Using Simulated Annealing in Structure Determination and Design of X-Ray Multilayers

Elsebeth Schröder
e-mail: schroder@nbivax.nbi.dk

July 8, 1992

Contents

1 Introduction

This report describes the implementation of an optimization method, the so-called simulated annealing method, in

- structure determination of x-ray multilayer mirrors from their reflectivity curves
- design of multilayer mirrors with maximum reflectivity at certain energies or at certain incident angles of an x-ray beam.

The work was carried out during my stay at the European Synchrotron Radiation Facility (ESRF), Grenoble, in May and June 1992, under the supervision of and in collaboration with P. Høghøj and E. Ziegler.

2 A Short Description of the Problem

The reflectivity of x-rays on a solid is in general very small, but it can be improved by different means. X-ray multilayers (figure 1) are artificially produced stacks of materials that possess high reflectivity due to constructive interference of the reflected beams at the individual layers. The choice of materials and the characteristics of each layer decide at which angles and at which energies of the x-ray beam the reflectivity will be at maximum. Multilayers will be used at the storage ring at the ESRF.

Other means to get relatively high reflectivity for x-rays are the use of very small incident angles (grazing), and crystal based optics.

The construction of the multilayer structures depends on at which range of energies the maximum reflectivity is desired, given a fixed incident angle θ of the x-ray beam — or, given a fixed energy of the beam, at which angles of the incident beam the maximum is desired. This is a design problem; by varying the structure of the multilayers (e.g. the thickness of the layers, the roughness at the intersections) the maximum reflectivity can be obtained. Another problem consists in infering the structure of an actual multilayer mirror given its measured reflectivity curve. This is a structure determination problem. The design problem is a special case of the structure determination problem, in that the "experimental" reflectivity curve is set to unity over the range of energies or angles where a peak in the reflectivity is desired.

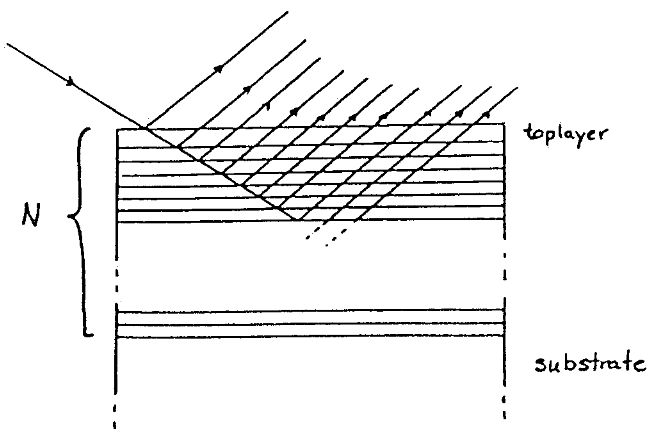


Figure 1: Multilayer structure with N layers

The reflectivity curve does not contain information about the phase of the reflected beam, and thus no direct calculation of the multilayer structure from the knowledge of its reflectivity curve is possible. One is faced with an inverse problem as in many other branches of physics, e.g. in the deconvolution of seismic reflection data. Instead, one can evaluate the theoretical reflectivity curves for different choices of structures until a structure is found whose reflectivity curve fits the experimental curve within the precision desired. This, however, is a tedious task if one needs more than a few variables to describe the multilayers.

In reality an exhaustive search among the different multilayer structures for the best fit is out of question, and one has to rely on some algorithm to evaluate a subset of all possible structures and return the best fitting structure of this subset. For this we need a measure (the cost) of how well the reflectivity curve of a suggested structure fits the experimental curve. Examples of such local optimizing algorithms are random search and pure downhill search algorithms. Usually several local (cost) minima exist among the possible structures (the structure gives a better fit than any structure obtained by a slight change of the variables). This causes the pure downhill algorithm, which only looks for better neighbouring structures, to most likely get trapped in a local minimum. A random search will not get trapped, but will, of course, only have a small chance of coming across the best structure during the search.

Modifying the pure downhill algorithm to allow for some uphill moves reduces the risk of been trapped in the first local minimum; a further search past the barriers of the local minimum is made possible, without turning the search algorithm into a random search. The algorithm "simulated annealing" allows for such uphill moves from the start of the search, and during the search it reduces the chance of uphill moves.

3 Theoretical Computation of Reflectivity

Given the structure of a multilayer, a recursive formula involving the Fresnel reflection coefficients can be used to find the theoretical values of the reflectivity [5].

The Fresnel coefficients at the top of the $j+1$th layer are given by the coefficients at the top of the jth layer and by the optical indices n_j and n_{j+1} at the two interfaces:

$$r_j^p = \frac{n_{j+1}\sin\theta_{j+1} - n_j\sin\theta_j}{n_{j+1}\sin\theta_{j+1} + n_j\sin\theta_j} \tag{1}$$

$$r_j^s = \frac{n_{j+1}\sin\theta_j - n_j\sin\theta_{j+1}}{n_{j+1}\sin\theta_j + n_j\sin\theta_{j+1}} \tag{2}$$

The angle of incidence θ_j on the jth layer is found from the angle of incidence θ_m of the incoming beam using Snell-Descartes' law

$$n_j\sin\theta_j = (n_j^2 - \cos^2\theta_m)^{1/2} \tag{3}$$

The reflected amplitude X_j at the top of a layer with reflected amplitude X_{j-1} at the bottom of the layer is

$$X_j = \frac{r_j + X_{j-1}e^{-i2\varphi}}{1 + r_j X_{j-1}e^{-i2\varphi}} \tag{4}$$

where the phase φ is

$$\varphi = (2\pi/\lambda)d_j n_j\sin\theta_j = (2\pi/\lambda)d_j(n_j^2 - \cos^2\theta_m)^{1/2} \tag{5}$$

and d_i is the thickness of the ith layer, λ is the wavelength of the incident beam. With vanishing reflected amplitude at the substrate the reflected amplitude of the whole multilayer can be found by recursive use of equation (4) until the uppermost layer. The reflectivity for an N-layer structure is then given by

$$R = |X_N|^2 \tag{6}$$

Roughness of the surfaces of the layers can be taken into account by multiplying the Fresnel coefficients (1) and (2) by the Debye-Waller factor

$$\exp(-(4\pi/\lambda)\sigma_j \sin\theta_j) \tag{7}$$

where σ_j is the roughness.

4 Simulated Annealing

To avoid that a downhill algorithm is trapped in the first local minimum found, the algorithm can be modified to allow for some uphill moves. The optimization algorithm "simulated annealing" is such an algorithm; it is based on an analogy of optimization problems with the annealing of physical systems with many degrees of freedom. To bring a solid to its highly ordered crystalline ground state, first the solid is melted, then the temperature is lowered until crystallization is reached. The temperature must be carefully lowered — on one hand the temperature should be slowly lowered near the crystallization temperature to avoid hardening the solid in a crystalline state with higher energy than the ground state. However, to save time, quicker cooling is acceptable right after the melting of the solid.

If the variables in the optimization problem are interpreted as (the positions of) the particles of the solid, and if the costs of the possible solutions are interpreted as the energies of the physical system, then the states of the physical system might, in this analogy, be taken to be the possible solutions to the optimization problem, the configuration space. To avoid hardening of the solid before the ground state is reached corresponds to avoiding the optimization problem being trapped in a local minimum. The "temperature" in the optimization problem is the control parameter, a parameter that controls when uphill moves are allowed.

The analogy can be summarized as follows:

Physical system (solid)		Optimization problem
annealing	$\longleftrightarrow$	simulated annealing
a state of the solid	$\longleftrightarrow$	a possible solution
particles (their positions)	$\longleftrightarrow$	variables
temperature	$\longleftrightarrow$	control parameter
energy	$\longleftrightarrow$	cost
sample of different states	$\longleftrightarrow$	ensembles

The concept of ensembles will be explained in section 4.4.

With this analogy an optimization algorithm can make use of already existing algorithms simulating the behaviour of a physical condensed matter system. The Metropolis algorithm simulates a statistical system in equilibrium. It starts from one state ω and searches the configuration space by perturbing the present state slightly to get to a new state ω'. If the energy of the new state, $E(\omega')$, is less than the energy of the old state, $E(\omega)$, the new state is always accepted, else the new state is accepted with a probability $\exp(-\Delta E/T)$ where $\Delta E = E(\omega') - E(\omega)$, and T is the temperature of the system (T is normalized such that Boltzmann's constant is unity). The higher the temperature, the greater the probability of accepting a state with a higher energy.

Simulated annealing is a series of iterations of the Metropolis algorithm for decreasing values of the control parameter. The basic elements of the algorithm are:

1. Choose a solution ω in configuration space to start from
2. Choose a start value of the control parameter T
3. Anneal
   (a) Choose a neighbouring solution ω'
   (b) Calculate the difference in cost $\Delta E = E(\omega') - E(\omega)$
   (c) If $\Delta E \leq 0$ then accept the new solution ω', if $\Delta E > 0$ then accept the new solution with the probability $\exp(-\Delta E/T)$
   (d) Reduce the value of the control parameter T
4. Continue annealing until a stop criterion is reached

To actually use the algorithm, some parts of the algorithm must be clarified. For example, how will the algorithm find a "neighbouring" solution, and what exactly do we mean by "neighbouring"? How fast should the control parameter be lowered? And what is the stop criterion? I will use the expression "state" for a solution to the optimization problem, which in our case will be one of the possible multilayer structures.

4.1 Choice of Neighbourhood

The neighbours of a given state is called the neighbourhood. It is not always obvious how to choose the neighourhood, and the choice depends on the nature of the optimization problem. For the same problem, different choices of neighbourhood structure can give algorithms with very different efficiencies. The neighbourhood could be chosen to be the states obtained by changing one of the variables describing the state, by a certain amount.

Let $\mathcal{N}(\omega)$ be the neighbourhood of ω. There are three criterions of the structure of the neighbourhood:

1. All states have the same number of neighbours
2. $\omega' \in \mathcal{N}(\omega) \Leftrightarrow \omega \in \mathcal{N}(\omega')$, i.e. if ω is a neighbour of ω' then ω' is a neighbour of ω
3. Given two states ω and ω' it must be possible to get from ω to ω' in a finite number of steps

In traditional simulated annealing the choice of neighbour for the next attempt is made randomly among the states of the neighbourhood. Together with the three criterions above this ensures that at equilibrium the distribution of states is a Boltzmann distribution; the probability of being at a state ω is

$$p(\omega) = \frac{e^{-E(\omega)/T}}{Z(T)} \tag{8}$$

where $Z(T)$ is the partition function for the whole set of states in the configuration space. With a Boltzmann distribution the probability of being in low energy states increases as the temperature is lowered.

4.2 Choice of Cooling Scheme

A commonly used cooling scheme is the exponential cooling scheme

$$T(t) = T_{max}e^{-t/a} \tag{9}$$

with a a constant, and t "time" passed, i.e. number of iterations executed. The exponential cooling scheme was first suggested by Kirkpatrick *et al.* in [6] and has been widely used since then. It is an *a priori* scheme; the cooling is set from start and does not depend on the performance of the system during the annealing. Special cases of *a priori* cooling schemes are random search, with a constant infinitely large control parameter (all attempted moves will be accepted), and pure downhill search, with control parameter constantly zero.

Logarithmic cooling has also been suggested:

$$T(t) = \frac{d}{\log(t+1)} \tag{10}$$

with d a constant. It is the only scheme for which it has been shown [4] that, given infinite time, the system will certainly reach the ground state, but unfortunately the scheme is much too slow for practical purposes.

Other cooling schemes make use of knowledge of the system while annealing, one such cooling scheme will be explained later (thermodynamic cooling).

In any case a start value of the control parameter (temperature) must be given before the annealing starts. The value should be large enough to allow for an almost random search from the beginning, and small enough to avoid wasting too many iterations. Some cooling schemes are more sensitive to the start value than others. A condition on the choice of start value could be that the rate of accepted moves among the attempted moves should exceed 0.8 in the first iterations of the annealing.

4.3 Other Considerations

From a simulated annealing point of view the state to start from (step 1 in the algorithm) should be chosen randomly, but from the point of view of the application a qualified guess is perhaps better as a starting point. If the start value of the control parameter is large enough it should not matter which state the algorithm starts from.

Also, a stop criterion for the annealing must be given, since infinite time is not available ... in applications to large problems the available CPU time most often sets the limit, but also the cost of the current state, or a modification thereof, can be used to decide when to stop.

