! (z,n,k) (n,k)
          do iw=1,mwave
              iws = iws+1
              iwave = iiwave(iws)
              mbien = nnbent(iws)
              qq  = waveqq(iwave)               ! FILMSS
              do i=1,nrow
                  iptx(i) = 0                   ! vary ---> unique
                  ipty(i) = 0                   ! vary ---> full
                  kptx(i) = 0                   ! froz ---> unique
                  kpty(i) = 0                   ! froz ---> full
              end do
              iv = 0                            ! local, full, non-unique
              kv = 0                            ! local, vary, non-unique
              mv = 0                            ! local, vary,   compress
              ku = 0                            ! local, froz, non-unique
              mu = 0                            ! local, froz,   compress

              do m=1,mfilms                     ! films/substrate
                  if (m.ne.mfilms) then         ! films ~ z
                      mws = mws+1
                      iz = iifilm(mws)
                      zzz(m) = widths(iz)       ! FILMSS
                      iv = iv+1                 !    local, full
                      j = mlmnts+iz             ! nonlocal, full
                      if (lvaryz(iz).eq.1) then ! vary
```

```
                    i = iptv(j)             !    nonlocal, compress
                    kv = kv+1               !    local, non-unique
                    iptx(kv) = i            !    local ' nonlocal
                    ipty(kv) = iv           !    local, full
                    if (iptw(i).eq.0) then  !    compress, unique
                        mv = mv+1           !    local counter
                        iptw(i) = mv        !    local
                    end if
                else                        ! frozen
                    i = iptv(j)             !    nonlocal, compress
                    ku = ku+1               !    local, non-unique
                    kptx(ku) = i            !    local ' nonlocal
                    kpty(ku) = iv           !    local, full
                    if (iptw(i).eq.0) then  !    compress, unique
                        mu = mu+1           !    local counter
                        iptw(i) = mu        !    local
                    end if
                end if
            end if                          ! z
            do ink=1,2                      ! n+ik
                mws = mws+1
                nk = iifilm(mws)
                c(ink) = diefcn(nk)         ! n,k
                iv = iv+1                   ! local, full, non-unique
                if (lvaryl(nk).eq.1) then   ! vary
                    i = iptv(nk)            !    nonlocal, compress
                    kv = kv+1               !    local, non-unique
                    iptx(kv) = i
                    ipty(kv) = iv
                    if (iptw(i).eq.0) then  !    compress, unique
                        mv = mv+1           !    local counter
                        iptw(i) = mv        !    local
                    end if
                else                        ! frozen
                    i = iptv(nk)            !    nonlocal, compress
                    ku = ku+1               !    local, non-unique
                    kptx(ku) = i
                    kpty(ku) = iv
                    if (iptw(i).eq.0) then  !    compress, unique
                        mu = mu+1           !    local counter
                        iptw(i) = mu        !    local
                    end if
                end if
            end do      ! n+ik
            die(m) = cmplx (c(1),c(2))**2   ! FILMSS
        end do          ! mfilms

        if (mv.eq.0) then
            write (iout,100)  is, iw, iwave
            stop
        end if

        mvmu = mv+mu    ! compress '    (vary+froz| unique,local)
```

```
104
105                 do mbn=1,mbien                          ! ambients
106                     iaws = iaws+1
107                     imbien = iibent(iaws)
108   c*                mrpeat = nnpeat(iaws)
109                     air = ambent(imbien)                ! FILMSS
110   c*                do irpeat=1,mrpeat
111                         iraws = iraws+1
112                         mangl = mangle(iraws)           ! multiple angles
113                         maraws = maraws+mangl
114                         do iangl=1,ndegr                ! grid of incident angles
115                             ngl = ngl+1                 ! (angl,ambient,wave,sampl)
116
117                             angl = float (iangl)                ! degrees
118                             angl = angl*raddeg                  ! radians
119                             call scatr (qq,angl,  b,a,c)        ! (psi,delta)
120   c*                        call tform (   angl,  b,a,c)        ! (alpha,beta)
121
122                             uang = 0.002*raddeg     ! d(incident angle)
123                             b(1) = 0.050*raddeg     ! d(  psi),   experimental
124                             b(2) = 0.050*raddeg     ! d(delta),    uncertainty
125
126   c                         Retain, save, store the tabulation.
127
128                             psii(ngl) = b(1)        !     psi
129                             dell(ngl) = b(2)        !   delta
130                             psia(ngl) = c(1)        ! d(  psi) /d(incident angle)
131                             dela(ngl) = c(2)        ! d(delta) /d(incident angle)
132
133   c                         Combine/compress common coefficients, unique.
134
135                             do j=1,mvmu             ! local,unique " (vary+froz)
136                                 psid(j,ngl) = 0.0   !   psi
137                                 deld(j,ngl) = 0.0   ! delta
138                             end do
139                             if (kv.ne.0) then       ! vary,  compress
140                                 do k=1,kv           ! local, non-unique
141                                     iv = ipty(k)    ! local, full
142                                     iv2 = iv+iv     ! delta " a " local,full
143                                     iv1 = iv2-1     !   psi " a " local,full
144                                     i   = iptx(k)   ! nonlocal, compress
145                                     imv = iptw(i)   !    local, compress
146                                     psid(imv,ngl) = psid(imv,ngl) + a(iv1)
147                                     deld(imv,ngl) = deld(imv,ngl) + a(iv2)
148                                 end do   ! row
149                             end if       ! vary
150                             if (ku.ne.0) then       ! frozen,  compress
151                                 do k=1,ku           ! local, non-unique
152                                     iv = kpty(k)    ! local, full
153                                     iv2 = iv+iv     ! delta " a " local,full
154                                     iv1 = iv2-1     !   psi " a " local,full
155                                     i   = kptx(k)   ! nonlocal, compress
156                                     imu = iptw(i)   !    local, compress
```

```
157                                     imu = imu+mv   ! append ' (vary+froz)
158                                     psid(imu,ngl) = psid(imu,ngl) + a(iv1)
159                                     deld(imu,ngl) = deld(imu,ngl) + a(iv2)
160                                 end do   ! row
161                             end if       ! frozen
162                         end do       ! angle
163  c*                 end do           ! repeat
164                 end do               ! ambient
165
166                 if (kv.ne.0) then                   ! vary
167                     do k=1,kv                       !    reset
168                         i = iptx(k)                 !    unique-ness
169                         iptw(i) = 0                 !    indicator
170                     end do
171                 end if
172                 if (ku.ne.0) then                   ! frozen
173                     do k=1,ku                       !    reset
174                         i = kptx(k)                 !    unique-ness
175                         iptw(i) = 0                 !    indicator
176                     end do
177                 end if
178             end do        ! wave
179         end do            ! sample
180
181         mraws = iraws
182         return
183
184     100 format (/' oops,  there is NO varying model parameter '
185        &        /'         for the case involving:    sample = ', i3,
186        &        /'         where the ', i3, '-th wave = ', i3)
187         end
```

6.2.22 SEAM3.FOR

```
      subroutine  seam3                 ! fetch table <---- asmblx
      include  'iounit.'
      include  'defnit.'
      include  'filmmm.'
      include  'seamx1.'
      include  'seamx2.'
      logical   firstv, firstu

c     Construct sparse matrix associated with model experiment.

      ii = 1
      ia(1) = 1                         ! vary
      jj = 0

      iaa(1) = 1                        ! frozen
      jja = 0

      iws = 0                           !                       wave,sampl
      mws = 0                           !                nfilms,wave,sampl
      iaws = 0                          !               ambient,wave,sampl
      iraws = 0                         !         repeat,ambient,wave,sampl
      iaraws = 0
      ngl = 0                           !  ndegr,repeat,ambient,wave,sampl
      do is=1,msampl
          mfilm = nnfilm(is)                    ! FILMSS
          mwave = nnwave(is)
          mfilms = mfilm+1                      ! film/substrate
          nrow = mfilm*3+2                      ! (z,n,k) (n,k)
          do iw=1,mwave
              iws = iws+1
              iwave = iiwave(iws)
              mbien = nnbent(iws)
              do i=1,nrow
                  iptx(i) = 0               ! vary ---> unique
                  ipty(i) = 0               ! vary ---> full
                  kptx(i) = 0               ! froz ---> unique
                  kpty(i) = 0               ! froz ---> full
              end do
              iv = 0                        ! local, full, non-unique
              kv = 0                        ! local, vary, non-unique
              mv = 0                        ! local, vary,   compress
              ku = 0                        ! local, froz, non-unique
              mu = 0                        ! local, froz,   compress

              do m=1,mfilms                 ! films/substrate
                  if (m.ne.mfilms) then     ! films ' z
                      mws = mws+1
                      iz = iifilm(mws)
                      iv = iv+1             !    local, full
                      j = mlmnts+iz         ! nonlocal, full
```

```
51                      if (lvaryz(iz).eq.1) then  ! vary
52                          i = iptv(j)            !    nonlocal, compress
53                          kv = kv+1              !    local, non-unique
54                          iptx(kv) = i           !    local ~ nonlocal
55                          ipty(kv) = iv          !    local, full
56                          if (iptw(i).eq.0) then !    compress, unique
57                              mv = mv+1          !    local counter
58                              iptw(i) = mv       !    local
59                              jj = jj+1
60                              ja(jj) = i
61                          end if
62                      else                       ! frozen
63                          i = iptv(j)            !    nonlocal, compress
64                          ku = ku+1              !    local, non-unique
65                          kptx(ku) = i           !    local ~ nonlocal
66                          kpty(ku) = iv          !    local, full
67                          if (iptw(i).eq.0) then !    compress, unique
68                              mu = mu+1          !    local counter
69                              iptw(i) = mu       !    local
70                              jja = jja+1
71                              jaa(jja) = i       !    backwards, mm+1-i=k
72                          end if
73                      end if
74                  end if                         ! z
75                  do ink=1,2                     ! n+ik
76                      mws = mws+1
77                      nk = iifilm(mws)
78                      iv = iv+1                  ! local, full, non-unique
79                      if (lvaryl(nk).eq.1) then  ! vary
80                          i = iptv(nk)           !    nonlocal, compress
81                          kv = kv+1              !    local, non-unique
82                          iptx(kv) = i
83                          ipty(kv) = iv
84                          if (iptw(i).eq.0) then !    compress, unique
85                              mv = mv+1          !    local counter
86                              iptw(i) = mv       !    local
87                              jj = jj+1
88                              ja(jj) = i
89                          end if
90                      else                       ! frozen
91                          i = iptv(nk)           !    nonlocal, compress
92                          ku = ku+1              !    local, non-unique
93                          kptx(ku) = i
94                          kpty(ku) = iv
95                          if (iptw(i).eq.0) then !    compress, unique
96                              mu = mu+1          !    local counter
97                              iptw(i) = mu       !    local
98                              jja = jja+1
99                              jaa(jja) = i       !    backwards, mm+1-i=k
100                         end if
101                     end if
102                 end do     ! n+ik
103             end do         ! mfilms
```

```
104
105                 firstv = .true.
106                 firstu = .true.
107
108                 do mbn=1,mbien                      ! ambients
109                     iaws = iaws+1
110                     imbien = iibent(iaws)
111 c*                  mrpeat = nnpeat(iaws)
112 c*                  do irpeat=1,mrpeat              ! repeats
113                         iraws = iraws+1
114                         mangl = mangle(iraws)
115                         do iangl=1,mangl            ! angles
116                             iaraws = iaraws+1
117                             jjjj = iiii(iaraws)     ! angle
118                             ngll = ngl+jjjj         ! pointer
119
120                             ia(ii+1) = ia(ii)+mv              ! vary
121                             ia(ii+2) = ia(ii)+mv+mv
122                             if (firstv) then
123                                 firstv = .false.
124                                 do j=1,mv
125                                     jj = jj+1
126                                     ja(jj) = ja(jj-mv)
127                                 end do
128                             else
129                                 do j=1,mv
130                                     jj = jj+1
131                                     ja(jj   ) = ja(jj-mv)
132                                     ja(jj+mv) = ja(jj   )
133                                 end do
134                                 jj = jj+mv
135                             end if
136                             j1 = ia(ii  )
137                             j2 = ia(ii+1)-1
138                             do j=j1,j2
139                                 jaj = ja(j)
140                                 imv = iptw(jaj)
141                                 aa(j   ) = psid(imv,ngll)
142                                 aa(j+mv) = deld(imv,ngll)
143                             end do
144
145 c                           Extract info from the tabulation.
146
147                             bb(ii  ) = psii(ngll)
148                             bb(ii+1) = dell(ngll)
149                             cc(ii  ) = psia(ngll)
150                             cc(ii+1) = dela(ngll)
151
152                             iaa(ii+1) = iaa(ii)+mu            ! frozen
153                             iaa(ii+2) = iaa(ii)+mu+mu
154                             if (mu.ne.0) then
155                                 if (firstu) then
156                                     firstu = .false.
```

```
157                                     do j=1,mu
158                                         jja = jja+1
159                                         jaa(jja) = jaa(jja-mu)
160                                     end do
161                                 else
162                                     do j=1,mu
163                                         jja = jja+1
164                                         jaa(jja   ) = jaa(jja-mu)
165                                         jaa(jja+mu) = jaa(jja   )
166                                     end do
167                                     jja = jja+mu
168                                 end if
169                                 jj1 = iaa(ii  )
170                                 jj2 = iaa(ii+1)-1
171                                 do j=jj1,jj2
172                                     jaj = jaa(j)        ! backwards
173                                     imu = iptw(jaj)
174                                     imu = imu+mv        ! append
175                                     aaa(j   ) = psid(imu,ngll)
176                                     aaa(j+mu) = deld(imu,ngll)
177                                 end do
178                             end if        ! frozen
179                             ii = ii+2
180                         end do            ! angles
181                         ngl = ngl+ndegr   ! pointer
182 c*                  end do                ! repeat
183                 end do                    ! ambient
184
185                 if (kv.ne.0) then                   ! vary
186                     do k=1,kv                       !    reset
187                         i = iptx(k)                 !    unique-ness
188                         iptw(i) = 0                 !    indicator
189                     end do
190                 end if
191                 if (ku.ne.0) then                   ! frozen
192                     do k=1,ku                       !    reset
193                         i = kptx(k)                 !    unique-ness
194                         iptw(i) = 0                 !    indicator
195                     end do
196                 end if
197             end do          ! wave
198         end do              ! sample
199         meqns = ii-1
200
201         if (jj .ne. ia(ii)-1) then
202             write (iout,102)  ii,jj,ia(ii)
203             stop
204         end if
205         if (jja .ne. iaa(ii)-1) then
206             write (iout,103)  ii,jja,iaa(ii)-1
207             stop
208         end if
209         if (jj.gt.nnjaaa .or. jja.gt.nnjaaa) then
```

```
210           write (iout,104)
211           stop
212        end if
213
214        return
215
216     102 format (/' asmblx,  inconsistent format of sparse matrix, ',
217        &                                          'aa '' vary'
218        &        /'          ii = ', i10
219        &        /'          jj = ', i10, ' =/= ', i10, ' =/= ia(ii)-1')
220     103 format (/' asmblx,  inconsistent format of sparse matrix, '
221        &                                          'aaa '' froz'
222        &        /'          ii = ', i10
223        &        /'          jja= ', i10, ' =/= ', i10, ' =/= iaa(ii)-1')
224     104 format (/' asmblx,  array allocation for the sparse matrix '
225        &        /'          has been exceeded.'
226        &        /'           aa,ja  <---- nnjaaa (DEFNIT.)'
227        &        /'          aaa,jaa <---- nnjaaa (DEFNIT.)' )
228
229         end
```

6.2.23 POPLAT.FOR

```
      subroutine poplat (u, job)
      dimension   u(1)

      byte        labx(64), laby(64), labg(64)              ! graphics
      real        tlab(41)
      include    'iounit.'
      include    'defnit.'
      include    'filmmm.'
      include    'filmss.'
      include    'seamx1.'
      include    'seamx2.'
      include    'wstack.'

      parameter (ndos = 91)
      integer     kdos(ndos,nrowss)
      real        udos(ndos,nrowss), xdos(ndos)             ! graphics
      real        umin(nrowss), wsav(nrowss), xx2(2), yy2(2)
      logical     first(nrowss)

      parameter (keep=10)
      integer     kkkk(keep*nseams, nrowss),   kk(nrowss)

      common / dossav / udos, xdos, umin, wsav,
     &                  kkkk, kk, kfull, first
      equivalence  (kdos(1,1), udos(1,1))


      goto (1,2,3,4), job

    1 kfull = (keep*nseams)/maraws                ! limit/restrict storage
      kfull = maraws*min (kfull,keep)             ! retain minima, truncate
      do m=1,mvary
          first(m) = .true.
          kk(m) = 0                               ! pointer
          do i=1,ndos                             ! initialize
              kdos(i,m) = 0
          end do
      end do
      return


    2 do m=1,mvary
          w = u(m)                                ! graphics, positioning
          if ((w.gt.1.0E4) .or. (w.lt.0.0)) then ! truncate,   (-5,4)
              w = 4.0
          else if (w .gt. 1.0E-5) then            ! map
              w = alog10 (w)
          else                                    ! truncate
              w = -5.0
          end if
```

```
51
52          wi = (w+5.0)/9.0                        ! (0,1)
53          i = nint (wi*float (ndos))              ! index/position DOS
54          i = max (1, min (i,ndos))               ! assurance
55          kdos(i,m) = kdos(i,m)+1                 ! update
56
57          if (first(m)) then                      ! discern minimum
58              first(m) = .false.                  ! initialize
59              umin(m) = u(m)
60              wsav(m) = w
61              kk(m) = maraws
62              do j=1,maraws
63                  kkkk(j,m) = iiii(j)             ! retain angles
64              end do
65          else if (w.lt.wsav(m)) then             ! update
66              umin(m) = u(m)
67              wsav(m) = w
68              kk(m) = maraws
69              do j=1,maraws
70                  kkkk(j,m) = iiii(j)             ! retain angles
71              end do
72          else if (w.eq.wsav(m)) then             ! multiple minima
73              if (kk(m).lt.kfull) then
74                  k = kk(m)
75                  do j=1,maraws
76                      k = k+1
77                      kkkk(k,m) = iiii(j)
78                  end do
79                  kk(m) = k
80              else                                ! no retention, but
81                  kk(m) = kk(m)+maraws            !    continue counting
82              end if
83          end if
84        end do
85        return
86
87
88      3 do i=1,ndos
89          xdos(i) = - 5.0 * (float(ndos-i)/float(ndos-1))
90       &              + 4.0 * (float(   i-1)/float(ndos-1))
91        end do
92        xx2(1) = xdos(1   )                       ! min
93        xx2(2) = xdos(ndos)                       ! max
94        yy2(1) = 0.0                              ! min
95        yy2(2) = 1.0                              ! max
96
97        do m=1,mvary
98            write (iout,15)  m, umin(m), wsav(m)
99            write (iplt,19) ndos, m
100
101           kmax = 0.0                            ! Population, DOS
102           do i=1,ndos
103               kmax = max (kmax, kdos(i,m))
```

```
104             end do
105             umax = 0.94/alog (float (kmax))         !   scale ~  0.99 - .05
106             do i=1,ndos                             ! rescale
107                 if (kdos(i,m) .eq. 0) then
108                     udos(i,m) = 0.0
109                 else
110                     udos(i,m) = umax*alog (float (kdos(i,m))) + 0.05
111                 end if
112                 write (iplt,20) i, xdos(i), udos(i,m)
113             end do
114
115             encode (12, 21, labx)                   ! capital letters only
116             encode (32, 22, laby)                   !      terminated by a
117             encode (26, 23, labg) m                 !      dollar ($) sign.
118
119             linlog = 1                              ! (linear, linear) ~ (x,u)
120             call displa (2, 0, linlog)
121             call agsetf ('GRAPH/RIGHT.', 0.8)
122
123             call aggeti ('LINE/MAXI.', mlln)
124             call aggetf ('LINE/END.',  tcln)
125             call agcpyl (min0(80,mlln), tcln,    labg,   tlab,ncdum)
126             call agsetf ('LABE/NAME.', 4H   T)
127             call agseti ('LINE/NUMB.', 100)
128             call agsetp ('LINE/TEXT.', tlab, 1)
129
130             call anotat (labx,laby,1,1,   0,0)
131             call agstup (xx2,1,0,2,1,  yy2,1,0,2,1)
132             call agback
133             call agcurv (xdos,1,udos(1,m),1,ndos,1)     ! line solid
134             call flush
135             call frame    !--------------------------------------------
136         end do
137
138         do m=1,mvary
139             k = kk(m)/maraws                        ! population of minima
140             write (iout,31) m,k
141             k = min (k,keep)                        ! truncation
142             k2 = 0
143             do j=1,k                                ! distinct sets of angles
144                 k1 = k2+1
145                 k2 = k2+maraws
146                 write (iout,32) j, (kkkk(i,m), i=k1,k2)
147             end do
148         end do
149         return
150
151
152 c       Discern correlation among model parameters at minima.
153
154       4 do m=1,mvary
155             kt = kk(m)/maraws            ! distinct minima
156             write (iout,30)
```

```
157             write (iout,31) m, kt
158
159             kt = min (kt,keep)              ! truncate
160             ik = 0
161
162             do iki=1,kt
163                 do j=1,maraws               ! extract angles at minima
164                     ik = ik+1
165                     iiii(j) = kkkk(ik,m)
166                 end do
167                 write (iout,203) (iiii(j), j=1,maraws)
168
169                 call seam3                  ! model experiment
170
171                 kv = 0
172                 do jv=1,mvary               ! A(T)*A
173                     do iv=1,jv              ! diagonal, upper triangle
174                         ss = 0.0
175                         do i=1,meqns,2
176                             j1 = ia(i  )
177                             j2 = ia(i+1)-1
178                             mv = j2-j1+1
179                             s1 = 0.0
180                             s2 = 0.0
181                             s3 = 0.0
182                             s4 = 0.0
183                             do j=j1,j2              ! row
184                                 jaj = ja(j)         ! column
185                                 if (jaj.eq.iv) then
186                                     s1 = aa(j   )
187                                     s3 = aa(j+mv)
188                                 end if
189                                 if (jaj.eq.jv) then
190                                     s2 = aa(j   )
191                                     s4 = aa(j+mv)
192                                 end if
193                             end do
194                             ss = ss + s1*s2 + s3*s4
195                         end do      ! meqns
196                         ss = ss /float (meqns)
197                         kv = kv+1
198                         aat(kv) = ss
199                         if (iv.eq.jv) then
200                             u(jv) = sqrt (ss)
201                         end if
202                     end do
203                 end do
204                 kv = 0
205                 do jv=1,mvary                       ! renormalize
206                     do iv=1,jv
207                         kv = kv+1
208                         aat(kv) = aat(kv)/(u(iv)*u(jv))
209                     end do
```

```
210               end do
211               write (iout,204)
212
213               k2 = 0
214               do i=1,mvary
215                   k1 = k2+1
216                   k2 = k2+i
217                   write (iout,205)  i, (aat(k), k=k1,k2)
218               end do
219               write (iout,207)
220               write (iout,208)  (u(i), i=1,mvary)
221               if (iki.ne.kt)  write (iout,29)
222           end do   ! minima
223       end do       ! vary
224       write (iout,30)
225       return
226
227    15 format (/' vary, i =', i4, ',      Umin = ', 1pe11.4,
228      &                           ',        ws = ',   e11.4)
229    19 format (1x, i5, '  1',  i5,' ' ndos, nu, ivary')
230    20 format (1x, i5, 5x, 1p2e13.5)
231
232    21 format ('UNCERTAINTY$')                          ! 12
233    22 format ('POPULATION,   (LOG, NORMALIZED)$')      ! 32
234    23 format ('VARIATION PARAMETER, I=', I2, '$')      ! 26
235
236    29 format (1x, 15('----'))
237    30 format (1x, 15('===='))
238    31 format (/' vary =', i4, ',    population at minima =', i10)
239    32 format ( 7x, i4, ')', 20i4, : /(12x, 20i4))
240
241   203 format ( 10x, 'iiii:', 20i4, : /(15x, 20i4))
242   204 format (/' A(T)*A:        (correlation)')
243   205 format (1x, i4, ')', 1x, 10f10.5, : /(7x, 10f10.5))
244   207 format (/' normalization coefficients:')
245   208 format (1x, 1p10e10.2)
246       end
```

6.2.24 PLTE.FOR

```
c     Plot the data results from:    SEAMAX
c     Sensitivity/Error Analysis for Multiple:
c                angle, ambient, wave, sample.