4.4 Modifications of the Simulated Annealing Algorithm

Some authors (see for example [9]) define the simulated annealing algorithm to include several Metropolis steps before each cooling (steps 3.(a)–(c) in the algorithm are repeated a number of times). With more steps before cooling, the system has a chance to get closer to its equilibrium at the given temperature; in the analogy to physical systems this should improve the performance of the algorithm. A simple choice is to have a lower limit of accepted moves before each cooling.

Instead of starting the algorithm with only one state (one "walker" in configuration space), a whole ensemble of states can be chosen [12] (an ensemble of "walkers"). Each member of the ensemble "walks" the configuration space independently of the other "walkers". This allows for a statistical description of the system during the annealing, quantities like the mean energy, the variance, and even the heatcapacity of the corresponding physical system can be evaluated. These quantities can be used during the annealing — still in the analogy to the physical system — to suggest how fast to cool. A cooling scheme using the knowledge of the behaviour of the system will be explained below (thermodynamic cooling).

Even with an *a priori* cooling scheme and a fixed CPU time available an algorithm using ensembles should perform better than an algorithm using a single walker [12]. We did not find this to be the case in the problem of finding structures of the x-ray multilayer structures, see section 8.

Finally, a very different choice of neighbourhood has been suggested [10] for problems with a continuous configuration space, i.e. problems where the variables that describe a state are continuous variables. The structure determination of multilayers is a continuous problem: the thicknesses of the layers are continuous variables. The method uses the vertices of a simplex in the N-dimensional configuration space (a simple geometrical figure with $N+1$ vertices) to search the configuration space. The difference from the ensemble approach is that with the simplex method, the moves of the $N+1$ "walkers" are indeed very much correlated, and that the distance a "walker" covers in configuration space in one step depends on the costs of the states of the N other walkers. The simplex approach will be further explained in section 6.

5 Thermodynamic Cooling

Using ensembles during the annealing makes a statistical description of the system possible. At a given temperature each member of the ensemble is in a certain state ω which has an energy (cost) $E(\omega, T)$. Quantities like the mean energy $\langle E(T)\rangle_{\text{ensemble}}$ and the variance of energy $\sigma^2(T)_{\text{ensemble}}$ of the ensemble can be found. If the system is kept at a fixed temperature T it will eventually relax into the equilibrium distribution at that temperature, with mean energy $\langle E(T)\rangle_{\text{eq}}$ and variance $\sigma^2(T)_{\text{eq}}$. If the system is close enough to equilibrium we can use the classical theory of equilibrium thermodynamics on the ensemble. The idea in the thermodynamic cooling scheme is to lower the temperature in such a way that the mean energy of the ensembles is held at a fixed number of standard deviations from the equilibrium energy

$$\langle E(T)\rangle_{\text{ensemble}} - \langle E(T)\rangle_{\text{eq}} = v\sigma(T)_{\text{eq}} \tag{11}$$

where v is a constant (typically in the range 0.01 to 0.5). If the system is close to equilibrium (v is small) then instead of the standard deviation of the equilibrium $\sigma(T)_{\text{eq}}$ we can use the standard deviation of the ensemble $\sigma(T)_{\text{ensemble}}$.

We now want to determine the temperature T such that $\langle E(T)\rangle_{\text{eq}}$ satisfies (11). But since we do not know the properties of the equilibrium distribution of the system, the form of $\langle E(T)\rangle_{\text{eq}}$ as a function of T is unknown. One way to overcome this difficulty is the following. Let the system relax almost to equilibrium at one temperature by letting it run a large number of iterations at a constant temperature T_{max}. Then the mean energy of the ensemble can be used as $\langle E(T_{max})\rangle_{\text{eq}}$, and values of the function $\langle E(T)\rangle_{\text{eq}}$ for lower values of T can be found by extrapolating the function. To this end we need to know the derivative $\frac{dE}{dT}$, where for simplicity I write E for $\langle E(T)\rangle_{\text{eq}}$. This is the heat capacity $C(T)$, which can also be written

$$\frac{dE}{dT} = C(T) = \frac{\sigma^2(T)_{\text{eq}}}{T^2} \tag{12}$$

and again we can replace $\sigma(T)_{\text{eq}}$ by the one of the ensemble. Simple extrapolation gives

$$\frac{dE}{dT} \simeq \frac{\langle E(T_{max})\rangle_{\text{eq}} - \langle E(T)\rangle_{\text{eq}}}{T_{max} - T} \tag{13}$$

The cooling scheme can be summarized as the following steps:

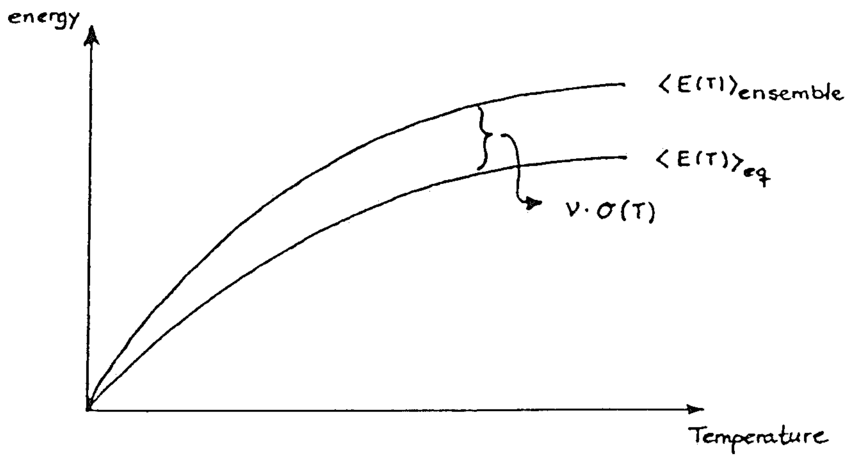


Figure 2: The ensemble mean energy $\langle E(T)\rangle_{\text{ensemble}}$ is kept at a distance $v\sigma(T)$ from the equilibrium mean energy $\langle E(T)\rangle_{\text{eq}}$

1. Run many Metropolis iterations at the temperature T_{max}
2. Evaluate mean energy and standard deviation of the ensemble and set $\langle E(T_{max})\rangle_{\text{eq}} \simeq \langle E(T_{max})\rangle_{\text{ensemble}}$, $\sigma(T_{max})_{\text{eq}} \simeq \sigma(T_{max})_{\text{ensemble}}$
3. Run one or more Metropolis steps
4. Evaluate $\langle E\rangle_{\text{ensemble}}$ and σ_{ensemble}
5. Let $\sigma(T_{new})_{\text{eq}} \simeq \sigma_{\text{ensemble}}$
6. Use equation (11) to find $\langle E(T_{new})\rangle_{\text{eq}}$
7. From T_{max}, $\langle E(T_{max})\rangle_{\text{eq}}$, $\langle E(T_{new})\rangle_{\text{eq}}$ and $\frac{dE}{dT} = \frac{\sigma^2(T_{max})_{\text{eq}}}{T_{max}^2}$ extrapolate value of T_{new} using eq. (13)
8. Let $T_{old} = T_{new}$ and continue from step 3 with T_{old} instead of T_{max}

Another way to find $\langle E(T)\rangle_{\text{eq}}$ as a function of T is described in [7]. There the probabilities of transition between the different energy levels are estimated during the annealing, and from this the equilibrium properties of the system — like $\langle E(T)\rangle_{\text{eq}}$ — are found.

Figure 3: Simplex in two and three dimensional space

6 The Simplex Approach to Simulated Annealing

The simplex approach can be applied to optimization problems with continuous variables, and is described by W.H. Press and S.A. Teukolsky in [10].

In other neighbourhood structures the variables can only be changed with a pre-set amount, and possible minima for in-between values of the variables will never be found. Also, the next attempted neighbour is chosen with equal probability among all neighbours, this is inefficient because, even when a few local downhill moves exist, the uphill moves are almost always attempted.

The simplex approach is based on a pure downhill search algorithm by Nelder and Mead, described in [11]. A simplex is a geometrical figure in N dimensions consisting of $N+1$ vertices, the interconnecting line segments, and the polygonal faces thus formed. In two dimensions the simplex is a triangle, in three dimensions a tetrahedron (figure 3).

When the optimization problem has N variables, then the simplex method uses a simplex in N dimensional space, where the $N+1$ vertices of the simplex are distinct possible solutions to the optimization problem. In our case the vertices are $N+1$ different multilayer structures. The algorithm for the pure downhill simplex method is the following (numbers according to the numbers in figure 4):

1. Evaluate the cost of all vertices, note which vertices have the highest, the second highest and the lowest cost (hereafter called the highest vertex etc.)

2. Reflect the highest vertex in the polygonal face spanned by the other N vertices; this new point x will be tried as a replacement for the currently highest vertex in the simplex.

3. Evaluate the cost of the new point x. There are three cases:

   (a) If the cost of x is higher than the second highest cost, then the direction seems to be wrong, and another direction is chosen; the point y is found at half the distance between the highest point and the polygonal face mentioned above. Evaluate the cost of y.

      i. If the cost of y is lower than the highest cost of the simplex, then y is accepted as a vertex instead of the highest vertex.

      ii. But if the cost of y is higher than the highest cost in the simplex, then the simplex is contracted around the lowest vertex.

   (b) If the cost of x is neither higher than the second highest cost, nor lower than the lowest cost, then x is kept as a vertex of the simplex as a replacement of the highest vertex.

   (c) If the cost of the point x is lower than the lowest cost in the simplex, then the point seems to be in the right direction, and a new point z is attempted at double the distance from the above mentioned polygonal face. Evaluate the cost of z.

      i. If the cost of the point z is not lower than the lowest cost in the simplex, then the highest vertex is replaced by the point x, which had a low cost.

      ii. But if the cost of z is lower than the lowest in the simplex, then z is taken to be a new vertex of the simplex, to replace the highest vertex.

4. Carry out more steps of the algorithm by starting from 1.

This is how the pure downhill simplex method works; when possible the method expands the simplex in a direction with lower costs to take larger downhill steps. The simplex converges to a local minimum. When used in connection with simulated annealing the simplex should be given a chance to escape local minima by sometimes accepting new vertices with higher costs. This is done as follows: to the cost of each vertex in the simplex a random number is added, a positive, logarithmically distributed random variable, proportional to the control parameter. Also, whenever the cost of a new point is evaluated, a similar random number is subtracted from the cost. This way a new point with a cost higher than the costs of the simplex might seem to the algorithm to have a lower cost and thus might be accepted as a new vertex.