      program   plte
      byte      labx(64), laby(64), labg(64)
      real      tlab(41),    s(9)
      character*64  filenm

      include  'defnit.'
      integer   nnwave(nsampl)
      integer   nnbent(nbient*nwaves*nsampl)
      integer   mangle(nbient*nwaves*nsampl)
      integer   isteps(nbient*nwaves*nsampl)
      integer   iii(nseams), ii1(nseams), ii2(nseams), kkk(nseams)
      real      u(nrowss), umin, wsav
      logical   first

      parameter (keep=10)
      integer    mmm(keep*nseams), mm

      parameter (nx = 89)                                  ! degrees
      real       ww(nx*nx), xx(nx), yy(nx)                 ! graphics
      integer    info(2*nx*nx)

      parameter  (ndos = 91)
      integer     kdos(ndos)
      real        udos(ndos), xdos(ndos)
      equivalence   (kdos(1), udos(1))
      data   in,iout,idat / 5,6,7 /


      read  (  in,301, err=102,end=102)  filenm
      close (unit=in)
      write (iout,303) filenm
      open (unit=idat, file=filenm, status='old',
     &      readonly, shared, err=103)

      loop = 0
   11 loop = loop+1                                        ! ivary
      rewind (unit=idat)

      read (idat,*) msampl, mvary, mdegr
      read (idat,*) (nnwave(is), is=1,msampl)
      iws = 0
      iaws = 0
      iraws = 0
      maraws = 0
      do is=1,msampl
          mwave = nnwave(is)
```

```
51            do iw=1,mwave
52                iws = iws+1
53                read (idat,*) mbien
54                nnbent(iws) = mbien
55                do mbn=1,mbien
56                    iaws = iaws+1
57  c*                read (idat,*) mrpeat                    ! " 1
58  c*                nnpeat(iaws) = mrpeat
59  c*                do irpeat=1,mrpeat
60                        iraws = iraws+1
61                        read (idat,*) istep,mangl
62                        isteps(iraws) = istep
63                        mangle(iraws) = mangl
64                        maraws = maraws+mangl
65  c*                end do     ! repeat
66                end do        ! ambient
67            end do           ! wave
68        end do              ! sample
69        mraws = iraws
70        mx    = mdegr
71        kfull = (keep*nseams)/maraws          ! prevent exceeding storage
72        kfull = maraws*min (kfull,keep)       ! retain minima, truncation
73
74        first = .true.
75        do i=1,ndos
76            kdos(i) = 0                       ! initialize
77        end do
78        kt = 0
79        mm = 0
80        m = loop                              ! ivary
81  c     ================================================================
82        j2 = 0
83        k = 0
84      1 k = k+1
85        if (k.gt.mraws) goto 4
86        istep = isteps(k)
87        mangl = mangle(k)
88        j1 = j2+1
89        j2 = j2+mangl
90        ii2(j1) = mdegr                       ! limit outer-most do
91        do j=j1,j2                            ! multiple angle
92            ii1(j) = 1+ (j2-j)*istep
93            iii(j) = ii1(j) - istep
94        end do
95
96        j = j1-1
97      2 j = j+1
98        if (j.gt.j2) goto 1
99      3 iii(j) = iii(j)+istep
100       if (iii(j) .le. ii2(j)) then          ! test upper limit
101           if (j.ne.j2)  ii2(j+1)=iii(j)-istep
102           goto 2
103       end if
```

```
104         iii(j) = ii1(j)-istep                   ! reset inner do
105         j = j-1                                 ! backup one do-level
106         if (j.ge.j1) goto 3
107
108         if (k.eq.1) goto 6                      ! escape do-nest
109         k = k-1
110         istep = isteps(k)
111         mangl = mangle(k)
112         j2 = j1-1
113         j1 = j1-mangl
114         goto 3
115       4 continue
116
117         j1save = j1
118         j2save = j2
119         if (j.ne.maraws+1) then
120             write (iout,107) maraws,j
121             stop
122         end if
123   c     ---------------------------------------------------------------
124         kt = kt+1
125         read (idat,*)  kkkk, (kkk(k),k=1,maraws)
126         read (idat,*)        (  u(k),k=1,mvary )
127
128         if (kkkk .ne. kt) then                  ! test consistency
129             write (iout,212)  kkkk,kt
130             goto 101
131         end if
132         do i=1,maraws                           ! consistency check
133             if (kkk(i) .ne. iii(i)) then
134                 write (iout,212)  kkkk, kt
135                 write (iout,213) (kkk(k), k=1,maraws)
136                 write (iout,214) (iii(k), k=1,maraws)
137                 goto 101
138             end if
139         end do
140
141         w = u(m)                                ! graphics, positioning
142         if ((w.gt.1.0E4) .or. (w.lt.0.0)) then          ! (-5,4)
143             w = 4.0
144         else if (w .gt. 1.0E-5) then
145             w = alog10 (w)
146         else                                    ! truncate
147             w = -5.0
148         end if
149
150         wi = (w+5.0)/9.0                        ! (0,1)
151         i = nint (wi*float (ndos))              ! Density of States
152         i = max (1, min (i,ndos))
153         kdos(i) = kdos(i)+1                     ! update
154
155         if (first) then                         ! discern minimum
156             first = .false.                     ! initialize
```

```
157             um = u(m)                           ! retain minimum
158             ws = w
159             do i=1,maraws
160                 mmm(i) = iii(i)                 ! retain angles
161             end do
162             mm = maraws                         ! pointer/counter
163         else if (w.lt.ws) then                  ! minimum update
164             um = u(m)
165             ws = w
166             do i=1,maraws
167                 mmm(i) = iii(i)                 ! retain angles
168             end do
169             mm = maraws
170         else if (w.eq.ws) then                  ! multiple minima
171             if (mm.lt.kfull) then               ! prevent exceeding storage
172                 do i=1,maraws
173                     mm = mm+1                   ! pointer/counter
174                     mmm(mm) = iii(i)            ! retain angles
175                 end do
176             else
177                 mm = mm+maraws
178             end if
179         end if
180
181 c       Note that all plots involve two dimensions.   So, for cases
182 c       involving multiple:    angles, ambients, waves, and samples,
183 c       ... it is NOT clear what should (or is able to) be plotted.
184 c       For the special case involving:    1 or 2  multiple angles,
185 c                                  1 ambient,  1 wave, and  1 sample,
186 c       we may consider plotting the following arrays:
187
188         if (maraws .eq. 1) then
189             i = iii(1)                          ! >0
190             ww(i   ) = float (i)                ! x ' angle
191             ww(i+mx) = w                        ! u ' uncertainty
192         else if (maraws .eq. 2) then
193             i = iii(1) + mx*(iii(2)-1)          !  (i1,i2)   symmetric
194             j = iii(2) + mx*(iii(1)-1)          !  (i2,i1)
195             ww(i) = w
196             ww(j) = w
197         end if
198 c       ---------------------------------------------------------------
199         k = mraws
200         j = maraws
201         j1 = j1save
202         j2 = j2save
203         goto 3
204       6 continue                                ! escape do-loops
205 c       ===============================================================
206
207         k = mm/maraws
208         write (iout,215)  m, k, um, ws          ! minimum
209         k = min (k,keep)                        ! truncate
```

```
      k2 = 0
      do j=1,k                                  ! angluar locations
          k1 = k2+1                             !     along valleys
          k2 = k2+maraws
          write (iout,216) j, (mmm(i), i=k1,k2)
      end do

      do i=1,ndos
          xdos(i) = - 5.0 * (float(ndos-i)/float(ndos-1))
     &              + 4.0 * (float(   i-1)/float(ndos-1))
      end do
      xx(1) = xdos(1    )                               ! min
      xx(2) = xdos(ndos)                                ! max
      yy(1) = 0.0                                       ! min
      yy(2) = 1.0                                       ! max

      kmax = 0.0                                        ! Population, DOS
      do i=1,ndos
          kmax = max (kmax, kdos(i))
      end do
      umax = 0.94 / alog (float (kmax))                 !   scale ~   0.99 - 0.05
      do i=1,ndos                                       ! rescale
          if (kdos(i) .eq. 0) then
              udos(i) = 0.0
          else
              udos(i) = umax*alog (float (kdos(i)))  + 0.05
          end if
      end do

      encode (12, 221, labx)                ! capital letters only
      encode (32, 222, laby)                !      terminated by a
      encode (26, 223, labg) m              !      dollar ($) sign.

      linlog = 1                            ! (linear, linear) ~ (x,u)
      call displa (2, 0, linlog)
      call agsetf ('GRAPH/RIGHT.', 0.8)

      call aggeti ('LINE/MAXI.', mlln)
      call aggetf ('LINE/END.',  tcln)
      call agcpyl (min0(80,mlln), tcln,    labg,   tlab,ncdum)
      call agsetf ('LABE/NAME.', 4H   T)
      call agseti ('LINE/NUMB.', 100)
      call agsetp ('LINE/TEXT.', tlab, 1)

      call anotat (labx,laby,1,1,   0,0)
      call agstup (xx,1,0,2,1,  yy,1,0,2,1)
      call agback
      call agcurv (xdos,1,udos,1,ndos,1)        ! line solid
      call flush
      call frame    !-----------------------------------------------

c     Plot:    uncertainty
```

```
263
264       if (maraws.eq.1) then                          ! single angle
265           mxs = mx                                   ! assume:  istep=1
266           if (istep.ne.1) then                       ! compress matrix
267               j = 0                                  ! compress
268               do i=1,mx,istep
269                   j = j+1
270                   ww(j   ) = ww(i   )                ! x
271                   ww(j+mx) = ww(i+mx)                ! u
272               end do
273               mxs = j                                ! consecutive points
274           end if
275
276           xx(1) = 0.0                                ! xmin
277           xx(2) = ww(mxs)                            ! xmax
278           yy(1) = -5.0                               ! umin
279           yy(2) =  4.0                               ! umax
280
281           if (xx(2) .lt. 45.0) then                  ! convenience
282               xx(2) = 45.0
283           else
284               xx(2) = 90.0
285           end if
286
287           encode (32, 224, labx)                     ! capital letters only
288           encode (12, 221, laby)                     !      terminated by a
289           encode (26, 223, labg) m                   !      dollar ($) sign.
290
291           linlog = 1                                 ! (linear, linear) ' (x,u)
292           call displa (2, 0, linlog)
293           call agsetf ('GRAPH/RIGHT.', 0.8)
294
295           call aggeti ('LINE/MAXI.', mlln)
296           call aggetf ('LINE/END.',  tcln)
297           call agcpyl (min0(80,mlln), tcln,    labg,   tlab,ncdum)
298           call agsetf ('LABE/NAME.', 4H   T)
299           call agseti ('LINE/NUMB.', 100)
300           call agsetp ('LINE/TEXT.', tlab, 1)
301
302           call anotat (labx,laby,1,1,   0,0)
303           call agstup (xx,1,0,2,1,  yy,1,0,2,1)
304           call agback
305           call agcurv (ww(1),1,  ww(1+mx),1,  mxs,1)  ! line solid
306           call flush
307           call frame    !--------------------------------------
308       end if
309
310
311       if (maraws.eq.2) then            ! double angle,    surface plot
312           span = 9.0                   ! (-5,4)
313           do i=1,mx,istep
314               j = i + (i-1)*mx         ! diagonal ' (i,i)
315               ww(j) = 4.0              ! maximum
```

```
316             xy = (float(i-1)/float(mx-1)) * 2.0*span     ! scale plots
317             xx(i) = xy                ! domain ' plot routine
318             yy(i) = xy                ! domain ' plot routine
319         end do
320
321         mxs = mx                      ! assume:   istep=1
322         if (istep.ne.1) then          ! compress matrix
323             mxs = 0                   ! compress index
324             k = 0
325             do j=1,mx,istep
326                 jmx = (j-1)*mx
327                 do i=1,mx,istep
328                     k = k+1
329                     ww(k) = ww(i+jmx)
330                 end do
331                 mxs = mxs+1
332                 xx(mxs) = xx(j)
333                 yy(mxs) = yy(j)
334             end do
335         end if
336
337         xn = 1.0                                ! viewing vector
338         yn = 1.0
339         zn = 1.0
340
341 c       s ' defines the line of sight of viewer and object.
342 c       the viewer's eye is at :   s(1-3) ' (x,y,z) ' NCAR
343 c       the point looked at is :   s(4-6)           ' NCAR
344 c       effective radius of obj:   s(7-9)           ' convenience
345
346         s(4) = span        ! (xmax+xmin)*0.5    center of object or
347         s(5) = span        ! (ymax+ymin)*0.5       point being viewed
348         s(6) = 0.0         ! (umax+umin)*0.5
349
350         s(7) = span        ! (xmax-xmin)*0.5    radius of object
351         s(8) = span        ! (ymax-ymin)*0.5
352         s(9) = 0.0         ! (umax-umin)*0.5
353
354         radius = amax1 (s(7), s(8), s(9)) ! effective radius
355         dist = radius*5.0                 ! convenient distance from
356         ss   = xn*xn+yn*yn+zn*zn          !   which to view the object
357         dist = dist/sqrt(ss)              ! induce unit vector
358         s(1) = s(4) + xn*dist
359         s(2) = s(5) + yn*dist
360         s(3) = s(6) + zn*dist
361
362         call seti (10,10)
363         call srface (xx,yy,ww,info,  mxs,mxs,mxs,  s,0.0)
364         call ezcntr (ww, mxs,mxs)
365       end if
366       if (loop.ne.mvary) goto 11
367
368   101 close (unit=idat)
```

```
      stop
  102 write (iout,302) in
      stop
  103 write (iout,302) idat
      stop

  107 format (' oops,    maraws,j: ', 2i5)
  212 format (' oops:    ... input data is out of sync.'
     &       /'          card index, kkkk = ', i10
     &       /'          card count,   kt = ', i10)
  213 format (/' card data, kkk:',  10i4, : /(16x, 10i4))
  214 format (/' do-loop,   iii:',  10i4, : /(16x, 10i4))
  215 format (/' vary, i =',i4,',     population of minima =',i10
     &        / 20x, 'Umin =', 1pe11.4,  5x, 'w=', e11.4)
  216 format (5x, i4, ')', 20i4, : /(10x, 20i4))

  221 format ('UNCERTAINTY$')                          ! 12
  222 format ('POPULATION,   (LOG, NORMALIZED)$')      ! 32
  223 format ('VARIATION PARAMETER, I=', I2, '$')      ! 26
  224 format ('ANGLE OF INCIDENCE,   (DEGREES)$')      ! 32

  301 format ( a64)
  302 format (' Unable to open IO unit = ', i3)
  303 format ( 1x, a64)
      end
```

6.3 General Utilities

6.3.1 DOT.FOR

```
      subroutine dot (n,x,y,  xy,k)
      real       x(1), y(1)

      xy = 0.0
      xn = 0.0
      yn = 0.0
      do i=1,n                ! find maximum
          xn = amax1 (xn, abs (x(i)))
          yn = amax1 (yn, abs (y(i)))
      end do
      if (xn.eq.0.0 .or. yn.eq.0.0)  return

      xs = 0.0
      ys = 0.0
      do i=1,n
          xx = x(i)/xn        ! scaled vector component
          yy = y(i)/yn
          xs = xs + xx*xx     ! dot product ' |x*x|
          ys = ys + yy*yy     ! dot product ' |y*y|
          xy = xy + xx*yy     ! dot product ' |x*y|
      end do

      h  = float (n)
      xs = xs/h               ! mean square value
      ys = ys/h
      xy = xy/h
      xs = sqrt (xs)          ! root mean square value
      ys = sqrt (ys)

      if (k.eq.0) then        ! usual dot product
          xy = xy*xn*yn*h
      else if (k.eq.1) then   ! un-normalized
          xy = xy*xn*yn
      else                    !    normalized
          xy = xy/(xs*ys)
      end if

      return
      end
```

6.3.2 NORM.FOR

```
1       subroutine norm (n,x, xn,k)
2       real  x(1)
3
4       xn = 0.0
5       do i=1,n                          !  find maximum
6          xn = amax1 (xn, abs (x(i)))
7       end do
8       if (xn .eq. 0.0) return
9
10      xx = 0.0
11      do i=1,n                          !  dot product
12          xx = xx + (x(i)/xn)**2        !  |x*x|
13      end do
14
15      if (k.eq.1) then                  ! mean squared value
16          xx = xx/float (n)
17      end if
18      xn = xn*sqrt (xx)                 !  Euclidean norm
19
20      return
21      end
```

6.3.3 APROD.FOR

```
      subroutine aprod (mode,m,n,x,y,  ia,ja,aa)
      integer    mode, m, n, ia(1), ja(1)
      real       x(n), y(m), aa(1)
      data       iout / 6 /

c     ------------------------------------------------------------------
c     A ~ A(m,n),                Transpose operator ~ (')
c     Operation:     mode= 1,    set:   y  = y  + A *x
c                    mode= 2,    set:   x  = x  + A'*y
c                                       x' = x' + y'*A
c     ------------------------------------------------------------------

      if (mode.eq.1) then              ! y = y + Ax
          do i=1,m                     ! scan rows of matrix A
              mj = ia(i+1)-ia(i)       ! number of columns in row
              if (mj.ne.0) then
                  ss = 0.0             ! sum
                  jj = ia(i)-1         ! indexing
                  do j=1,mj            ! scan columns of row
                      jj = jj+1
                      kk = ja(jj)      ! column
                      ss = ss + aa(jj)*x(kk)
                  end do
                  y(i) = y(i)+ss       ! y = y+Ax
              end if
          end do

      else if (mode.eq.2) then         ! x' = x' + y'*A
          do i=1,m                     ! scan rows of matrix A
              mj = ia(i+1)-ia(i)       ! number of columns in row
              if (mj.ne.0) then
                  yy = y(i)
                  jj = ia(i)-1         ! indexing
                  do j=1,mj            ! scan columns of row
                      jj = jj+1
                      kk = ja(jj)      ! column
                      x(kk) = x(kk) + yy*aa(jj)
                  end do
              end if
          end do

      else
          write (iout,10)  mode
          stop
      end if

      return

   10 format (/' aprod,  ... error,  mode= ', i2)
      end
```

6.3.4 SCALII.FOR

```
      subroutine scalii (m,n,  ia,ja,a,b,  p,k)
      integer    ia(1), ja(1), m, n, k
      real       a(1), b(1), p(1)

c     ---------------------------------------------------------------
c     Scale the rows in matrix,       A(m,n), b(m).
c     Retain scaling coefficients in  P(m).
c     Evaluate  P  only when    k  = 0,1,2,3.
c     Re-scale  A  only when   |k| =     2,3.
c     Re-scale  B  only when   |k| =   1,  3.
c     Matrix A is stored (row-wise) in the Yale Sparse Matrix Format.
c     ---------------------------------------------------------------

      ka = iabs (k)                                ! convenience
      if (ka.gt.3) then                            ! out of range
          do i=1,m                                 ! scan rows
              p(i) = 1.0                           ! default
          end do
          return
      end if

      if (k.ge.0) then                             ! determine scaling
          do i=1,m                                 ! scan rows
              mj = ia(i+1)-ia(i)                   ! number of columns in row
              if (mj.ne.0) then
                  big = 0.0                        ! initialize
                  jj = ia(i)-1                     ! indexing
                  do j=1,mj                        ! scan columns in row
                      jj = jj+1
                      big = amax1 (big, abs (a(jj)))
                  end do
                  if (big .eq. 0.0) then           ! trivial case
                      p(i) = 1.0                   ! default
                  else
                      ss = 0.0                     ! initialize
                      jj = jj-mj                   ! reset indexing
                      do j=1,mj                    ! scan columns in row
                          jj = jj+1
                          ss = ss + (a(jj)/big)**2
                      end do
                      p(i) = big*sqrt (ss)         ! scale factor
                  end if
              end if
          end do
      end if

      if (ka.eq.2 .or. ka.eq.3) then               ! rescale A
          do i=1,m                                 ! scan rows, A
              mj = ia(i+1)-ia(i)                   ! number of columns in row
              if (mj.ne.0) then
```

```
51                  pp = p(i)                       ! scale factor of row
52                  jj = ia(i)-1                    ! indexing
53                  do j=1,mj                       ! scan columns in row
54                      jj = jj+1
55                      a(jj) = a(jj)/pp            ! rescale the row
56                  end do
57              end if
58          end do
59      end if
60
61      if (ka.eq.1 .or. ka.eq.3) then              ! rescale b
62          do i=1,m                                ! scan rows
63              b(i) = b(i)/p(i)
64          end do
65      end if
66
67      return
68      end
```

6.3.5 SCALJJ.FOR

```
1       subroutine  scaljj (m,n,  ia,ja,a,b,  p,w,k)
2       integer     ia(1), ja(1), k, m, n
3       real        a(1), b(1), p(1), w(1)
4
5   c   ------------------------------------------------------------------
6   c   Scale matrix A so the diagonal of:      [Transpose(A)*A] = 1.
7   c   Matrix A is stored row-wise in the Yale Sparse Matrix Format.
8   c   Scale the columns of matrices,  A(m,n), b(n).
9   c   Retain the scaling coefficients in      P(n).
10  c   Use as a dummy work storage array,      W(n).
11  c   Evaluate  P  only when     k  = 0,1,2,3.
12  c   Re-scale  A  only when    |k| =     2,3.
13  c   Re-scale  B  only when    |k| =   1,  3.
14  c   ------------------------------------------------------------------
15
16      ka = iabs (k)                                ! convenience
17      if (ka.gt.3) then                            ! out of range
18          do j=1,n                                 ! scan columns
19              p(j) = 1.0                           !      default
20          end do
21          return
22      end if
23
24      if (k.ge.0) then                             ! determine scaling
25          do j=1,n                                 ! scan columns
26              p(j) = 0.0                           ! initialize
27              w(j) = 0.0                           ! sums
28          end do
29
30          do i=1,m                                 ! scan rows, A
31              mj = ia(i+1)-ia(i)                   ! number of columns in row
32              if (mj.ne.0) then
33                  jj = ia(i)-1                     ! indexing
34                  do j=1,mj                        ! scan columns of row
35                      jj = jj+1
36                      kk = ja(jj)                  ! column
37                      p(kk) = amax1 (p(kk), abs (a(jj)))   ! max |A(,j)|
38                  end do
39              end if
40          end do
41
42          do i=1,m                                 ! scan rows, A
43              mj = ia(i+1)-ia(i)                   ! number of columns
44              if (mj.ne.0) then
45                  jj = ia(i)-1                     ! indexing
46                  do j=1,mj                        ! scan columns in row
47                      jj = jj+1
48                      kk = ja(jj)                              ! column
49                      w(kk) = w(kk) + (a(jj)/p(kk))**2         ! sums
50                  end do
```

```
51              end if
52          end do
53
54          do j=1,n                                ! scan columns
55              p(j) = p(j)*sqrt (w(j))             ! retain scale factor
56          end do
57      end if
58
59      if (ka.eq.2 .or. ka.eq.3) then              ! rescale A
60          do i=1,m                                ! scan rows, A
61              mj = ia(i+1)-ia(i)                  ! number of columns
62              if (mj.ne.0) then
63                  jj = ia(i)-1                    ! indexing
64                  do j=1,mj                       ! scan columns in row
65                      jj = jj+1
66                      kk = ja(jj)                 ! column
67                      a(jj) = a(jj)/p(kk)         ! rescale column
68                  end do
69              end if
70          end do
71      end if
72
73      if (ka.eq.1 .or. ka.eq.3) then              ! rescale b
74          do j=1,n                                ! scan columns
75              b(j) = b(j)/p(j)                    ! rescale column
76          end do
77      end if
78
79      return
80      end
```

6.3.6 CGNL.FOR

```
1         subroutine cgnl (ma,na,ia,ja,aa,  b,x,
2        &                 itmax,   u,v,w,  xx,se)
3
4         integer    ia(1), ja(1)
5         real       aa(1), b(1), x(1)
6         real       u(1), v(1), w(1), xx(1), se(1)
7         external   aprod
8         data       iout /6/
9
10   c    ------------------------------------------------------------
11   c    real       b(ma), u(ma),              aa(ma,na)
12   c    real       x(na), v(na), w(na), xx(na), se(na)
13
14   c    Solve the linear or matrix algebra problem, Ax=b.
15   c    Matrix A is stored row-wise in the Yale Sparse Matrix Format.
16   c    Reference:    C.C.Paige and M.A.Saunders,
17   c       ''LSQR:    An Algorithm for Sparse Linear Equations
18   c                  and Sparse Least Squares'',
19   c             Association for Computing Machinery,
20   c             Transactions on Mathematical Software,
21   c             Volume 8, Number 1, March 1982, pp.  43-71, (Note pp. 50-51).
22   c    ibid.,   Volume 8, Number 2,  June 1982, pp. 195-209.
23   c    ------------------------------------------------------------
24
25        loop = 0                                    ! initialize
26       1 loop = loop+1                              ! update counter
27        rr = 0.0                                    ! norm of residual
28        do i=1,ma                                   ! scan rows,  r=b-Ax
29            mj = ia(i+1)-ia(i)                      ! number of columns in row
30            if (mj .ne. 0) then
31                ss = 0.0                            ! initialize sum
32                jj = ia(i)-1                        ! indexing
33                do j=1,mj                           ! scan columns of row
34                    jj = jj+1
35                    kk = ja(jj)                     ! column
36                    ss = ss + aa(jj)*x(kk)          ! Ax
37               end do
38               ss = b(i)-ss                         !  r = b-Ax = residual
39               u(i) = ss                            !  r        = residual
40               rr = rr+ss*ss                        ! |r|**2
41            end if
42        end do
43        if (rr .eq. 0.0) return
44
45        if (loop .eq. 1) then                       ! retain first norm
46            rrr = rr
47        else if (rr .ge. rrlast*0.98) then          ! rate of convergence
48   c*       ratio = sqrt (rr/rrr)
49   c*       write (iout,103)  loop,  ratio
50   c*       write (iout,104)  istop, anorm, acond, rnorm, arnorm, xnorm
```