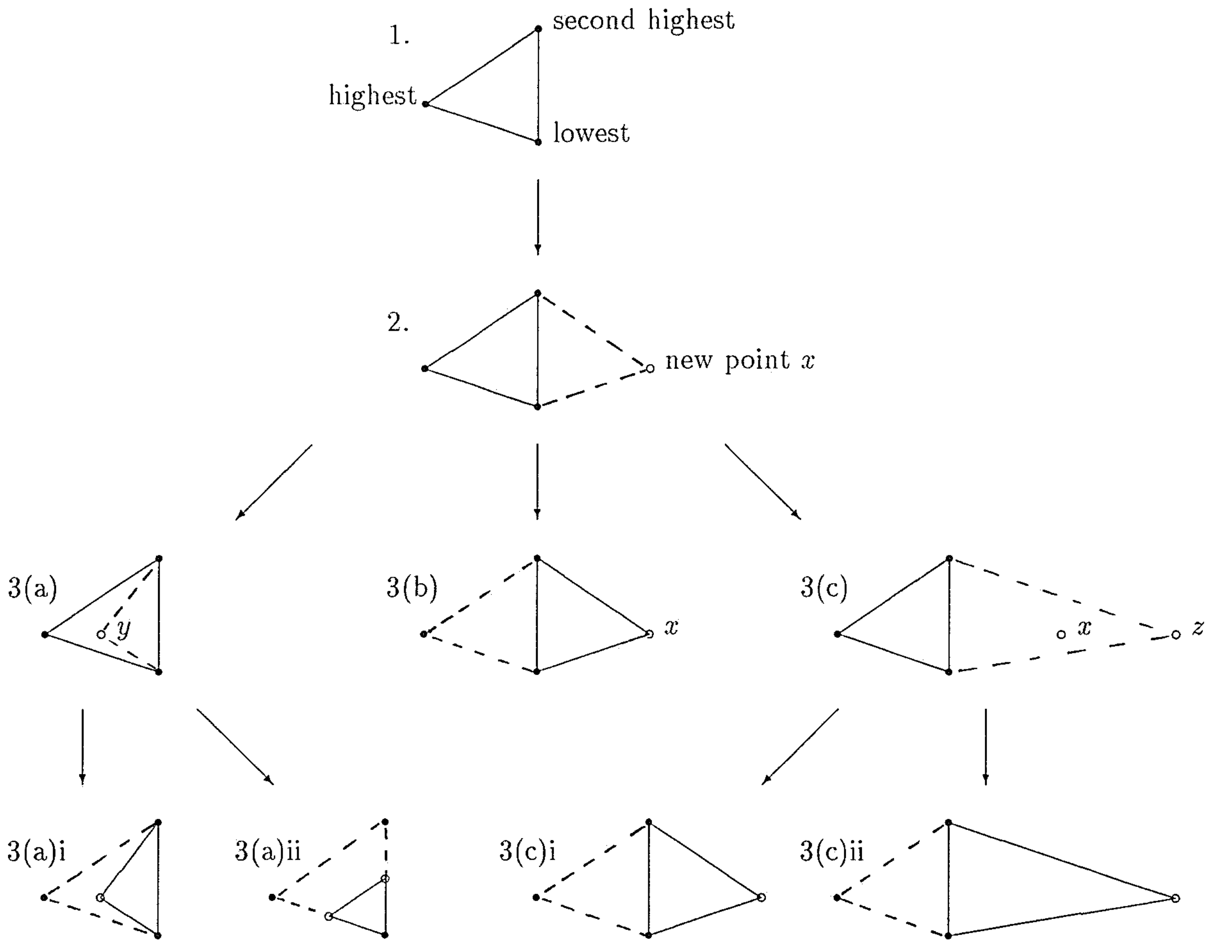


Figure 4: The pure downhill simplex algorithm in two dimensional space. Numbers correspond to the steps in the algorithm, section 6.

A ready-to-use `FORTRAN` subroutine for the simulated annealing simplex method is found in [10] and, with slight modifications, as the two last subroutines in the print out of the `FORTRAN` program `sasi7` in appendix C.

7 Implementation

As will be clear by now, a number of decisions have to be made before implementing the algorithm simulated annealing. In the limited time available for this project I could only try out a few combinations of cooling schemes, neighbourhood structures etc., and even then only relatively short test runs were possible (time scale in the order of days). The aim was to test the simulated annealing algorithm used on reflectivity data, not so much to obtain the best possible results using all possible tricks. I have tried to keep the algorithms relatively general.

For structure determination of multilayers the inputs to the algorithm are the number of layers, the substance and/or the density of the layers and of the substrate on which the layers are grown, a qualified guess of the thickness d_i of each single layer and of the roughness σ_i of their surfaces. The thickness and roughness are taken to be the variables of the simulated annealing, but other variables could be added, as will be explained later. A maximum and minimum value of the variables (thickness and roughness) should be given to avoid wasting time evaluating structures that are not realistic — structures that can not be produced due to limitations in the actual production phase (for the design problem) or, for structure determination, structures with values that are beyond the range of values suggested by other means (e.g. if a bilayer is found to be of thickness 35Å $\pm$ 3Å by experimental methods there is no need to test a structure with thicknesses 30Å and 19Å for the two layers).

The theoretical reflectivity curve of a given multilayer is calculated using the equations of section 3, with the Debye-Waller roughness factor. The cost function is a measure of how well the theoretical reflectivity curve fits the datapoints of the experimental curve. We chose to use the cost function

$$\text{cost} = \frac{1}{n}\sum_{1}^{n}(\frac{R_{calc} - R_{exp}}{R_{calc}})^2 + (\frac{R_{exp} - R_{calc}}{R_{exp}})^2 \qquad (14)$$

where n is the number of datapoints.

Name of program	sasi7	sasi8	sasi9	sasi10
Cooling scheme	exponential	exponential	exponential	thermodyn.
Ensembles?	simplex	single walker	ensemble	ensemble

Four `FORTRAN` programs where written, all with the same basic structure, but with different choices for neighbourhood or cooling scheme (see table). The choice of neighbourhood structure is important, because the energy (cost) landscape mainly depends on this. A bad choice might give steep barriers that the walker can only pass at high temperatures, whereas a different choice might give a smoother landscape easy to move around in even at low temperatures. In programs `sasi8`, `sasi9` and `sasi10` a neighbour is found by changing the thickness or the roughness of one (randomly chosen) layer by a pre-set amount ($\pm\Delta$ thickness or roughness). To keep the number of neighbours the same for all states — a criterion for the neighbourhood structure — the variables are made cyclic; if a move is attempted past the upper limit of one of the variables then the lower limit is substituted before it is decided if the move will be accepted. The change in thickness is usually set to 0.5Å, and such a change can give rise to a cost difference from almost none to the order of $10,000$, which is a steep barrier compared to the best cost found (often of order 1 or 0.1). It seems that such a choice of neighbourhood structure is not very good, but except for slight modifications of it, or the totally different simplex method, no other choices have been suggested as far as I know. That is, not if all the thickness and roughness variables should be allowed to vary independently of each other. A parameter, determining the likelihood of changing the roughnesses (as opposed to the thicknesses) when choosing a neighbour, was implemented. If the value of the parameter is set to 1 only the thickness variables can be changed, and if the parameter is set to 0.5 the thickness and roughness variables are changed with equal probability.

In program `sasi7` the simplex method has been used, as described in section 6. Only variables that have a range to vary on (i.e. the maximum and minimum values are not equal) add to the number of dimensions of the space in which the simplex moves.

The cooling schemes used where exponential cooling and thermodynamic cooling. The exponential cooling scheme is easy to implement, and is used in `sasi7`, `sasi8` and `sasi9`. The scheme is sensitive to the start value of the control parameter, and should be chosen such that the accept rate is 0.8 in the beginning. A minimum value of the control parameter is also needed. This should in principle be close to zero, but to avoid spending too much time at the final minimum (whether it is the global minimum or a local minimum) the value should be set such that the accept rate at the end of the annealing is small but not vanishing.

The thermodynamic cooling scheme was implemented in `sasi10` using the algorithm described in section 5.

Multilayer structures with a realistic ~ 100 layers will give ~ 200 variables to optimize (when both thickness and roughness is to be optimized), this is a large number of variables, even for the simulated annealing algorithm, and an optimization will take a long time on present day computers. A way to cut down the number of variables, and thereby the computation time, is to assume a periodic structure of the layers. This was not done in any of the four `FORTRAN` programs, but would be interesting to test.

It is also possible to assume that the thickness of the layers depend on the position of the layer in the stack, e.g. one could assume that the layers near the top are thicker than the layers near the substrate, the layers that only the higher energies of the beam will reach. A functional form of this dependence with some unknown parameters must be given to the simulated annealing algorithm, which then optimizes the parameters; the number of variables of the simulated annealing is the number of unknown parameters of the function.

8 Results

The aim of the project was to test simulated annealing in structure determination and design of multilayer structures, not to test the methods of calculating the reflectivity. Therefore, for most of the structure determination testruns, the "experimental" reflectivity curve is actually a curve calculated from a known structure (unknown to the program, of course), thereby eliminating possible effects arising from the reflectivity calculations.

Two design problems were tested. They both consist in maximizing the reflectivity of a Platinum/Carbon multilayer structure for energies of the beam in the range 5.7keV to 7.0keV. The number of periods is given to be 12 (25 layers including the substrate), and the incident angle of the beam is $\theta = 1.53^o$ for the first problem and $\theta = 1.00^o$ for the second problem. For design no roughness of the interfaces of the layers is assumed.

The $\theta = 1.00^o$ problem was run with a single walker (`sasi8`), with a simplex (`sasi7`) and with thermodynamic cooling (`sasi10`). The three programs found different optimal structures, but all with almost the same reflectivity in the energy range considered. The best costs found (using equation (14)) by the three programs were 0.849, 0.906 and 0.941 respectively; the reflectivity curve found by `sasi8` is in appendix B. The numbers on the y-axis are the $\log_{10}$ values of the reflectivity, and in the energy range considered the reflectivity of the found structure is between $0.44 (= 10^{-0.36})$ and $0.66 (= 10^{-0.18})$. These results are better than what a manual fit gives, and they are certainly less tiresome to obtain.

Most of the design testruns were run with $500,000$ iterations, which on the machines used took one night. For the programs using exponential cooling a maximum and a minimum value of the control parameter should be given. The testruns where the maximum value was set to 10^{-1} had an acceptance rate around 0.8 to start with. The minimum value of the control parameter was set to 10^{-4}, and the acceptance rate at the end of the testruns was around 0.3. At the testrun with the simplex program the simplex "collapsed" after few iterations, i.e. the vertices in the simplex all got the same values of the coordinates except for one coordinate. For the multilayer structures this means that all the supposedly different structures making up the vertices of the simplex after a few iterations all got the same thicknesses except for *one* layer, in this case layer number five. This problem of a collapsing simplex is not specific for the design problem, also in structure determination the simplex tends to collaps. The problem will be discussed below.

For the program using thermodynamic cooling the maximum value of the control parameter should not matter too much, as long as the value is not set too low. In the light of the acceptance rates found for a single walker, using 10^{-1} as the maximum value, it was decided to use the values 2 and 20 for two testruns of the thermodynamic cooling program. In the first testrun the constant v (see section 5 for explanation) was set to 0.05, but after almost $500,000$ iterations the algorithm still seemed able to find better structures given more time. $v = 0.05$ was perhaps too strict a condition, and $v = 0.25$ was tried in the next testrun. This, on the other hand, seemed to be too weak a condition: the temperature does not fall off monotonously. Therefore, at later testruns (for structure determination), the constant v is kept at the value 0.10.

For the incident angle $\theta = 1.53^o$ a best cost of 12 was found, the reflectivity is approximately 0.25. The lower values of the reflectivity found at $\theta = 1.53^o$ do not surprise, since the reflectivity of solids at larger angles is small.

The structure determination was tested on three different Tungsten/

Silicon structures: a periodic structure with 11 layers (including the substrate), an a-periodic structure with 11 layers, and a periodic structure with 51 layers. All three "experimental" reflectivity curves, from which the programs should find the multilayer structures, are given by 301 datapoints in the range of angles $\theta = 0^o$ to 3^o. The energy of the beam is 8keV. The "experimental" reflectivity curves to be used at the tests were found from theoretical multilayer structures without roughness of the interfaces, and thus in the testruns roughness was not considered. Adding roughness to the variables in the test is computationally approximately the same as doubling the number of layers.

The 11 layers periodic structure was tested with the programs `sasi8`, `sasi7` and `sasi10`. The single walker, `sasi8`, finds the original structure after $50,000$ iterations in one of the testruns, but gets stuck at a very high cost after only 34 iterations (!) in another testrun. Also, in the first testrun, the original structure is *almost* found after approx. $20,000$ iterations, with only the thickness of one layers off by 0.5Å, and the algorithm wanders off in a worse direction from there. This suggests that there is a great deal of randomness involved in finding the right structure using the single walker, and the idea in using ensembles or simplexes is to be more certain to get to the best structure, though no guarantee is given.

With the time available for tests (approx. $250,000$ iterations per test) this did not seem to be the case. Neither the ensemble program with exponential cooling (`sasi9`), the ensemble program with thermodynamic cooling (`sasi10`), nor the simplex program (`sasi7`) find the original structure of the 11 layers periodic structure. `sasi10` finds a structure with reflectivity curve and cost (2.17) not far from reflectivity of the original, whereas a random search after $328,000$ iterations still has not found anything reasonable, the best cost is 580. A pure downhill search was conducted twice; the program got stuck at structures with costs 26 and 400 after 400 iterations. No matter what the hopes for the simulated annealing programs were, at least the programs prove to be — by far — better than random searches or pure downhill searches!

One test was run on the 51 layers periodic structure. The original structure was not found, and it became obvious that to get any reasonable test results the program would have to run for a longer time than was available. Whenever the number of layers is increased by a factor n, the time needed for the reflectivity calculations is increased by at least a factor n, and the number of iterations necessary to change each layer a given number of times (in average) is also increased by a factor n, so that the necessary computational time is increased by at least a factor n^2. A typical test of the 11 layers structure took one night, and a test of the 51 layers structure would then take a week or two.