```
51          return
52       end if
53       rrlast = rr                                ! update
54
55 c------------------------------------------------ ! Set-up for LSQR
56       relpr =  1.0E-06  ! relative precision of floating point arithmetic
57       damp =   1.0E+00  !
58       atol =   1.0E-06  ! relative error of data in A
59       btol =   1.0E-06  ! relative error of data in B ' rhs
60       conlim = 1.0E+04  ! apparent condition number of matrix A-bar
61                         ! (upper limit)
62       itnlim = itmax    ! upper limit on number of iterations
63       nout   = -iout    ! output index to printer
64
65       call lsqr (ma, na, aprod, damp,
66      &           ia, ja, aa,
67      &           u, v, w, xx, se,
68      &           atol, btol, conlim, itnlim, nout,
69      &           istop, anorm, acond, rnorm, arnorm, xnorm)
70
71       do i=1,na
72           x(i) = x(i) + xx(i)          ! update solution
73           xx(i) = 0.0                  ! reset
74       end do
75       goto 1                           ! loop back
76
77    101 format (' cgnl, singular    row= ', i10)
78    102 format (' cgnl, singular column= ', i10)
79    103 format (' cgnl,  ', i5, 1p1e11.3, ' ' loop, ratio '
80      &                                 , '(residual reduction)')
81    104 format (' cgnl,  ', i5, 1p5e11.3)
82        end
```

6.3.7 LSQR.FOR

The following subroutine is located in the software library,

> *Guide to Available Mathematical Software*, by R.F. Boisvert, S.E. Howe, and D.K. Kahaner, Center for Applied Mathematics, National Institute of Standards and Technology (formerly, National Bureau of Standards), U.S. Department of Commerce, 1985.

The source code is copyrighted by the Association for Computing Machinery, Inc. The references for the following source code include the following:

1) C.C. Paige and M.A. Saunders, "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *ACM Trans. Math. Softw.* **8**, 43-71 (1982).

2) C.C. Paige and M.A. Saunders, "Algorithm 583, LSQR: Sparse Linear Equations and Least Squares Problems," *ACM Trans. Math. Softw.* **8**, 195-209 (1982).

```
      SUBROUTINE LSQR (M,N,APROD,DAMP,
     1                 IA,JA,AA,
     2                 U,V,W,X,SE,
     3                 ATOL,BTOL,CONLIM,ITNLIM,NOUT,
     4                 ISTOP,ANORM,ACOND,RNORM,ARNORM,XNORM )
C
      EXTERNAL   APROD
      INTEGER    M,N,ITNLIM,NOUT,ISTOP
      INTEGER    IA(M),JA(M)
      REAL       AA(M),                    U(M),V(N),W(N),X(N),SE(N),
     1           ATOL,BTOL,CONLIM,DAMP,ANORM,ACOND,RNORM,ARNORM,XNORM
C     ------------------------------------------------------------------
C
C     LSQR  FINDS A SOLUTION  X  TO THE FOLLOWING PROBLEMS...
C
C     1. UNSYMMETRIC EQUATIONS --    SOLVE  A*X = B
C
C     2. LINEAR LEAST SQUARES  --    SOLVE  A*X = B
C                                    IN THE LEAST-SQUARES SENSE
C
C     3. DAMPED LEAST SQUARES  --    SOLVE  (   A    )*X = ( B )
C                                           ( DAMP*I )     ( 0 )
C                                    IN THE LEAST-SQUARES SENSE
C
C     WHERE  A  IS A MATRIX WITH  M  ROWS AND  N  COLUMNS,
C            B  IS AN M-VECTOR, AND
C        DAMP  IS A SCALAR (ALL QUANTITIES REAL).
C     THE MATRIX  A  IS INTENDED TO BE LARGE AND SPARSE.
C     IT IS ACCESSED BY MEANS OF SUBROUTINE CALLS OF THE FORM
```

```
30  C
31  C                CALL APROD ( MODE,M,N,X,Y,  IA,JA,AA )
32  C
33  C     WHICH MUST PERFORM THE FOLLOWING FUNCTIONS...
34  C
35  C                IF MODE = 1, COMPUTE  Y = Y + A*X.
36  C                IF MODE = 2, COMPUTE  X = X + A(TRANSPOSE)*Y.
37  C
38  C     THE VECTORS X AND Y ARE INPUT PARAMETERS IN BOTH CASES.
39  C     IF MODE = 1, Y SHOULD BE ALTERED WITHOUT CHANGING X.
40  C     IF MODE = 2, X SHOULD BE ALTERED WITHOUT CHANGING Y.
41  C     THE PARAMETERS:   IA, JA, AA.
42  C     MAY BE USED FOR WORKSPACE AS DESCRIBED BELOW.
43  C
44  C     THE RHS VECTOR  B  IS INPUT VIA  U,  AND SUBSEQUENTLY OVERWRITTEN.
45  C
46  C
47  C     NOTE.  LSQR USES AN ITERATIVE METHOD TO APPROXIMATE THE SOLUTION.
48  C     THE NUMBER OF ITERATIONS REQUIRED TO REACH A CERTAIN ACCURACY
49  C     DEPENDS STRONGLY ON THE SCALING OF THE PROBLEM.  POOR SCALING OF
50  C     THE ROWS OR COLUMNS OF  A  SHOULD THEREFORE BE AVOIDED WHERE
51  C     POSSIBLE.
52  C
53  C     FOR EXAMPLE, IN PROBLEM 1 THE SOLUTION IS UNALTERED BY
54  C     ROW-SCALING.  IF A ROW OF  A  IS VERY SMALL OR LARGE COMPARED TO
55  C     THE OTHER ROWS OF  A,  THE CORRESPONDING ROW OF  ( A  B )  SHOULD
56  C     BE SCALED UP OR DOWN.
57  C
58  C     IN PROBLEMS 1 AND 2, THE SOLUTION  X  IS EASILY RECOVERED
59  C     FOLLOWING COLUMN-SCALING.  IN THE ABSENCE OF BETTER INFORMATION,
60  C     THE NONZERO COLUMNS OF  A  SHOULD BE SCALED SO THAT THEY ALL HAVE
61  C     THE SAME EUCLIDEAN NORM (E.G.  1.0).
62  C
63  C     IN PROBLEM 3, THERE IS NO FREEDOM TO RE-SCALE IF  DAMP  IS
64  C     NONZERO.  HOWEVER, THE VALUE OF  DAMP  SHOULD BE ASSIGNED ONLY
65  C     AFTER ATTENTION HAS BEEN PAID TO THE SCALING OF  A.
66  C
67  C     THE PARAMETER  DAMP  IS INTENDED TO HELP REGULARIZE
68  C     ILL-CONDITIONED SYSTEMS, BY PREVENTING THE TRUE SOLUTION FROM
69  C     BEING VERY LARGE.  ANOTHER AID TO REGULARIZATION IS PROVIDED BY
70  C     THE PARAMETER  ACOND,  WHICH MAY BE USED TO TERMINATE ITERATIONS
71  C     BEFORE THE COMPUTED SOLUTION BECOMES VERY LARGE.
72  C
73  C
74  C     NOTATION
75  C     --------
76  C
77  C     THE FOLLOWING QUANTITIES ARE USED IN DISCUSSING THE SUBROUTINE
78  C     PARAMETERS...
79  C
80  C     ABAR   =  (   A    ),          BBAR  =  ( B )
81  C               ( DAMP*I )                   ( 0 )
82  C
```

```
C     R      =  B  -  A*X,          RBAR  =  BBAR  -  ABAR*X
C
C     RNORM  =  SQRT( NORM(R)**2  +  DAMP**2 * NORM(X)**2 )
C            =  NORM( RBAR )
C
C     RELPR  =  THE RELATIVE PRECISION OF FLOATING-POINT ARITHMETIC
C               ON THE MACHINE BEING USED.  FOR EXAMPLE, ON THE IBM 370,
C               RELPR  IS ABOUT 1.0E-6 AND 1.0D-16 IN SINGLE AND DOUBLE
C               PRECISION RESPECTIVELY.
C
C     LSQR  MINIMIZES THE FUNCTION  RNORM  WITH RESPECT TO  X.
C
C
C     PARAMETERS
C     ----------
C
C     M         INPUT      THE NUMBER OF ROWS IN  A.
C
C     N         INPUT      THE NUMBER OF COLUMNS IN  A.
C
C     APROD     EXTERNAL   SEE ABOVE.
C
C     DAMP      INPUT      THE DAMPING PARAMETER FOR PROBLEM 3 ABOVE.
C                          (DAMP  SHOULD BE 0.0 FOR PROBLEMS 1 AND 2.)
C                          IF THE SYSTEM  A*X = B  IS INCOMPATIBLE, VALUES
C                          OF  DAMP  IN THE RANGE 0 TO SQRT(RELPR)*NORM(A)
C                          WILL PROBABLY HAVE A NEGLIGIBLE EFFECT.
C                          LARGER VALUES OF  DAMP  WILL TEND TO DECREASE
C                          THE NORM OF  X  AND TO REDUCE THE NUMBER OF
C                          ITERATIONS REQUIRED BY LSQR.
C
C                          THE WORK PER ITERATION AND THE STORAGE NEEDED
C                          BY LSQR ARE THE SAME FOR ALL VALUES OF  DAMP.
C
C     IA        INPUT      CONTAINS ROW    INFORMATION OF      ARRAY   A.
C     JA        INPUT      CONTAINS COLUMN INFORMATION OF A ROW WITHIN   A.
C     AA        INPUT      THE  A  ARRAY.
C
C               NOTE.  LSQR DOES NOT EXPLICITLY USE THE PREVIOUS FOUR
C               PARAMETERS, BUT PASSES THEM TO SUBROUTINE APROD FOR
C               POSSIBLE USE AS WORKSPACE.  IF APROD DOES NOT NEED
C               IW  OR  RW,  THE VALUES  LENIW = 1  OR  LENRW = 1  SHOULD
C               BE USED, AND THE ACTUAL PARAMETERS CORRESPONDING TO
C               IW  OR  RW  MAY BE ANY CONVENIENT ARRAY OF SUITABLE TYPE.
C
C     U(M)      INPUT      THE RHS VECTOR  B.  BEWARE THAT  U  IS
C                          OVER-WRITTEN BY LSQR.
C
C     V(N)      WORKSPACE
C     W(N)      WORKSPACE
C
C     X(N)      OUTPUT     RETURNS THE COMPUTED SOLUTION  X.
C
```

```
C     SE(N)   OUTPUT     RETURNS STANDARD ERROR ESTIMATES FOR THE
C                        COMPONENTS OF  X.  FOR EACH  I,  SE(I)  IS SET
C                        TO THE VALUE  RNORM * SQRT( SIGMA(I,I) / T ),
C                        WHERE  SIGMA(I,I)  IS AN ESTIMATE OF THE I-TH
C                        DIAGONAL OF THE INVERSE OF ABAR(TRANSPOSE)*ABAR
C                        AND  T = 1      IF  M .LE. N,
C                             T = M - N  IF  M .GT. N  AND  DAMP = 0,
C                             T = M      IF  DAMP .NE. 0.
C
C     ATOL    INPUT      AN ESTIMATE OF THE RELATIVE ERROR IN THE DATA
C                        DEFINING THE MATRIX  A.  FOR EXAMPLE,
C                        IF  A  IS ACCURATE TO ABOUT 6 DIGITS, SET
C                        ATOL = 1.0E-6 .
C
C     BTOL    INPUT      AN ESTIMATE OF THE RELATIVE ERROR IN THE DATA
C                        DEFINING THE RHS VECTOR  B.  FOR EXAMPLE,
C                        IF  B  IS ACCURATE TO ABOUT 6 DIGITS, SET
C                        BTOL = 1.0E-6 .
C
C     CONLIM  INPUT      AN UPPER LIMIT ON  COND(ABAR),  THE APPARENT
C                        CONDITION NUMBER OF THE MATRIX  ABAR.
C                        ITERATIONS WILL BE TERMINATED IF A COMPUTED
C                        ESTIMATE OF  COND(ABAR)  EXCEEDS  CONLIM.
C                        THIS IS INTENDED TO PREVENT CERTAIN SMALL OR
C                        ZERO SINGULAR VALUES OF  A  OR  ABAR  FROM
C                        COMING INTO EFFECT AND CAUSING UNWANTED GROWTH
C                        IN THE COMPUTED SOLUTION.
C
C                        CONLIM  AND  DAMP  MAY BE USED SEPARATELY OR
C                        TOGETHER TO REGULARIZE ILL-CONDITIONED SYSTEMS.
C
C                        NORMALLY,  CONLIM  SHOULD BE IN THE RANGE
C                        1000  TO  1/RELPR.
C                        SUGGESTED VALUE --
C                        CONLIM = 1/(100*RELPR)  FOR COMPATIBLE SYSTEMS,
C                        CONLIM = 1/(10*SQRT(RELPR))  FOR LEAST SQUARES.
C
C              NOTE.  IF THE USER IS NOT CONCERNED ABOUT THE PARAMETERS
C              ATOL, BTOL  AND  CONLIM,  ANY OR ALL OF THEM MAY BE SET
C              TO ZERO.  THE EFFECT WILL BE THE SAME AS THE VALUES
C              RELPR, RELPR  AND  1/RELPR  RESPECTIVELY.
C
C     ITNLIM  INPUT      AN UPPER LIMIT ON THE NUMBER OF ITERATIONS.
C                        SUGGESTED VALUE --
C                        ITNLIM = N/2     FOR WELL CONDITIONED SYSTEMS,
C                        ITNLIM = 4*N     OTHERWISE.
C
C     NOUT    INPUT      FILE NUMBER FOR PRINTER.  IF POSITIVE,
C                        A SUMMARY WILL BE PRINTED ON FILE  NOUT.
C
C     ISTOP   OUTPUT     AN INTEGER GIVING THE REASON FOR TERMINATION...
C
C                0       X = 0  IS THE EXACT SOLUTION.
```

```
C                       NO ITERATIONS WERE PERFORMED.
C
C                  1    THE EQUATIONS  A*X = B  ARE PROBABLY
C                       COMPATIBLE.  NORM(A*X - B)  IS SUFFICIENTLY
C                       SMALL, GIVEN THE VALUES OF  ATOL  AND  BTOL.
C
C                  2    THE SYSTEM  A*X = B  IS PROBABLY NOT
C                       COMPATIBLE.  A LEAST-SQUARES SOLUTION HAS
C                       BEEN OBTAINED WHICH IS SUFFICIENTLY ACCURATE,
C                       GIVEN THE VALUE OF  ATOL.
C
C                  3    AN ESTIMATE OF  COND(ABAR)  HAS EXCEEDED
C                       CONLIM.  THE SYSTEM  A*X = B  APPEARS TO BE
C                       ILL-CONDITIONED.  OTHERWISE, THERE COULD BE AN
C                       AN ERROR IN SUBROUTINE APROD.
C
C                  4    THE EQUATIONS  A*X = B  ARE PROBABLY
C                       COMPATIBLE.  NORM(A*X - B)  IS AS SMALL AS
C                       SEEMS REASONABLE ON THIS MACHINE.
C
C                  5    THE SYSTEM  A*X = B  IS PROBABLY NOT
C                       COMPATIBLE.  A LEAST-SQUARES SOLUTION HAS
C                       BEEN OBTAINED WHICH IS AS ACCURATE AS SEEMS
C                       REASONABLE ON THIS MACHINE.
C
C                  6    COND(ABAR)  SEEMS TO BE SO LARGE THAT THERE IS
C                       NOT MUCH POINT IN DOING FURTHER ITERATIONS,
C                       GIVEN THE PRECISION OF THIS MACHINE.
C                       THERE COULD BE AN ERROR IN SUBROUTINE APROD.
C
C                  7    THE ITERATION LIMIT  ITNLIM  WAS REACHED.
C
C     ANORM   OUTPUT    AN ESTIMATE OF THE FROBENIUS NORM OF  ABAR.
C                       THIS IS THE SQUARE-ROOT OF THE SUM OF SQUARES
C                       OF THE ELEMENTS OF  ABAR.
C                       IF  DAMP  IS SMALL AND IF THE COLUMNS OF  A
C                       HAVE ALL BEEN SCALED TO HAVE LENGTH  1.0,
C                       ANORM  SHOULD INCREASE TO ROUGHLY  SQRT(N).
C                       A RADICALLY DIFFERENT VALUE FOR  ANORM  MAY
C                       INDICATE AN ERROR IN SUBROUTINE APROD (THERE
C                       MAY BE AN INCONSISTENCY BETWEEN MODES 1 AND 2).
C
C     ACOND   OUTPUT    AN ESTIMATE OF  COND(ABAR),  THE CONDITION
C                       NUMBER OF  ABAR.  A VERY HIGH VALUE OF  ACOND
C                       MAY AGAIN INDICATE AN ERROR IN APROD.
C
C     RNORM   OUTPUT    AN ESTIMATE OF THE FINAL VALUE OF NORM(RBAR),
C                       THE FUNCTION BEING MINIMIZED (SEE NOTATION
C                       ABOVE).  THIS WILL BE SMALL IF  A*X = B  HAS
C                       A SOLUTION.
C
C     ARNORM  OUTPUT    AN ESTIMATE OF THE FINAL VALUE OF
C                       NORM( ABAR(TRANSPOSE)*RBAR ), THE NORM OF
```

```
C                       THE RESIDUAL FOR THE USUAL NORMAL EQUATIONS.
C                       THIS SHOULD BE SMALL IN ALL CASES.  (ARNORM
C                       WILL OFTEN BE SMALLER THAN THE TRUE VALUE
C                       COMPUTED FROM THE OUTPUT VECTOR  X.)
C
C     XNORM   OUTPUT    AN ESTIMATE OF THE NORM OF THE FINAL
C                       SOLUTION VECTOR  X.
C
C
C     SUBROUTINES AND FUNCTIONS USED
C     ------------------------------
C
C     USER        APROD
C     LSQR        NORMLZ
C     BLAS        SCOPY,SNRM2,SSCAL  (SEE LAWSON ET AL. BELOW)
C                 (SNRM2 IS USED ONLY IN NORMLZ)
C     FORTRAN     ABS,MOD,SQRT
C
C
C     PRECISION
C     ---------
C
C     THE NUMBER OF ITERATIONS REQUIRED BY LSQR WILL USUALLY DECREASE
C     IF THE COMPUTATION IS PERFORMED IN HIGHER PRECISION.  TO CONVERT
C     LSQR  AND  NORMLZ  BETWEEN SINGLE- AND DOUBLE-PRECISION, CHANGE
C     THE WORDS
C                 SCOPY, SNRM2, SSCAL
C                 ABS, REAL, SQRT
C     TO THE APPROPRIATE BLAS AND FORTRAN EQUIVALENTS.
C
C
C     REFERENCES
C     ----------
C
C     PAIGE, C.C. AND SAUNDERS, M.A.  LSQR©D AN ALGORITHM FOR SPARSE
C        LINEAR EQUATIONS AND SPARSE LEAST SQUARES.
C        ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE 8, 1 (MARCH 1982).
C
C     LAWSON, C.L., HANSON, R.J., KINCAID, D.R. AND KROGH, F.T.
C        BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE.
C        ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE 5, 3 (SEPT 1979),
C        308-323 AND 324-325.
C
C
C     LSQR.       THIS VERSION DATED 22 FEBRUARY 1982.
C     ------------------------------------------------------------------
C
C     FUNCTIONS AND LOCAL VARIABLES
C
      INTEGER     I,ITN,MOD,NCONV,NSTOP
      REAL        ABS,SQRT
      REAL        ALFA,BBNORM,BETA,BNORM,
     1            CS,CS1,CS2,CTOL,DAMPSQ,DDNORM,DELTA,
```

```
     2            GAMMA,GAMBAR,ONE,PHI,PHIBAR,PSI,
     3            RES1,RES2,RHO,RHOBAR,RHBAR1,RHBAR2,RHS,RTOL,
     4            SN,SN1,SN2,T,TAU,TEST1,TEST2,TEST3,
     5            THETA,T1,T2,T3,XXNORM,Z,ZBAR,ZERO
C
C
C     INITIALIZE.
C
      IF (NOUT .GT. 0)
     1   WRITE(NOUT, 1000) M,N,DAMP,ATOL,CONLIM,BTOL,ITNLIM
      ZERO   = 0.0
      ONE    = 1.0
      CTOL   = ZERO
      IF (CONLIM .GT. ZERO) CTOL = ONE/CONLIM
      DAMPSQ = DAMP**2
      ANORM  = ZERO
      ACOND  = ZERO
      BBNORM = ZERO
      DDNORM = ZERO
      RES2   = ZERO
      XNORM  = ZERO
      XXNORM = ZERO
      CS2    = -ONE
      SN2    = ZERO
      Z      = ZERO
      ITN    = 0
      ISTOP  = 0
      NSTOP  = 0
C
      DO 10 I = 1, N
         V(I) = ZERO
         X(I) = ZERO
        SE(I) = ZERO
   10 CONTINUE
C
C     SET UP THE FIRST VECTORS FOR THE BIDIAGONALIZATION.
C     THESE SATISFY   BETA*U = B,   ALFA*V = A(TRANSPOSE)*U.
C
      CALL NORMLZ( M,U,BETA )
      CALL APROD ( 2,M,N,V,U,  IA,JA,AA )
      CALL NORMLZ( N,V,ALFA )
      CALL SCOPY ( N,V,1,W,1 )
C
      RHOBAR = ALFA
      PHIBAR = BETA
      BNORM  = BETA
      RNORM  = BETA
      ARNORM = ALFA*BETA
      IF (ARNORM .LE. ZERO) GO TO 800
      IF (NOUT   .LE.  0  ) GO TO 100
      IF (DAMPSQ .LE. ZERO) WRITE(NOUT, 1200)
      IF (DAMPSQ .GT. ZERO) WRITE(NOUT, 1300)
      TEST1  = ONE
```

```
348         TEST2  = ALFA/BETA
349         WRITE(NOUT, 1500) ITN,X(1),RNORM,TEST1,TEST2
350         WRITE(NOUT, 1600)
351   C
352   C     ------------------------------------------------------------------
353   C     MAIN ITERATION LOOP.
354   C     ------------------------------------------------------------------
355     100 ITN = ITN + 1
356   C
357   C     PERFORM THE NEXT STEP OF THE BIDIAGONALIZATION TO OBTAIN THE
358   C     NEXT  BETA, U, ALFA, V.  THESE SATISFY THE RELATIONS
359   C                BETA*U  =  A*V  -  ALFA*U,
360   C                ALFA*V  =  A(TRANSPOSE)*U  -  BETA*V.
361   C
362         CALL SSCAL ( M,(-ALFA),U,1 )
363         CALL APROD ( 1,M,N,V,U,  IA,JA,AA )
364         CALL NORMLZ( M,U,BETA )
365         BBNORM = BBNORM + ALFA**2 + BETA**2 + DAMPSQ
366         CALL SSCAL ( N,(-BETA),V,1 )
367         CALL APROD ( 2,M,N,V,U,  IA,JA,AA )
368         CALL NORMLZ( N,V,ALFA )
369   C
370   C
371   C     USE A PLANE ROTATION TO ELIMINATE THE DAMPING PARAMETER.
372   C     THIS ALTERS THE DIAGONAL (RHOBAR) OF THE LOWER-BIDIAGONAL MATRIX.
373   C
374         RHBAR2 = RHOBAR**2 + DAMPSQ
375         RHBAR1 = SQRT(RHBAR2)
376         CS1    = RHOBAR/RHBAR1
377         SN1    = DAMP/RHBAR1
378         PSI    = SN1*PHIBAR
379         PHIBAR = CS1*PHIBAR
380   C
381   C
382   C     USE A PLANE ROTATION TO ELIMINATE THE SUBDIAGONAL ELEMENT (BETA)
383   C     OF THE LOWER-BIDIAGONAL MATRIX, GIVING AN UPPER-BIDIAGONAL MATRIX.
384   C
385         RHO    = SQRT(RHBAR2 + BETA**2)
386         CS     = RHBAR1/RHO
387         SN     = BETA/RHO
388         THETA  =  SN*ALFA
389         RHOBAR = -CS*ALFA
390         PHI    =  CS*PHIBAR
391         PHIBAR =  SN*PHIBAR
392         TAU    =  SN*PHI
393   C
394   C
395   C     UPDATE  X, W  AND THE STANDARD ERROR ESTIMATES.
396   C
397         T1 =    PHI/RHO
398         T2 = -THETA/RHO
399         T3 =    ONE/RHO
400   C
```