Instead the tests were concentrated on the 11 layers a-periodic structure. None of the testruns found the original structure. The best fit to the reflectivity curve of the original structure was found by the single walker. The cost is low (0.124), and the two reflectivity curves look very similar. The curves are shown in the second diagram in appendix B, the dark grey curve is from the original structure, the light grey curve is the best fit. Although the two reflectivity curves look almost the same, the multilayer structures are different:

	Original structure (thickness in Å)	Structure for best fit (thickness in Å)
substrate	10^8	10^8
layer 1	17.5	17.0
layer 2	43.5	44.5
layer 3	19.0	18.0
layer 4	45.5	46.5
layer 5	15.0	14.5
layer 6	45.5	45.0
layer 7	17.0	18.0
layer 8	44.5	43.0
layer 9	17.0	18.0
toplayer	45.5	48.0

This is a problem not only for the simulated annealing algorithm. That more multilayer structures give almost the same reflectivity curves makes it difficult to determine the structure from which an experimental reflectivity curve originates, no matter which optimization algorithm is used. For the design of multilayers this is not a problem, all we are interested in is to find *one* structure that gives the desired reflectivity.

The testruns using thermodynamic cooling give best fits with costs 0.617 and 0.738. The reflectivity curve for the cost 0.738 is shown in the third diagram in appendix B. Again the dark grey curve is the original, and the light grey curve is the best fit.

In most of the testruns the program using thermodynamic cooling gives results not quite as good as the single walker. The temperature is lowered according to the behaviour of the members of the ensemble. In all tests an ensemble of 50 walkers was used, this might not be enough to find reliable statistical quantities. In some of the testruns the temperature falls off very slowly and it even *rises* once in a while. In program `sasi10` the temperature is lowered after each Metropolis step of the ensemble. As mentioned before more steps might be necessary to let the system get closer to its equilibrium distribution before each lowering of the temperature. One test was run on a modification of `sasi10` with 10 Metropolis steps of the ensemble before each cooling step. Still the temperature rose once in a while, but not quite as often. The best fit was not better than obtained by one Metropolis step per cooling step. It would be interesting to test a thermodynamic cooling with more Metropolis steps.

In general the simplex program did not perform very well. In almost all testruns the simplex collapsed, with only one or two variables not stuck. Once, in a very long testrun, the simplex collapsed and stayed collapsed for many iterations but then very slowly began to move, though not enough to change the best fit. The cause might be rounding errors after many iterations, a change to higher precision variables in the `FORTRAN` program will show if this is the case. But higher precision variables (probably) still will not prevent the simplex from collapsing. One solution will be to set a minimum difference of the variables of the simplex, e.g. for each layer there should be a minimum difference between the thickness of the thickest and the thinest of the structures in the simplex. The minimum difference should then decrease with the temperature. I am not sure that this will prevent the simplex from collapsing, but it is the only solution I can imagine.

The testruns using simplexes did not perform very well because the simplexes collapsed, but even if they had not collapsed, they would probably have needed more CPU than the other programs to find good fits. This is because the simplex searches a *continuous* space, whereas the other programs only search for (in our tests) half integer values of the thicknesses. The original structures have half integer values, and thus the tests favourize the other programs. In a test with real experimental data this would not be the case, of course.

9 The `FORTRAN` Programs

For this project five `FORTRAN` programs were written, simulated annealing in `sasi7-10`, and `graphics`, a small program to show diagrams after the annealing. The programs `sasi7`, `sasi8` and `sasi10` are in appendix C.

Name of program	sasi7	sasi8	sasi9	sasi10
Cooling scheme	exponential	exponential	exponential	thermodyn.
Ensembles?	simplex	single walker	ensemble	ensemble

In this section I will use the term "S.A. variable" about a variable in the simulated annealing (thickness, roughness), and "`FORTRAN` variable" or simply "variable" about a variable in the `FORTRAN` programs.

A `FORTRAN` program `mlp`, written by P. Høghøj, calculates the reflectivity curve for a given multilayer using the method described in section 3. `mlp` was taken as a starting point for the implementation of simulated annealing. `sasi8` is the simplest of the four simulated annealing programs, and it was also the first program to be written. The main structure of `sasi8` is:

1. read input (everything except experimental reflectivity curve) (`rdfile`)
2. anneal: (`anneal`)
    - (a) read experimental curve (`rddat`)
    - (b) prepare optical constants (`optconan` or `optconen`)
    - (c) for a number of steps do:

      choose a neighbour (a multilayer structure) (`neighbour`)
      calculate reflectivity for neighbour (`reflecan` or `reflecen`)
      calculate the cost of the neighbour (`cost1`)
      decide if neighbour should be accepted (`accept`)
      move to neighbour if it was accepted
      lower the control parameter (`cool`)
    - (d) write output (`wrtfile`)

The S.A. variables are kept in a one-dimensional array `thick`. The variables used for the optical calculations are all kept by pointers, that way the size of the variables does not need to be declared before the program is compiled and run, and the size can be decided after reading the input file (variables `delta`, `beta`, `theta`, `energy`, `cd_index`). The reflectivity data are also kept by pointers, the variables are `calcrefl`, `exprefl`, `bestrefl`.

More S.A. variables can be added by changing the size of the array `thick` and the arrays connected to it (`bestthick`, `maxthick` etc.).

During the annealing the program will show a diagram with the experimental reflectivity curve, a curve for the best structure found so far, and a curve for the structure the program attempts now, all versus the angle or energy. The diagram is updated according to the value read from the input data. The graphs are drawn using a set of subroutines `EXPG` written by A. Hammersley, ESRF.

The subroutine `cost1` calculates the cost of a given reflectivity curve and uses the cost function (14). Two other cost function subroutines are included (`cost2` and `cost4`), they are not used by the program, but can be used if `cost1` is changed to `cost2` or `cost4` everywhere except at the subroutine itself.

`sasi9` is similar to `sasi8` except that ensembles have been implemented. `thick` is now a two dimensional array where each row represents a member of the ensemble, i.e. one row of `thick` in `sasi9` is the same as the whole one dimensional array `thick` in `sasi8`. The steps 2.(c) in the algorithm are carried out for each member of the ensemble.

The thermodynamic cooling scheme is implemented in `sasi10`. A number of Metropolis steps are carried out at the maximum value of the control parameter before the annealing takes place. This is done to let the ensemble approach its equilibrium distribution, as explained in section 5. The maximum number of steps is set to 1000 per member of the ensemble, but the ensemble is considered close enough to equilibrium if in 10 steps the change in mean energy of the ensemble is less than its standard deviation times the constant v divided by 10:

$$\Delta\langle E(T)\rangle_{\text{ensemble}} < v\sigma(T)_{\text{eq}}/10 \tag{15}$$

In practice the mean energy remains reasonably stable at the last iterations before (15) stops the search for the equilibrium distribution.

The structure of `sasi10` is similar to `sasi8` except for the change to ensembles and for the extra steps needed to get close to the equilibrium before annealing:

```
1. read input (rdfile)
2. anneal: (anneal)
   (a) read experimental curve (rddat)
   (b) prepare optical constants (optconan or optconen)
   (c) find equilibrium properties at max. control parameter
   (d) for a number of steps do:
           for each member of the ensemble do: (step_ensemble)
               choose a neighbour (neighbour)
               calculate reflectivity for neighbour (reflecan / reflecen)
               calculate the cost of the neighbour (cost1)
               decide if neighbour should be accepted (accept)
               move to neighbour if it was accepted
           lower the control parameter (cool)
   (e) write output (wrtfile)
```

In `sasi10` the control parameter is changed when all the members of the ensemble have gone through one iteration. The program can easily be changed to take more iterations per member before next change of control parameter. The subroutine `cool` uses the equations and the algorithm of section 5 to lower the control parameter.

`sasi7` uses the simplex method to choose neighbours. Of the S.A. variables describing the multilayer structure only actual variables are used in the simplex, as opposed to variables that are fixed by the input data, e.g. with Δthickness zero, or with the maximum value equal to the minimum value. Let N be the number of such actual variables. The data is first read to the `FORTRAN` variable `thick`, including any fixed variables. The N actual variables are then transfered to the first row of an $(N+1) \times N$-sized array `sim_p` as the first vertex of the simplex, and the array `thickconst` keeps track of the correspondance between the multilayer structure and the dimensions of the simplex space. The $N+1$ rows of `sim_p` are the $N+1$ vertices of of the simplex, and the start values of the vertices are found by adding a multiplum of Δthickness or Δroughness to either the thickness or the roughness of one of the layers, provided it is not a fixed variable.

The subroutines for the simplex movements are kept as close as possible to the original subroutines suggested in [10]. To do this, the calculation of the reflectivity was moved to a subroutine of the cost evaluation. It does not alter the way the program calculates the cost, but does make the program structure look a bit different:

1. read input (`rdfile`)
2. get rid of fixed variables
3. anneal: (`anneal`)
   (a) read experimental curve (`rddat`)
   (b) prepare optical constants (`optconan` or `optconen`)
   (c) prepare a start simplex
   (d) for a number of steps do:
       call the simplex subroutines (`amebsa`)
       lower the control parameter (`cool`)
   (e) write output (`wrtfile`)

10 Format of In- and Outdata

An example of an input file is shown in appendix A. It is essential to keep the exact format of the input file. The `Input data filename` is the name of the file containing the datapoints of the experimental reflectivity curve. It has four text lines of no importance to the program, and two columns of numbers: the first column contains the x-axis values (energy in eV or angle in degrees), the second column contains the corresponding values of the experimental reflectivity.

The `Output data filename` is the name of the file to where the best structure found will be written at the end of the annealing, along with the calculated and the experimental reflectivity, and the `Name of .log file` is the file where the results during the annealing will be written (i.e. the number of iterations, the control parameter, the best cost so far, the rate of accepted moves ...).

The `Number of loops before output to screen` at the end of the input file is the number of iterations run before the graph on the screen is updated and the values are written in the `.log` file and on the screen. When using ensembles the "loop" is one step for each member of the ensemble, and thus the value should be set accordingly lower than for the program that uses only a single walker. Likewise the simplex program counts one "loop" as one call of the `amebsa` subroutine.

After termination of the simulated annealing program the `.log` file can be inspected using the program `graphs`, a simple program that gives choices for the x- and y-axis and draws a graph. A `.cgm` file with the graphs can be made for later print out (use `ralcgm`). `graphs` also uses the graphic subroutines `EXPG`.

References

[1] B. Andresen *Finite-time Thermodynamics and Simulated Annealing* Fysisk Laboratorium, University of Copenhagen **Report no. 89-12** (1989)

[2] T.W. Barbee, Jr. *Opt. Eng.* **25**, 898 (1986)

[3] W.J. Bartels, J. Hornstra, D.J.W. Lobeek *Acta Cryst.* **A42**, 539 (1986)

[4] S. Geman, D. Geman *IEEE Trans., PAMI* **6**, 721 (1984)

[5] P. Høghøj, E. Ziegler, C. Riekel *d-Spacing Design of Double Multilayer Monochromators in the X-Ray Region* ESRF, Grenoble (1991)

[6] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi *Science* **220**, 671 (1983)

[7] J.D. Nulton, P. Salamon *Phys. Rev.* **A 37**, 1351 (1988)

[8] J.M. Pedersen *Simuleret Udglødning, En global optimeringmetode* Fysisk Laboratorium, University of Copenhagen (1989) (in Danish)

[9] J. Pannetier *Phys. Conf. Ser.* **107** 23 (1990)

[10] W.H. Press, S.A. Teukolsky *Computers in Physics* **July/August**, 426 (1991)

[11] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling *Numerical Recipes: The Art of Scientific Computing* Cambridge University Press, New York (1986)

[12] G. Ruppeiner, J.M. Pedersen, P. Salamon *Ensemble Approach to Simulated Annealing* Physics Lab., Univ. of Copenhagen **Report no. 89-17** (1989)

[13] E. Spiller *Revue Phys. Appl.* **23**, 1687 (1988)

Appendix A

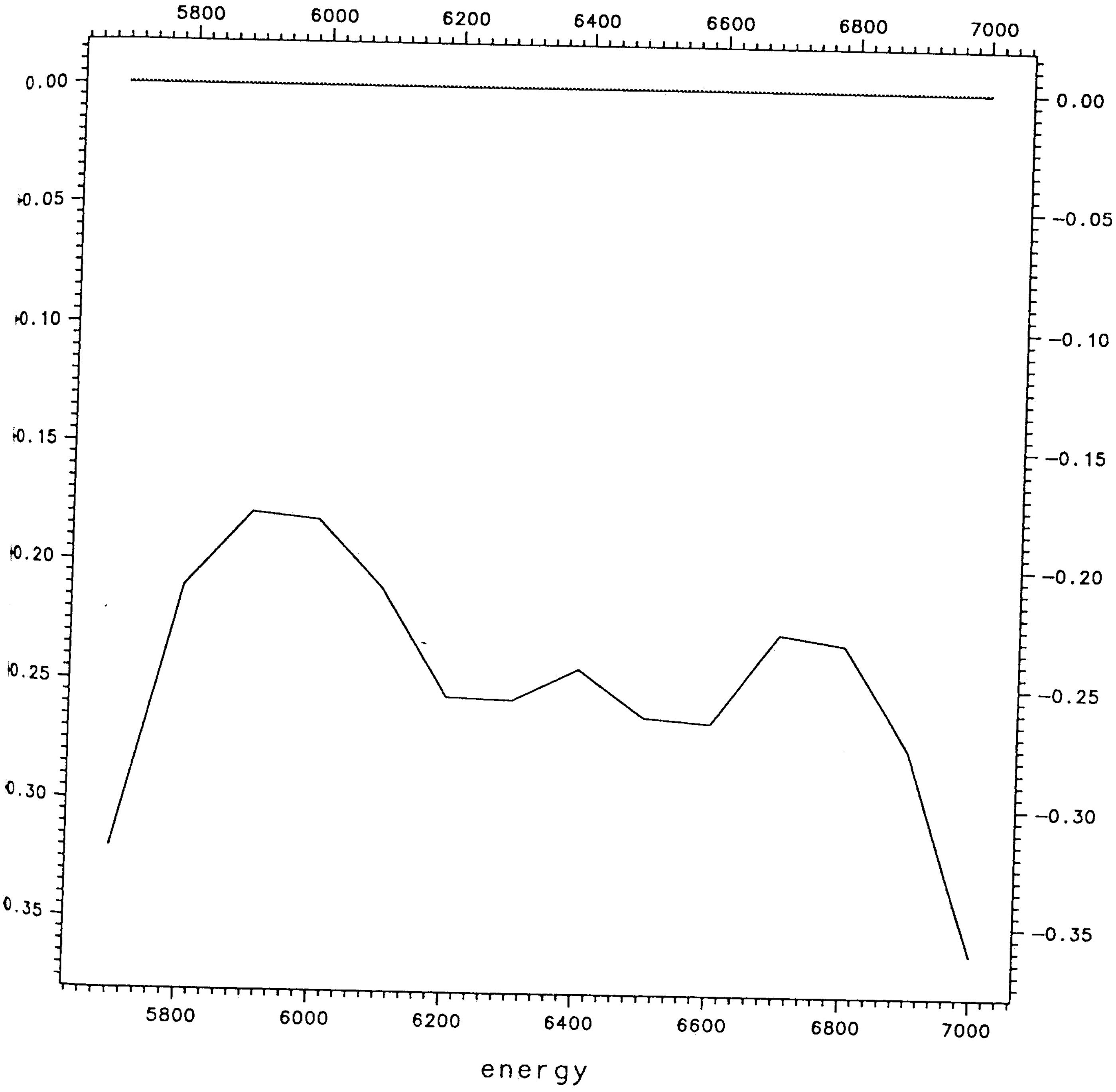
Input File

```
This is an inputfile for the multilayer simulation program mlp ver.1.00.
Please edit the file to describe your multilayer and simulation, while
preserving the exact format of the file. Have fun.     Nov-91  PH   sasi
************************************************************************
Input data filename: testdata/s0102.indat
Output data filename: testdata/s0102b.outdat
Name of .log file: testdata/s0102b.log
The simulation:
Do you want to calculate the reflectivity as a function of angle or
energy [a/e]? a
********
Initial value of the control parameter=100.
Final value of the control parameter=0.05
Number of iterations=250000
Number of datapoints=301
If angle:
Energy[ev]=8000.
Minimum angle[degrees]=0.
Maximum angle[degrees]=3.0
Angle increment[degrees]=0.01
Do you wish to include the angular dependence of the optical constants?
(or just use the value for the mean angle)[y/n]? y

If energy:
Angle[degrees]=0.42
Minimum energy[eV]=6000
Maximum energy[eV]=8000
Energy increment[eV]=100

The multilayer[optional name and description]:
"This is a test of the Simulated Annealing Multilayer program sasi,
development version. "

Number of layers (incl. substrate)=11
Do you wish to specify all layers or only the substrate and the next two
layers (mlp will assume the rest of the multilayer to be a sequence of
these two layers)[all/2]? all

Formula        rho     rough   min     max     delta   thick   min     max     delta
              [g/ccm]  [A]     [A]     [A]     [A]     [A]     [A]     [A]     [A]
Si             0.0     0.0     0.0     0.0     0.0     1.e8    1.e8    1.e8    0.0
W              0.0     0.0     0.0     0.0     0.0     20.0    10.0    30.0    0.5
Si             0.0     0.0     0.0     0.0     0.0     40.0    30.0    50.0    0.5
W              0.0     0.0     0.0     0.0     0.0     20.0    10.0    30.0    0.5
Si             0.0     0.0     0.0     0.0     0.0     40.0    30.0    50.0    0.5
W              0.0     0.0     0.0     0.0     0.0     20.0    10.0    30.0    0.5
Si             0.0     0.0     0.0     0.0     0.0     40.0    30.0    50.0    0.5
W              0.0     0.0     0.0     0.0     0.0     20.0    10.0    30.0    0.5
Si             0.0     0.0     0.0     0.0     0.0     40.0    30.0    50.0    0.5
W              0.0     0.0     0.0     0.0     0.0     20.0    10.0    30.0    0.5
Si             0.0     0.0     0.0     0.0     0.0     40.0    30.0    50.0    0.5

Parameters:
-----------
Number of loops before output to screen=40
If ensemble program: ensemble size=50
If thermodyn. cooling: v=0.10
If simplex program: ftol=1e-5
Number of itrs between change of controlparameter=100



!23456789*123456789*123456789*123456789*123456789*123456789*123456789*123456789*1234
```