```
401         DO 200 I = 1, N
402            T     = W(I)
403            X(I)  = T1*T + X(I)
404            W(I)  = T2*T + V(I)
405            T     =(T3*T)**2
406            SE(I) = T + SE(I)
407            DDNORM= T + DDNORM
408       200 CONTINUE
409   C
410   C
411   C     USE A PLANE ROTATION ON THE RIGHT TO ELIMINATE THE
412   C     SUPER-DIAGONAL ELEMENT (THETA) OF THE UPPER-BIDIAGONAL MATRIX.
413   C     THEN USE THE RESULT TO ESTIMATE  NORM(X).
414   C
415         DELTA  =  SN2*RHO
416         GAMBAR = -CS2*RHO
417         RHS    = PHI - DELTA*Z
418         ZBAR   = RHS/GAMBAR
419         XNORM  = SQRT(XXNORM + ZBAR**2)
420         GAMMA  = SQRT(GAMBAR**2 + THETA**2)
421         CS2    = GAMBAR/GAMMA
422         SN2    = THETA/GAMMA
423         Z      = RHS/GAMMA
424         XXNORM = XXNORM + Z**2
425   C
426   C
427   C     TEST FOR CONVERGENCE.
428   C     FIRST, ESTIMATE THE NORM AND CONDITION OF THE MATRIX  ABAR,
429   C     AND THE NORMS OF  RBAR  AND  ABAR(TRANSPOSE)*RBAR.
430   C
431         ANORM  = SQRT(BBNORM)
432         ACOND  = ANORM*SQRT(DDNORM)
433         RES1   = PHIBAR**2
434         RES2   = RES2 + PSI**2
435         RNORM  = SQRT(RES1 + RES2)
436         ARNORM = ALFA*ABS(TAU)
437   C
438   C     NOW USE THESE NORMS TO ESTIMATE CERTAIN OTHER QUANTITIES,
439   C     SOME OF WHICH WILL BE SMALL NEAR A SOLUTION.
440   C
441         TEST1  = RNORM/BNORM
442         TEST2  = ARNORM/(ANORM*RNORM)
443         TEST3  = ONE/ACOND
444         T1     = TEST1/(ONE + ANORM*XNORM/BNORM)
445         RTOL   = BTOL +  ATOL*ANORM*XNORM/BNORM
446   C
447   C     THE FOLLOWING TESTS GUARD AGAINST EXTREMELY SMALL VALUES OF
448   C     ATOL, BTOL  OR  CTOL.  (THE USER MAY HAVE SET ANY OR ALL OF
449   C     THE PARAMETERS  ATOL, BTOL, CONLIM  TO ZERO.)
450   C     THE EFFECT IS EQUIVALENT TO THE NORMAL TESTS USING
451   C     ATOL = RELPR,  BTOL = RELPR,  CONLIM = 1/RELPR.
452   C
453         T3 = ONE + TEST3
```

```
454         T2 = ONE + TEST2
455         T1 = ONE + T1
456         IF (ITN .GE. ITNLIM) ISTOP = 7
457         IF (T3  .LE. ONE   ) ISTOP = 6
458         IF (T2  .LE. ONE   ) ISTOP = 5
459         IF (T1  .LE. ONE   ) ISTOP = 4
460   C
461   C     ALLOW FOR TOLERANCES SET BY THE USER.
462   C
463         IF (TEST3 .LE. CTOL) ISTOP = 3
464         IF (TEST2 .LE. ATOL) ISTOP = 2
465         IF (TEST1 .LE. RTOL) ISTOP = 1
466   C     ==================================================================
467   C
468   C     SEE IF IT IS TIME TO PRINT SOMETHING.
469   C
470         IF (NOUT .LE. 0) GO TO 600
471         IF (M.LE.40 .OR. N.LE.40) GO TO 400
472         IF (ITN .LE. 10)          GO TO 400
473         IF (ITN .GE. ITNLIM-10)   GO TO 400
474         IF (MOD(ITN,10) .EQ. 0)   GO TO 400
475         IF (TEST3 .LE.  2.0*CTOL) GO TO 400
476         IF (TEST2 .LE. 10.0*ATOL) GO TO 400
477         IF (TEST1 .LE. 10.0*RTOL) GO TO 400
478         GO TO 600
479   C
480   C     PRINT A LINE FOR THIS ITERATION.
481   C
482     400 WRITE(NOUT, 1500) ITN,X(1),RNORM,TEST1,TEST2,ANORM,ACOND
483         IF (MOD(ITN,10) .EQ. 0) WRITE(NOUT, 1600)
484   C     ==================================================================
485   C
486   C     STOP IF APPROPRIATE.
487   C     THE CONVERGENCE CRITERIA ARE REQUIRED TO BE MET ON  NCONV
488   C     CONSECUTIVE ITERATIONS, WHERE  NCONV  IS SET BELOW.
489   C     SUGGESTED VALUE --   NCONV = 1, 2  OR  3.
490   C
491     600 IF (ISTOP .EQ. 0) NSTOP = 0
492         IF (ISTOP .EQ. 0) GO TO 100
493         NCONV = 1
494         NSTOP = NSTOP + 1
495         IF (NSTOP .LT. NCONV  .AND.  ITN .LT. ITNLIM) ISTOP = 0
496         IF (ISTOP .EQ. 0) GO TO 100
497   C     ------------------------------------------------------------------
498   C     END OF ITERATION LOOP.
499   C     ------------------------------------------------------------------
500   C
501   C
502   C     FINISH OFF THE STANDARD ERROR ESTIMATES.
503   C
504         T = ONE
505         IF (M .GT. N) T = M - N
506         IF (DAMPSQ .GT. ZERO) T = M
```

```
      T = RNORM/SQRT(T)
C
      DO 700 I = 1, N
         SE(I) = T*SQRT(SE(I))
  700 CONTINUE
C
C     PRINT THE STOPPING CONDITION.
C
  800 IF (NOUT .LE. 0) GO TO 900
      WRITE(NOUT, 1900) ITN,ISTOP
      IF (ISTOP .EQ. 0) WRITE(NOUT, 2000)
      IF (ISTOP .EQ. 1) WRITE(NOUT, 2100)
      IF (ISTOP .EQ. 2) WRITE(NOUT, 2200)
      IF (ISTOP .EQ. 3) WRITE(NOUT, 2300)
      IF (ISTOP .EQ. 4) WRITE(NOUT, 2400)
      IF (ISTOP .EQ. 5) WRITE(NOUT, 2500)
      IF (ISTOP .EQ. 6) WRITE(NOUT, 2600)
      IF (ISTOP .EQ. 7) WRITE(NOUT, 2700)
  900 RETURN
C     ------------------------------------------------------------------
C
 1000 FORMAT(
     1   // 25X, 46HLSQR   --   LEAST-SQUARES SOLUTION OF  A*X = B
     2   // 25X, 18HTHE MATRIX  A  HAS, I6, 11H ROWS   AND, I6, 5H COLS
     3    / 25X, 36HTHE DAMPING PARAMETER IS    DAMP   =, 1PE10.2
     4   // 25X,  8HATOL   =, 1PE10.2, 10X, 8HCONLIM =, 1PE10.2
     5    / 25X,  8HBTOL   =, 1PE10.2, 10X, 8HITNLIM =, I10)
 1200 FORMAT(// 3X, 3HITN, 9X, 4HX(1), 14X, 8HFUNCTION, 7X,
     1   45HCOMPATIBLE INCOMPATIBLE    NORM(A)    COND(A) /)
 1300 FORMAT(// 3X, 3HITN, 9X, 4HX(1), 14X, 8HFUNCTION, 7X,
     1   45HCOMPATIBLE INCOMPATIBLE NORM(ABAR) COND(ABAR) /)
 1500 FORMAT(I6, 1PE20.10, 1PE19.10, 1P2E13.3, 1P2E11.2)
 1600 FORMAT(1X)
 1900 FORMAT(/ 20H NO. OF ITERATIONS =, I6,
     1   8X, 21H STOPPING CONDITION =, I3)
 2000 FORMAT(/ 52H THE EXACT SOLUTION IS  X = 0.                        )
 2100 FORMAT(/ 52H A*X - B  IS SMALL ENOUGH, GIVEN  ATOL, BTOL          )
 2200 FORMAT(/ 52H THE LEAST-SQRS SOLN IS GOOD ENOUGH, GIVEN  ATOL      )
 2300 FORMAT(/ 52H THE ESTIMATE OF  COND(ABAR)  HAS EXCEEDED  CONLIM    )
 2400 FORMAT(/ 52H A*X - B   IS SMALL ENOUGH FOR THIS MACHINE           )
 2500 FORMAT(/ 52H THE LEAST-SQRS SOLN IS GOOD ENOUGH FOR THIS MACHINE)
 2600 FORMAT(/ 52H COND(ABAR)   SEEMS TO BE TOO LARGE FOR THIS MACHINE)
 2700 FORMAT(/ 52H THE ITERATION LIMIT HAS BEEN REACHED                 )
C     END OF LSQR
      END
C-----------------------------------------------------------------------
      SUBROUTINE NORMLZ( N,X,BETA )
      INTEGER    N
      REAL       X(N),BETA
C
C     NORMLZ  IS REQUIRED BY SUBROUTINE LSQR.  IT COMPUTES THE
C     EUCLIDEAN NORM OF  X  AND RETURNS THE VALUE IN  BETA.
C     IF  X  IS NONZERO, IT IS SCALED SO THAT NORM(X) = 1.
```

```
560   C
561   C     FUNCTIONS AND SUBROUTINES
562   C
563   C     BLAS        SNRM2,SSCAL
564   C
565         REAL        ONE,SNRM2,ZERO
566   C
567   C
568         ZERO = 0.0
569         ONE  = 1.0
570         BETA = SNRM2( N,X,1 )
571         IF (BETA .GT. ZERO) CALL SSCAL( N,(ONE/BETA),X,1 )
572         RETURN
573   C
574   C     END OF NORMLZ
575         END
```

6.3.8 SQRTT.FOR

```
1       function sqrtt (z)              ! assure proper branch
2       complex  sqrtt, z, cmplx
3       data  pi / 3.1415926E0 /
4
5       x =  real (z)
6       y = aimag (z)
7       call polar (x,y,r,a,1)          ! [0,2)
8       if (r.eq.0.0) then
9            sqrtt = cmplx (0.0, 0.0)
10      else
11           r = sqrt (r)
12           a = a*pi*0.5               ! [0, pi)
13           sqrtt = cmplx (r*cos(a), r*sin(a))
14      end if
15      return
16      end
```

6.3.9 POLAR.FOR

```
c-------------------------------------------------------------
      subroutine polar (x, y, r, a, icase)
c       given:  x,y
c     discern:  r,a
c       where:  r>0,  x=r*cos(a*pi), y=r*sin(a*pi)
c      case 1:  a ~ [ 0, 2)
c           2:  a ~ (-1, 1]
c           3:  a ~ [ 0, 2*pi)              !  radians
c           4:  a ~ [ 0, 360)               !  degrees

      data  pi / 3.14159265E0 /

      if (y .eq. 0.0) then                  !  x axis
          if (x .lt. 0.0) then
              a = 1.0
              r = -x
          else
              a = 0.0
              r = x
          end if
          goto 1
      end if
      if (x .eq. 0.0) then                  !  y axis
          if (y .lt. 0.0) then
              a = 1.5
              r = -y
          else
              a = 0.5
              r = y
          end if
          goto 1
      end if
      xx = abs (x)
      yy = abs (y)
      if (xx .lt. yy) then
          a = x/y                           !  ratio < 1
          r = yy * sqrt (a*a+1.0)
          a = 0.5 - atan (a) /pi            !  top
          if (y .lt. 0.0)  a=a+1.0          !  bottom
      else
          a = y/x                           !  ratio < 1
          r = xx * sqrt (a*a+1.0)
          a = atan (a) /pi
          if (x .lt. 0.0) then              !  lhs
              a = a+1.0
          else if (y .lt. 0.0) then
              a = a+2.0                     !  rhs
          end if
      end if

```