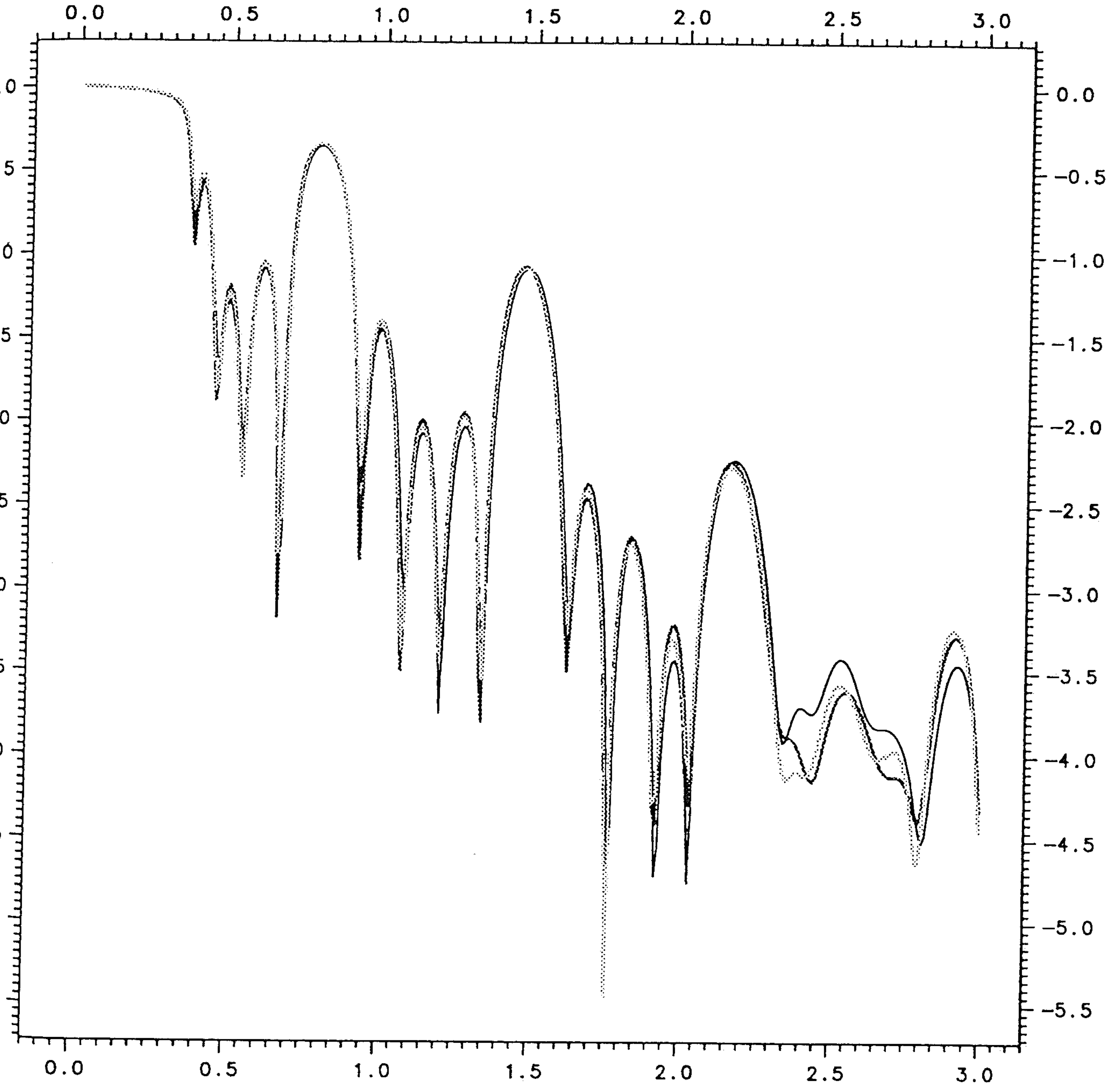
Appendix B

Diagrams

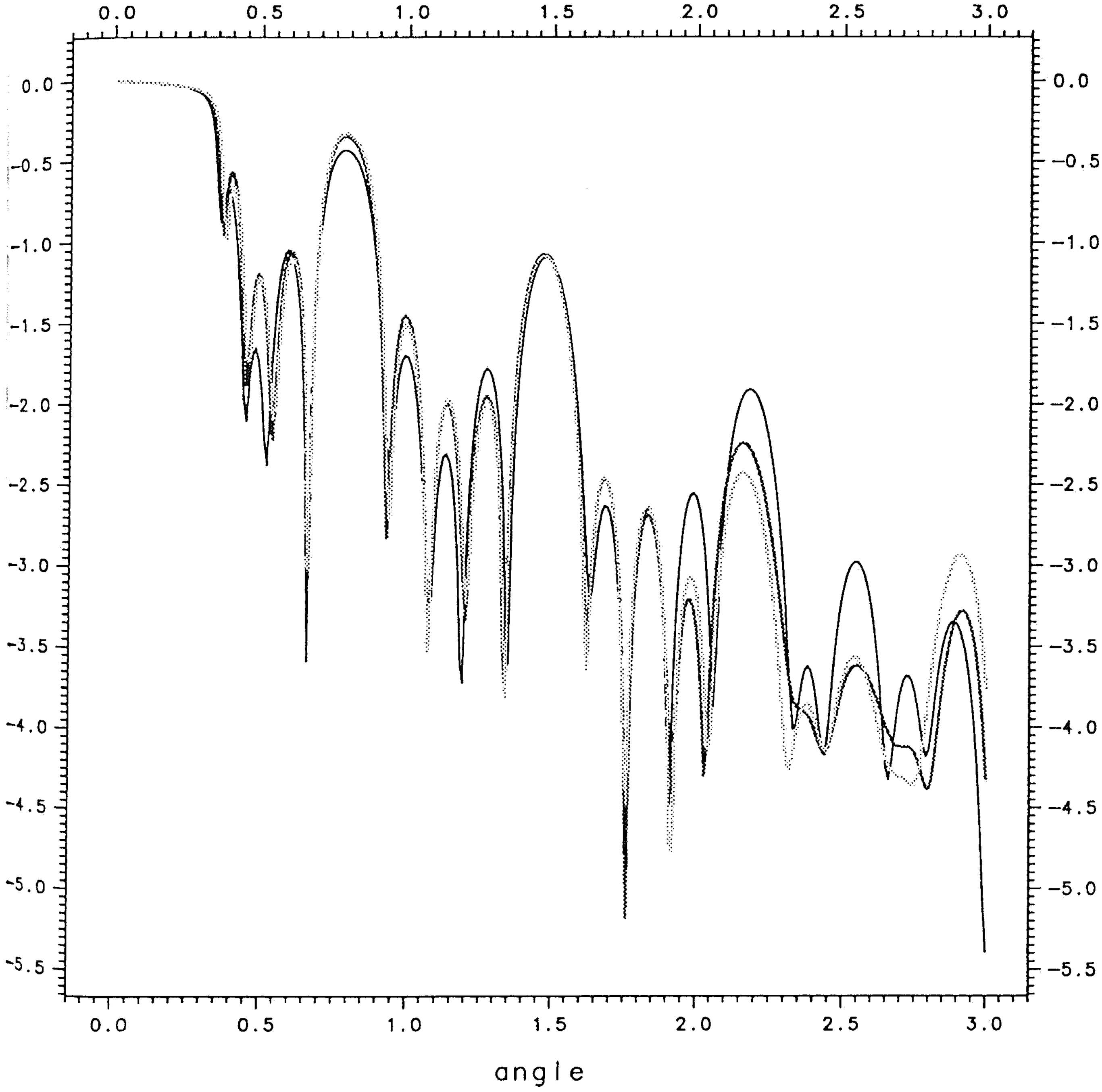
test graph, sasi8



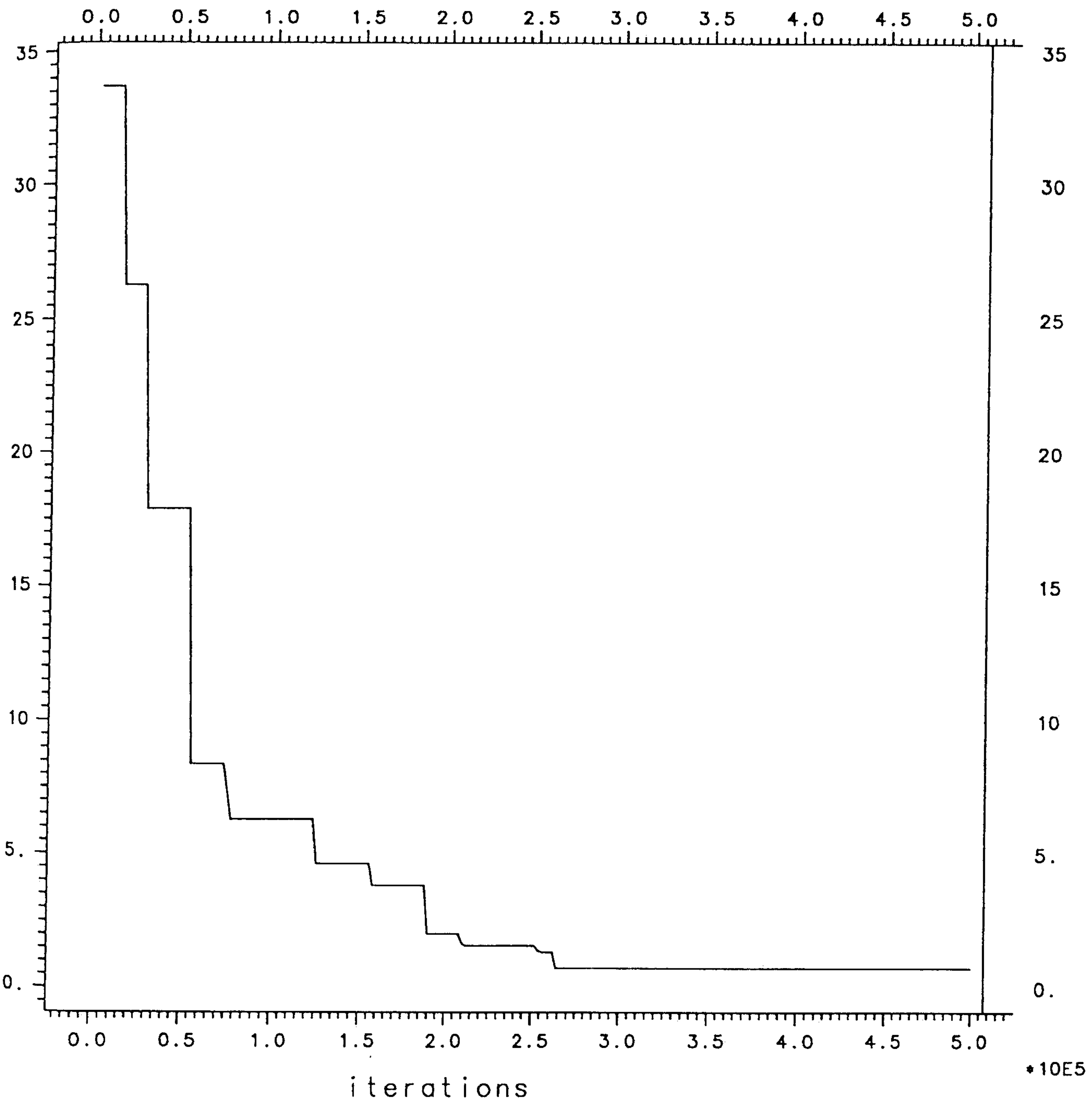
single, expon. cooling



ensemble, thermodyn. cooling



testdata/s0202g.log



Appendix C

FORTRAN Programs

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        program sasi7
C       =============
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C This is the Simulated Annealing Multilayer program mlp, development version.
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C The program performs simulated annealing on a multilayer structure in order
C to optimize the fit to a reflectivity curve. A number of subroutines are used:
C
C anneal        Performs the annealing using the subroutines
C rdfile        Reads the input file
C rddata        Reads the data file
C optconan      Finds optical constants
C reflecan      Calculates the reflectivity of a multilayer structure
C cost          Calculates the cost function of a reflectivity curve
C cool          Lowers the temperature exponentially
C wrtfile       Writes the result to a file
C amebsa        Simplex subroutine
C amotsa        Chooses new simplex point
C
C The optical method is based on an article by:
C   W.J. Bartels, J. Hornstra and D.J.W. Lobeek, Acta. Cryst. A42, 539 (1986)
C
C The simulated annealing is based on a report by:
C   E. Schroeder, S. Friis-Jensen, Physics Laboratory, U. of Copenhagen (1989)
C
C Elsebeth Schroder, Peter H0gh0j, ESRF, June-92
C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

        integer*4 nlayers,npts,size1,size2,byt1,byt2,nsteps,malloc,n,i,
     1            size3,sim_index,ndim,thickconst(3001*2),aoe,consskip,
     1            itrtemp
        character*80 inname,indat,outdat,sym(3001),outlol
        character*20 anoren,angdep,allortwo
        real*4 rho(3001),ftol,
     1  thetmin,thetmax,dthet,engmin,thetamean,
     1  engmax,deng,eng,the,mincontrol,maxcontrol,minthick(3001*2),
     1  maxthick(3001*2),bestthick(3001*2),deltathick(3001*2),
     1  cost(3001*2+1),sim_pbest(3001*2),sim_pmin(3001*2),
     1  sim_pmax(3001*2),sim_pdelta(3001*2)
        real*4 delta,beta,theta,bestnowrefl,exprefl,bestrefl,sim_p,
     1         energy
        complex*16 cd_index
        pointer (c,cd_index) ,(d,delta),(b,beta),(t,theta),
     1          (r1,bestnowrefl),(r2,exprefl),(r3,bestrefl)
        pointer  (p,sim_p),(e,energy)
        parameter (byt1=4,byt2=16)
        data n /3/
        integer status
        integer Expg_st_good
        external Expg_st_good
        Implicit None
        real x_coordinates(3001,5),y_coordinates(3001,5)
        common/points/npts,nlayers,ndim,aoe

C       Executable code:

        write(*,'(''Enter input file name: '')')
        read(*,'(a50)') inname

        call rdfile(inname,indat,outdat,outlol,anoren,mincontrol,
     1  maxcontrol,nsteps,eng,thetmin,thetmax,dthet,angdep,the,
     1  engmin,engmax,deng,nlayers,allortwo,sym,rho,
     1  bestthick,minthick,maxthick,deltathick,npts,consskip,
     1  ftol,itrtemp)

        type *,'maxcontrol',maxcontrol
        type *,'mincontrol',mincontrol

C       Prepare variables for simplex method
        sim_index=0
        do i=1,nlayers*2
         if ((deltathick(i).eq.0.).or.(minthick(i).ge.maxthick(i))) then
              thickconst(i)=1
            else
              sim_index=sim_index+1
              thickconst(i)=0
              sim_pbest(sim_index)=bestthick(i)
              sim_pmin(sim_index)=minthick(i)
              sim_pmax(sim_index)=maxthick(i)
              sim_pdelta(sim_index)=deltathick(i)
           end if
        end do
        ndim=sim_index

C       Graphics
        status=Expg_st_good()
        call EXPG_GR_OPEN_GRAPHICS(status)
        call EXPG_GR_SET_CURVELINE(2,1,7,2.0,.false.,0,status)
        call EXPG_GR_SET_CURVELINE(3,1,5,2.0,.false.,0,status)
        call EXPG_GR_SET_CURVESTYLE(1,.true.,.false.,.false.,status)
        call EXPG_GR_SET_CURVESTYLE(2,.true.,.false.,.false.,status)
        call EXPG_GR_SET_CURVESTYLE(3,.true.,.false.,.false.,status)

        call blank(anoren,n)
        if (anoren .eq. 'a  ') then
           aoe= 1
           size1=npts*byt1
           size2=npts*(nlayers+1)*byt2
           size3=byt1*ndim*(ndim+1)
           b=malloc(size1)
           d=malloc(size1)
           c=malloc(size2)
           e=malloc(byt1)
           r1=malloc(size1)
           r2=malloc(size1)
           r3=malloc(size1)
           t=malloc(size1)
           p=malloc(size3)
           energy=eng
           thetamean=(thetmax+thetmin)/2.
        else if (anoren .eq. 'e  ') then
           aoe= -1
!           npts = int((engmax-engmin)/deng+1.+1.0e-15)
           size1=npts*byt1
           size2=npts*(nlayers+1)*byt2
           size3=byt1*ndim*(ndim+1)
           b=malloc(size1)
           d=malloc(size1)
           c=malloc(size2)
           e=malloc(size1)
           r1=malloc(size1)
           r2=malloc(size1)
           r3=malloc(size1)
           t=malloc(byt1)
           p=malloc(size3)
           theta=the
        else
           type *,'Error in file',inname
           type *,'- neither angle (a) nor energy (e) to vary'
           goto 3
        end if
        call anneal(inname,indat,outdat,outlol,mincontrol,maxcontrol,
     1              nsteps,energy,angdep,sym,rho,thetamean,
     1              sim_p,sim_pmin,sim_pmax,sim_pbest,
     1              sim_pdelta,cd_index,delta,beta,theta,
     1              bestnowrefl,exprefl,bestrefl,bestthick,status,
     1              x_coordinates,y_coordinates,cost,thickconst,
     1              consskip,ftol,itrtemp)

3       call free (b)
        call free (d)
        call free (c)
        call free (e)
        call free (r1)
        call free (r2)
        call free (r3)
        call free (t)
        call free (p)

C       Graphics
        write(*,*)
        call EXPG_GR_OUT_CGM (status)
```

```fortran
      write(*,*)
      write(*,*) ' <return> to end program'
      read(*,*)
      call EXPG_GR_CLOSE_GRAPHICS(status)
      if (status .ne. Expg_st_good()) then
         call EXPG_ST_OUT(status)
      end if

      end
C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

      subroutine anneal(inname,indat,outdat,outlo1,mincontrol,
     1 maxcontrol,
     1 nsteps,energy,angdep,sym,rho,thetamean,
     1 sim_p,sim_pmin,sim_pmax,sim_pbest,
     1 sim_pdelta,cd_index,delta,beta,theta,
     1 bestnowrefl,exprefl,bestrefl,bestthick,status,
     1 x_coordinates,y_coordinates,cost,thickconst,
     1 consskip,ftol,itrtemp)

C     ===================================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       Performs the annealing using the subroutines                    C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      integer*4 nsteps,i,j,k,l,n,bestj,consskip,ndim,aoe,
     1          init,itr,itrtemp,npts,nlayers,sim_index,
     1          status,start_coordinates(5),end_coordinates(5),
     1          thickconst(2*nlayers),ngood,n_attempts
      common/points/npts,nlayers,ndim,aoe
      common /quality/ ngood,n_attempts

      character*80 inname,indat,outdat,outlo1,sym(nlayers),xleg,yleg
      character*20 angdep
      character*6 x_label
      complex*16 cd_index(nlayers+1,npts)
      real*4 control,mincontrol,maxcontrol,
     1     sim_p(ndim+1,ndim),sim_px(2*3001),
     1     sim_pmin(ndim),
     1     sim_pmax(ndim),sim_pbest(ndim),
     1     bestthick(2*nlayers),sim_pdelta(ndim),
     1     energy(((aoe-1)*npts+(aoe+1))/2),!()=1 if ang,=npts if endep
     1     rho(nlayers), thetamean,
     1     theta(((aoe+1)*npts+(aoe-1))/2),!()=1 if endep, =npts if ang
     1     delta( ((aoe+1)*npts+(aoe-1))/2,((aoe-1)*npts+(aoe+1))/2 ),
     1     beta( ((aoe+1)*npts+(aoe-1))/2 ,((aoe-1)*npts+(aoe+1))/2 ),
     1     cost(ndim+1),bestcost,exprefl(npts),bestnowrefl(npts),
     1     bestrefl(npts),xthick(2*3001),
     1     rate,time,secnds,
     1     x_coordinates(npts,5), y_coordinates(npts,5),
     1     cost1,cost2,cost4,ftol,maxi(2*3001),mini(2*3001)

      parameter (degrad=0.01745329)
      data rate /0./
      implicit none
      external cost1,cost2,cost4

C Executable code:
      time=secnds(0.)
      init=nint(time/2)*2+1
      type *,'initvalue  ',init
      if (aoe.eq.1) then
         call rddat(indat,npts,theta,exprefl)
         call optconan(npts,energy,theta,nlayers,sym,rho,cd_index,
     1             delta,beta,angdep,thetamean)
      else
         call rddat(indat,npts,energy,exprefl)
         call optconen(npts,energy,theta,nlayers,sym,rho,cd_index,
     1             delta,beta)
      end if
      control=maxcontrol

C Graphics before annealing
      if (aoe.eq.1) then
         call reflecan(bestrefl,energy,theta,bestthick,cd_index)
         do i=1,npts
            x_coordinates(i,2)=theta(i)/degrad
            x_coordinates(i,1)=theta(i)/degrad
         end do
         x_label='angle '
      else
         call reflecen(bestrefl,energy,theta,bestthick,cd_index)
         do i=1,npts
             x_coordinates(i,2)=energy(i)
             x_coordinates(i,1)=energy(i)
         end do
         x_label='energy'
      end if
      do i=1,2
         start_coordinates(i)=1
         end_coordinates(i)=npts
      end do
      do i=1,npts
         y_coordinates(i,2)=Alog10(exprefl(i))
         y_coordinates(i,1)=Alog10(bestrefl(i))
      end do
      call EXPG_GR_XYGRAPH(npts,5,2,
     1      start_coordinates,end_coordinates,
     1      x_coordinates,y_coordinates,
     1      'test graph',x_label,'reflectivity',status)

C Prepare points for simplex
      do i=1,ndim
         sim_p(1,i)=sim_pbest(i)
      end do
      time=secnds(0.)
      cost(1)=cost1(sim_pbest,exprefl,cd_index,energy,theta,
     1               bestthick,thickconst)
!     type *,'cost(1)=',cost(1)
      bestcost=cost(1)
      do j=2,ndim+1
         do i=1,ndim
            sim_p(j,i)=sim_pbest(i)
            sim_px(i)=sim_pbest(i)
         end do
         sim_p(j,j-1)=sim_p(j,j-1)+sim_pdelta(j-1)*18
         sim_px(j-1)=sim_px(j-1)+sim_pdelta(j-1)*18
         cost(j)=cost1(sim_px,exprefl,cd_index,energy,theta,
     1                 bestthick,thickconst)
!         type *,'cost(',j,')',cost(j)
      end do
      time=secnds(time)
      type *,'sec.s ',time
      k=0
      n=80
      call blank(outlo1,n)
      open(unit=2,file=outlo1,status='unknown',
     1      access='sequential',form='formatted')
      write(2,'(i8)') int(nsteps/itrtemp/consskip)  ! +1
C Start annealing
      do j=1,int(nsteps/itrtemp)
         itr=itrtemp
         n_attempts=0
         ngood=0
         call amebsa(sim_p,cost,sim_pbest,bestcost,ftol,
     1      cost1,itr,control,exprefl,cd_index,energy,theta,init,
     1      sim_pmin,sim_pmax,bestthick,thickconst,bestj)
         k=k+1
         if (mod(k,consskip).eq.0) then
C Graphics during annealing
C recover all 2*nlayers variables
            sim_index=0
            do i=1,2*nlayers
               if (thickconst(i).eq.1) then
                  xthick(i)=bestthick(i)
               else
                  sim_index=sim_index+1
                  xthick(i)=sim_p(1,sim_index)
                  bestthick(i)=sim_pbest(sim_index)
               end if
            end do

            if (aoe.eq.1) then
               call reflecan(bestrefl,energy,theta,bestthick,cd_index)
```

```
              call reflecan(bestnowrefl,energy,theta,xthick,cd_index)
              do i=1,npts
                 x_coordinates(i,1)=theta(i)/degrad
                 x_coordinates(i,2)=theta(i)/degrad
                 x_coordinates(i,3)=theta(i)/degrad
              end do
           else
              call reflecen(bestrefl,energy,theta,bestthick,cd_index)
              call reflecen(bestnowrefl,energy,theta,xthick,cd_index)
              do i=1,npts
                 x_coordinates(i,1)=energy(i)
                 x_coordinates(i,2)=energy(i)
                 x_coordinates(i,3)=energy(i)
              end do
           end if
           do i=1,3
              start_coordinates(i)=1
              end_coordinates(i)=npts
           end do
           do i=1,npts
              y_coordinates(i,1)=Alog10(bestnowrefl(i))
              y_coordinates(i,2)=Alog10(exprefl(i))
              y_coordinates(i,3)=Alog10(bestrefl(i))
           end do
           call EXPG_GR_XYGRAPH(npts,5,3,
     1          start_coordinates,end_coordinates,
     1          x_coordinates,y_coordinates,
     1          'sasi7, single',x_label,'reflectivity',status)

           rate=ngood/real(n_attempts)
           write(*,*)
           write(*,'(x,a9,4x,a9,4x,a4,6x,a7)') 'iter. no.',
     1          'best iter.','rate','control'
           write(*,'(x,i8,5x,i8,3x,g10.5,2x,g10.5)') j*itrtemp,bestj,
     1          rate,control
           write(*,'(x,a15,x,a19)') '   bestcost    ',
     1          'bestcost in simplex'
           write(*,'(x,g15.7,3x,g15.7)') bestcost,cost(1)
           write(2,'(a9,2x,a9,3x,a4,3x,a7)') 'iter. no.',
     1          'best iter.','rate','control'
           write(2,'(x,i8,3x,i8,x,g10.5,g10.5)') j*itrtemp,bestj,
     1          rate,control
           write(2,'(x,a15,x,a19)') '   bestcost    ',
     1          'bestcost in simplex'
           write(2,'(x,g15.7,3x,g15.7)') bestcost,cost(1)
           write(*,*)
           write(*,'(13x,a13,6x,a15,5x,a15,5x,a15)') 'best till now',
     1          'best in simplex','min. in simplex',
     1          'max. in simplex'
           write(*,'(a9,3x,8(a5,5x))') 'layer no.','thick','rough',
     1           'thick','rough','thick','rough','thick','rough'
           sim_index=0
           do I=1,nlayers*2
               if (thickconst(i).eq.1) then
                  mini(i)=bestthick(i)
                  maxi(i)=bestthick(i)
               else
                  sim_index=sim_index+1
                  maxi(i)=sim_p(1,sim_index)
                  mini(i)=sim_p(1,sim_index)
                  do l=2,ndim+1
                     if (sim_p(l,sim_index).lt.mini(i)) then
                         mini(i)=sim_p(l,sim_index)
                     else if (sim_p(l,sim_index).gt.maxi(i)) then
                         maxi(i)=sim_p(l,sim_index)
                     end if
                  end do
               end if
            end do
            do i=1,nlayers
               write(*,'(i8,2x,4(g10.5,g10.5))') i,bestthick(i),
     1         bestthick(i+nlayers),xthick(i),xthick(i+nlayers),
     1         mini(i),mini(i+nlayers),maxi(i),maxi(i+nlayers)
           end do
           rate=0
        end if   ! of if consskip...
        call cool(control,maxcontrol,mincontrol,nsteps,itrtemp)
     end do
```

```
        close(2)

C Save results
        if (aoe.eq.1) then
           call reflecan(bestrefl,energy,theta,bestthick,cd_index)
           xleg='angle'
           yleg='reflectivity exp  calc'
           call wrtfile(outdat,npts,theta,exprefl,bestrefl,inname,
     1       xleg,yleg,nlayers,bestthick)
        else
           call reflecen(bestrefl,energy,theta,bestthick,cd_index)
           xleg='energy'
           yleg='reflectivity exp  calc'
           call wrtfile(outdat,npts,energy,exprefl,bestrefl,inname,
     1       xleg,yleg,nlayers,bestthick)
        end if

        return
        end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine cool(control,maxcontrol,mincontrol,nsteps,itrtemp)
C       ==============================================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Lowers the control parameter (temperature) exponentially       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nsteps,itrtemp
        real*4 control,maxcontrol,mincontrol
        implicit none

C Executable code:

        control=control*exp(itrtemp*log(mincontrol/maxcontrol)/nsteps)

        return
        end


C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        function cost1(sim_px,exprefl,cd_index,energy,theta,bestthick,
     1               thickconst)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        integer*4 npts,nlayers,i,ndim,thickconst(2*nlayers),aoe,
     1            sim_index
        common/points/npts,nlayers,ndim,aoe
        real*4 exprefl(npts), calcrefl(3001),kost,cost1,sim_px(ndim),
     1  energy(((aoe-1)*npts+(aoe+1))/2),!( )=1 if angel, =npts if endep
     1  theta(((aoe+1)*npts+(aoe-1))/2), !( )=npts if endep., =1 if ang
     1  bestthick(2*nlayers),xthick(2*3001)
        complex*16 cd_index(nlayers+1,npts)
        implicit none

C Executable code:
        kost=0.

C       recover all 2*nlayers variables
        sim_index=0
        do I=1,2*nlayers
           if (thickconst(i).eq.1) then
              xthick(i)=bestthick(i)
           else
              sim_index=sim_index+1
              xthick(i)=sim_px(sim_index)
           end if
        end do
        if (aoe.eq.1) then
           call reflecan(calcrefl,energy,theta,xthick,cd_index)
        else
```

```
            call reflecen(calcrefl,energy,theta,xthick,cd_index)
         end if
         do i=1,npts
            kost=kost+(1-(calcrefl(i)/exprefl(i)))**2
     1                +(1-(exprefl(i)/calcrefl(i)))**2
         end do
         cost1=kost/npts

         return
         end


C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

         function cost2(sim_px,exprefl,cd_index,energy,theta,bestthick,
     1                  thickconst)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
         integer*4 npts,nlayers,i,ndim,thickconst(2*nlayers),aoe,
     1              sim_index
         common/points/npts,nlayers,ndim,aoe
         real*4 exprefl(npts), calcrefl(3001),kost,cost2,sim_px(ndim),
     1   energy(((aoe-1)*npts+(aoe+1))/2),!( )=1 if angel, =npts if endep
     1   theta(((aoe+1)*npts+(aoe-1))/2), !( )=npts if endep., =1 if ang
     1   bestthick(2*nlayers),xthick(2*3001)
         complex*16 cd_index(nlayers+1,npts)
         implicit none

C Executable code:
         kost=0.