```
51      1 if (icase .eq. 1) return
52        if (icase .eq. 2) then
53            if (a .gt. 1.0)  a=a-2.0
54            return
55        end if
56        if (icase .eq. 3) then                ! radians
57            a = a*pi
58        else                                  ! degrees
59            a = a*180.0
60        end if
61
62        return
63        end
```

6.3.10 IINDEX.FOR

```
1   c     Discern the sequential index (k) for the (i,j) matrix element,
2   c     where the matrix is symmetric,  a(i,j) = a(j,i) = a(k),
3   c     but where only the upper/lower triangle is stored,  k=k(i,j).
4   c     N ' size of the square matrix, i.e.,   A ' a(i,j)   is NxN.
5   c     L ' indicates the storage format, 1-4.
6
7         function iindex (l,n,i,j)
8         integer  iindex, l,n,i,j
9
10        if (l.lt.1 .or. n.lt.1 .or. i.lt.1 .or. j.lt.1 .or.
11       &    l.gt.6 .or.             i.gt.n .or. j.gt.n     ) then
12            call exit (2)                ! 2'error,  4'severe error
13        end if
14
15        mn = min (i,j)
16        mx = max (i,j)
17        goto (1,2,3,4,5,6), l '
18
19  c     Symmetric Matix ------------------------------------
20
21  c     Lower triangular matrix, j<=i, stored column by column.
22       1 iindex = mx + ((n+n-mn)*(mn-1))/2
23        return
24
25  c     Lower triangular matrix, j<=i, stored row by row.
26       2 iindex = mn + ((mx-1)*mx)/2
27        return
28
29  c     Upper triangular matrix, i<=j, stored column by column.
30       3 iindex = mn + ((mx-1)*mx)/2
31        return
32
33  c     Upper triangular matrix, i<=j, stored row by row.
34       4 iindex = mx + ((n+n-mn)*(mn-1))/2
35        return
36
37  c     Asymmetric Matrix -----------------------------------
38
39  c     Full matrix, stored column by column.
40       5 iindex = i + (j-1)*n
41        return
42
43  c     Full matrix, stored row by row.
44       6 iindex = j + (i-1)*n
45        return
46
47        end
```

6.3.11 ISTIME.FOR

```
1       function istime (i)         ! dummy argument
2       integer  istime, iftime, i, it
3       logical  first
4       data     first /.true./
5
6       entry iftime (i)
7       if (first) then             ! initial
8           first = .false.
9           call times (-1,it)      ! initialize cpu clock
10      end if
11      call times (1,it)           ! query elapsed cpu time (centi-seconds)
12      istime = it*10              ! milli-seconds
13      return
14      end
```

6.3.12 TIMES.FOR

```
1       subroutine times (is, it)
2       include   'sys$library:libdef.for'
3       include   'sys$library:sigdef.for'
4  c*   include   'sys$library:mthdef.for'
5  c*   include   'sys$library:fordef.for'
6
7       parameter (io=6,  nh=2)
8       integer*4  handle(nh), status
9       save       handle
10
11 c-------------------------------------------------------------------
12 c     This routine maintains or keeps track of several distinct
13 c          or separate and independent clocks.
14 c     Each clock is equivalent to any other clock,
15 c          i.e., among themselves.
16 c     The quantity of clocks is restricted by the magnitude
17 c          or dimension of the array, "handle", i.e., "nh".
18 c     The clocks are indexed by the magnitude of "is" = |is|.
19 c     A clock is initalized whenever "is" is negative;
20 c          all clocks are initalized whenever "is" is zero.
21 c     Time displacements or intervals, "it", are:
22 c          a)   evaluated whenever "is" is positive
23 c          b)   expressed in units of:  centi-seconds
24 c     -----------------------------------------------------------------
25
26      if (is.eq.0) then
27          do i=1,nh
28              status = lib$init_timer (handle(i))
29              if (status.ne.ss$_normal) then
30                  write (io,6)
31                  stop
32              end if
33          end do
34          return
35      end if
36
37      if (iabs(is).gt.nh) then
38          write (io,8) is
39          stop
40      end if
41
42      if (is.lt.0) then
43          status = lib$init_timer (handle(-is))
44          if (status.eq.ss$_normal) return
45          write (io,6)
46          stop
47      else
48          status = lib$stat_timer (2,it,handle(is))
49          if (status.eq.ss$_normal) return
50          write (io,7)
```

```
51            stop
52          end if
53
54        6 format (' times,  error:   clock initalization problem')
55        7 format (' times,  error:   clock evaluation problem')
56        8 format (' times,  error:   attempting use of non-existing'
57         &       ,' clock #', i3)
58          end
```

7. Acknowledgments

It is the author's pleasure to acknowledge Drs. Stanley Ruthberg and George Candela for having suggested and supported this project. On a melancholy note, we mourn the loss of Dr. Ruthberg, whose untimely death occurred during the course of the work which led to the preparation of this manuscript. His vitality and friendship are missed. On a much happier note, the author appreciates the questions and suggestions from Deane Chandler-Horowitz and Pradip Dutta, who have been among the first to apply the software package. The author also appreciates the discussions with M. Carroll Croarkin regarding things statistical.

8. References

[1] M. Born and E. Wolf, *Principles of Optics*, sixth corrected edition (Pergamon Press, Oxford, 1980).

[2] R. M. A. Azzam and N. M. Bashara, *Ellipsometry and Polarized Light*, paperback edition (North-Holland, Amsterdam, 1987).

[3] Z. Knittl, *Optics of Thin Films* (John Wiley & Sons, Ltd., London, 1976).

[4] J. Lekner, *Theory of Reflection* (Martinus Nijhoff Publishers, Dordrecht, Netherlands, 1987).

[5] *Handbook of Optical Constants of Solids*, edited by E. W. Palik (Academic Press, Orlando, 1985).

[6] F. E. Jones, "The Refractivity of Air," *J. Res. Natl. Bur. Stand. (U.S.)* **86**, 27-32 (January-February 1981).

[7] F. L. McCrackin, "A Fortran Program for Analysis of Ellipsometer Measurements," *NBS Tech. Note 479*, Natl. Bur. Stand. (U.S. Department of Commerce), April 1969.

[8] J. Mouchart, "Calcul Matriciel Appliqué aux Couches Minces Élaboration des Dérivées Partielles du Coefficient de Réflexion," *Optica Acta* **27**, 401-408 (1980).

[9] J. Humlíček, "Evaluation of Derivatives of Reflection and Transmittance by Stratified Structures and Solution of the Reverse Problem of Ellipsometry," *Optica Acta* **30**, 97-105 (1983).

[10] J. Humlíček, "Sensitivity Extrema in Multiple-Angle Ellipsometry," *J. Opt. Soc. Am.* **2**, 713-722 (1985).

[11] G. H. Bu-Abbud, N. M. Bashara, and J. A. Woolam, "Variable Wavelength, Variable Angle Ellipsometry Including a Sensitivities Correlation Test," *Thin Solid Films* **138**, 27-41 (1986).

[12] G. H. Bu-Abbud and N. M. Bashara, "Parameter Correlation and Precision in Multiple-Angle Ellipsometry," *Appl. Opt.* **20**, 3020-3026 (1981).

[13] M. M. Ibrahim and N. M. Bashara, "Parameter-Correlation and Computational Considerations in Multiple-Angle Ellipsometry," *J. Opt. Soc. Am.* **61**, 1622-1629 (1971).

[14] D. H. Loescher, R. J. Detry, and M. J. Clauser, "Least-Squares Analysis of the Film-Substrate Problem in Ellipsometry," *J. Opt. Soc. Am.* **61**, 1230-1235 (1971).

[15] S. Y. Kim and K. Vedam, "Proper Choice of the Error Function in Modeling Spectroellipsometric Data," *Appl. Opt.* **25**, 2013-2021 (1986).

[16] R. F. Boisvert, S. E. Howe, and D. K. Kahaner, *Guide to Available Mathematical Software*, Center for Applied Applied Mathematics, National Bureau of Standards (U.S. Department of Commerce), 1985.

[17] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, 1979).

[18] C. C. Paige and M. A. Saunders, "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *ACM Trans. Math. Softw.* **8**, 43-71 (1982).

[19] C. C. Paige and M. A. Saunders, "Algorithm 583, LSQR: Sparse Linear Equations and Least Squares Problems," *ACM Trans. Math. Softw.* **8**, 195-209, (1982).

[20] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *J. Res. Natl. Bur. Stand. (U.S.)* **49**, 409-436 (1952).

[21] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, "Variational Iterative Methods for Nonsymmetric Systems of Linear Equations," *SIAM J. Numer. Anal.* **20**, 345-357 (1983).

[22] V. Faber and T. Manteuffel, "Necessary and Sufficient Conditions for the Existence of a Conjugate Gradient Method," *SIAM J. Numer. Anal.* **21**, 352-362 (1984).

[23] R. V. Hogg and A. T. Craig, *Introduction to Mathematical Statistics*, third edition (Macmillan, New York, 1970).

[24] D. Chandler-Horowitz, "*Semiconductor Measurement Technology:* Analytical Analysis of Ellipsometric Errors," *NBS Spec. Publ. 400-78*, Natl. Bur. Stand. (U.S. Department of Commerce), May 1986.

[25] D. Chandler-Horowitz, "Ellipsometric Metrology of Ultrathin Films: Dual Angle of Incidence," *Proc. Soc. Photo-Optical & Instrum. Engrs.*, Micron and Submicron Integrated Circuit Metrology **565**, 93-97 (1985).

[26] G. A. Candela, D. Chandler-Horowitz, J. F. Marchiando, D. B. Novotny, B. J. Belzer, and M. C. Croarkin, "*Standard Reference Materials*: Preparation and Certification of SRM-2530, Ellipsometric Parameters Δ and ψ and Derived Thickness and Refractive Index of a Silicon Dioxide Layer on Silicon," *NIST Spec. Publ. 260-109*, Natl. Inst. Stand. Technol. (U.S. Department of Commerce), October 1988.

[27] "The SCD Graphics Utilities", edited by G. R. McArthur, NCAR Technical Note 166+IA, Scientific Computing Division, National Center for Atmospheric Research, Boulder, Colorado (1981).

[28] "The System Plot Package," edited by G. R. McArthur, NCAR Technical Note 162+IA, Scientific Computing Division, National Center for Atmospheric Research, Boulder, Colorado (1981).

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NIST/SP-400/83	2. Performing Organ. Report No.	3. Publication Date July 1989

4. TITLE AND SUBTITLE

Semiconductor Measurement Technology: A Software Program for Aiding the Analysis of Ellipsometric Measurements, Simple Models

5. AUTHOR(S)

J. F. Marchiando

6. PERFORMING ORGANIZATION (*If joint or other than NBS, see instructions*)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (*Street, City, State, ZIP*)

National Institute of Standards and Technology
U.S. Department of Commerce
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 89-600743

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (*A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here*)

MAIN1 is a software program for aiding the analysis of ellipsometric measurements. MAIN1 consists of a suite of routines written in FORTRAN that are used to invert the standard reflection ellipsometric equations for simple systems. Here, a system is said to be simple if the solid material sample may be adequately characterized by models which assume at least the following: (1) materials are nonmagnetic; (2) samples exhibit depth-dependent optical properties, such as one with layered or laminar structure atop a substrate that behaves like a semi-infinite half-space; (3) layers are flat and of uniform thickness; and (4) the dielectric function within each layer/substrate is isotropic, homogeneous, local, and linear. Each layer is characterized in part by a thickness (z), while the optical properties for a given material and wavelength are expressed in terms of a refractive index (n) and extinction coefficient (k). The ellipsometric equations are formulated as a standard damped nonlinear least-squares problem and then solved by an iterative method when possible. Estimates of the uncertainties associated with assigning numerical values to the model parameters are calculated as well.

12. KEY WORDS (*Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons*)

ellipsometry; FORTRAN; measurement; modeling; program; sensitivity; software; uncertainty

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES 252

15. Price

USCOMM-DC 6043-P80

☆ U.S. GOVERNMENT PRINTING OFFICE: 1989—242-311/04077

NIST Technical Publications

Periodical

Journal of Research of the National Institute of Standards and Technology—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW., Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NIST research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NIST publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NIST Interagency Reports (NISTIR)—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, MD 20899