C        recover all 2*nlayers variables
         sim_index=0
         do i=1,2*nlayers
            if (thickconst(i).eq.1) then
               xthick(i)=bestthick(i)
            else
               sim_index=sim_index+1
               xthick(i)=sim_px(sim_index)
            end if
         end do
         if (aoe.eq.1) then
            call reflecan(calcrefl,energy,theta,xthick,cd_index)
         else
            call reflecen(calcrefl,energy,theta,xthick,cd_index)
         end if
         do i=1,npts
            kost=kost+(1-(calcrefl(i)/exprefl(i)))**2
         end do
         cost2=kost/npts

         return
         end



C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

         function cost4(sim_px,exprefl,cd_index,energy,theta,bestthick,
     1                  thickconst)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
         integer*4 npts,nlayers,i,ndim,thickconst(2*nlayers),aoe,
     1              sim_index
         common/points/npts,nlayers,ndim,aoe
         real*4 exprefl(npts), calcrefl(3001),kost,cost4,sim_px(ndim),
     1   energy(((aoe-1)*npts+(aoe+1))/2),!( )=1 if angel, =npts if endep
     1   theta(((aoe+1)*npts+(aoe-1))/2), !( )=npts if endep., =1 if ang
     1   bestthick(2*nlayers),xthick(2*3001)
         complex*16 cd_index(nlayers+1,npts)
         implicit none

C Executable code:
         kost=0.

C        recover all 2*nlayers variables
         sim_index=0
         do i=1,2*nlayers
            if (thickconst(i).eq.1) then
               xthick(i)=bestthick(i)
            else
               sim_index=sim_index+1
               xthick(i)=sim_px(sim_index)
            end if
         end do
         if (aoe.eq.1) then
            call reflecan(calcrefl,energy,theta,xthick,cd_index)
         else
            call reflecen(calcrefl,energy,theta,xthick,cd_index)
         end if
         do i=1,npts
            kost=kost+log(1+(1-(calcrefl(i)/exprefl(i)))**2
     1                +(1-(exprefl(i)/calcrefl(i)))**2)
         end do
         cost4=kost/npts

         return
         end

C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12
C Jan-92

         subroutine optconan(npts,energy,theta,nlayers,sym,rho,cd_index,
     1   delta,beta,angdep,thetamean)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Finds optical constants for each layer and for each angle      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

         character*20 unit,angdep
         integer nlayers,npts,i,j,k,n
         character*80 sym(3001)
         complex*16 cd_index(nlayers+1,npts)
         real*4 delta(npts,1),beta(npts,1),
     1          energy(1),theta(npts),rho(3001),thetamean
         parameter (degrad=0.01745329)
         implicit none

C Executable code
!          type *,'in optconan'
         unit= 'ev'
         do i=1,npts
            theta(i)=theta(i)*degrad
         end do
         n=3
         call blank(angdep,n)
         if (angdep .eq. 'y  ') then
            type *,sym(1)
            call f12(npts,1,sym(1),energy,unit,theta,delta,beta,rho(1))
!             type *,' called f12 (1)'
            do i=1,npts
               cd_index(1,i)= dcmplx(1-delta(i,1),-beta(i,1))
               cd_index(nlayers+1,i) =(1.,0.)
            end do
            do j=2,nlayers
               do k=1,j-1
                  if (sym(j) .eq. sym(k)) then
                     do i=1,npts
                        cd_index(j,i)=cd_index(k,i)
!                       type *,cd_index(j,i)
                     end do
                     go to 111
                  end if
               end do
              call f12(npts,1,sym(j),energy,unit,theta,delta,beta,rho(j))
!               type *,' called f12'
```

```
            do i=1,npts
               cd_index(j,i)= dcmplx(1-delta(i,1),-beta(i,1))
!              type *,cd_index(j,i)
            end do
            go to 111
111       end do
!          type *,'Found optical constants'
        end if

        if (angdep .eq. 'n  ') then
        call f12(1,1,sym(1),energy,unit,thetamean,delta,beta,rho(1))
!       type *,' called f12  (1)'
        do i=1,npts
            cd_index(1,i)= dcmplx(1-delta(1,1),-beta(1,1))
            cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
            do k=1,j-1
                if (sym(j) .eq. sym(k)) then
                do i=1,npts
                    cd_index(j,i)=cd_index(k,i)
!                   type *,cd_index(j,i)
                end do
                go to 112
                end if
            end do
            call f12(1,1,sym(j),energy,unit,thetamean,delta,beta,rho(j))
!               type *,' called f12'
            do i=1,npts
                cd_index(j,i)= dcmplx(1-delta(1,1),-beta(1,1))
!               type *,cd_index(j,i)
            end do
            go to 112
112     end do
!       type *,'Found optical constants'
        end if

        return
        end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine reflecan(refl,energy,theta,xthick,cd_index)


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Calculates reflectivity of a multilayer structure as a function of angle C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer*4 npts,nlayers,i,j,ndim,aoe
        complex*16 cd_index(nlayers+1,npts), cd_r, cd_x_zero, cd_x_top
     1          , cd_thet1, cd_thet2, cd_exp_phi
        real*8 four_pi_on_lam,cos_sqr_th,sinthe
        real*4 refl(npts),energy(1),theta(npts),xthick(2*nlayers)
        parameter (four_pi_hc= 1.013536e-3) ! =4*pi/hc
        common/points/npts,nlayers,ndim,aoe
        implicit none

C Executable code:
        four_pi_on_lam= four_pi_hc* energy(1)

        do i=1, npts
            cos_sqr_th= cos( theta(i))**2
            sinthe= sin( theta(i))
            cd_x_zero= (0., 0.) ! reflected amplitude=0 at substrate
            cd_x_top= (0., 0.)
            do j=1,nlayers
                ! njsinthj= sqrt(nj^2-cos^2(th_inc))
                cd_thet2= cdsqrt( cd_index(j,i)**2- cos_sqr_th)
                if(dimag(cd_thet2) .gt. 0.) cd_thet2= dconjg(cd_thet2)
                cd_thet1= cdsqrt( cd_index(j+1,i)**2-cos_sqr_th)
                cd_r = (cd_thet1- cd_thet2)/( cd_thet1+ cd_thet2)
                ! include debye-waller roughness term
                cd_r=cd_r*exp(-(four_pi_on_lam*
     1                 xthick(j+nlayers)*sinthe)**2)
                cd_exp_phi=cdexp( (0.,-1.d0)*
     1                  four_pi_on_lam* xthick(j)* cd_thet2)
                cd_x_top= ( cd_r+ cd_x_zero*cd_exp_phi)
     1                  /( 1.+ cd_r* cd_x_zero*cd_exp_phi)
                cd_x_zero= cd_x_top

!               write(*,*) cd_x_top
!               write(*,*) cd_x_zero
!               write(*,*) cd_exp_phi
!               write(*,*) cd_r
!               write(*,*) cd_thet1
!               write(*,*) cd_thet2
            end do
            refl(i) = cdabs(cd_x_top)**2
!           write(*,*) theta(i),refl(i)
        end do

        return
        end
!***********************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
! sept-91

        subroutine optconen(npts,energy,theta,nlayers,sym,rho,cd_index,
     1  delta,beta)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Finds optical constants for each layer and for each energy     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        character*20 unit
        integer nlayers,npts,i,j,k
        character*80 sym(3001)
        complex*16 cd_index(nlayers+1,npts)
        real*4 delta(1,npts),beta(1,npts),energy(npts)
     1  ,theta(1),rho(3001)
        parameter (degrad=0.01745329)
        implicit none


C executable code

        theta(1)=theta(1)*degrad
        unit= 'ev'
        call f12(1,npts,sym(1),energy,unit,theta,delta,beta,rho(1))
!       type *,' called f12  (1)'
        do i=1,npts
            cd_index(1,i)= dcmplx(1-delta(1,i),-beta(1,i))
            cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
            do k=1,j-1
                if (sym(j) .eq. sym(k)) then
                do i=1,npts
                    cd_index(j,i)=cd_index(k,i)
!                   type *,cd_index(j,i)
                end do
                go to 111
                end if
            end do
        call f12(1,npts,sym(j),energy,unit,theta,delta,beta,rho(j))
!           type *,' called f12'
            do i=1,npts
                cd_index(j,i)= dcmplx(1-delta(1,i),-beta(1,i))
!               type *,cd_index(j,i)
            end do
            go to 111
111     end do
!       type *,'Found optical constants'

        return
        end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine reflecen(refl,energy,theta,xthick,cd_index)
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Calculates reflectivity of a multilayer structure as a function of energy C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nlayers,npts,i,j,ndim,aoe
        complex*16 cd_index(nlayers+1,npts), cd_r, cd_x_zero, cd_x_top
     1      , cd_thet1, cd_thet2, cd_exp_phi
        real*8 four_pi_on_lam, cos_sqr_th, sinthe
        real*4 refl(npts),energy(npts),theta(1),xthick(2*nlayers)
        parameter (four_pi_hc= 1.013536e-3) ! =4*pi/hc
        common/points/npts,nlayers,ndim,aoe
        implicit none

C executable code:

        cos_sqr_th= cos( theta(1))**2
        sinthe= sin( theta(1))

        do i=1, npts
            four_pi_on_lam= four_pi_hc* energy(i)
            cd_x_zero= (0., 0.) ! reflected amplitude=0 at substrate
            cd_x_top= (0., 0.)
            do j=1,nlayers
                ! njsinthj= sqrt(nj^2-cos^2(th_inc))
                cd_thet2= cdsqrt( cd_index(j,i)**2- cos_sqr_th)
                if(dimag(cd_thet2) .gt. 0.) cd_thet2= dconjg(cd_thet2)
                cd_thet1= cdsqrt( cd_index(j+1,i)**2-cos_sqr_th)
                cd_r = (cd_thet1- cd_thet2)/( cd_thet1+ cd_thet2)
                ! include debye-waller roughness term
                cd_r= cd_r*exp(- (four_pi_on_lam*
     1                    xthick(j+nlayers)*sinthe)**2)
                cd_exp_phi=cdexp( (0.,-1.d0)*
     1                  four_pi_on_lam* xthick(j)* cd_thet2)
                cd_x_top= ( cd_r+ cd_x_zero*cd_exp_phi)
     1                  /( 1.+ cd_r* cd_x_zero*cd_exp_phi)
                cd_x_zero= cd_x_top

!               write(*,*) cd_x_top
!               write(*,*) cd_x_zero
!               write(*,*) cd_exp_phi
!               write(*,*) cd_r
!               write(*,*) cd_thet1
!               write(*,*) cd_thet2
            end do

            refl(i) = cdabs(cd_x_top)**2
!       write(*,*) energy(i),refl(i)
        end do

        return
        end

C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine rddat(datname,npts,xval,yval)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       Reads the data file                                             C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        real*4 xval(npts),yval(npts)
        integer*4 npts,nskip,erval,i,n,fgetc
        character*80 datname
        character char
        parameter (nskip=4)
        implicit none

! Executable code:
        type  *,'npts in rddat',npts
        n=80
        call blank(datname,n)
        open(unit=3,file=datname,status='old',access='sequential',
     1      form='formatted')
        i=0
        do while (i.lt.nskip)         ! Skip over nskip (text) lines
            erval=fgetc(3,char)
            if  (char.eq.'\n') then    ! If char = return-character
               i=i+1
            end if
        end do
        do i=1,npts
            read (3,*) xval(i),yval(i)
            type *,xval(i),yval(i),npts
        end do

        close(3)

        return
        end

!*************************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

        subroutine rdfile(inname,indat,outdat,outlol,anoren,mincontrol,
     1  maxcontrol,nsteps,eng,thetmin,thetmax,dthet,angdep,the,
     1  engmin,engmax,deng,nlayers,allortwo,sym,rho,
     1  bestthick,minthick,maxthick,deltathick,npts,consskip,
     1  ftol,itrtemp)

        integer n,i
        character*80 inname,outdat,indat,sym(3001),outlol
        character*72 text
        character*5 anoren,angdep,allortwo
        real*4 eng,thetmin,thetmax,dthet,the,engmin,engmax,deng,
     1  bestthick(2*3001),ftol,
     1  rho(3001),mincontrol,maxcontrol,
     1  minthick(3001*2),maxthick(3001*2),deltathick(3001*2)
        integer*4 nlayers,nsteps,npts,consskip,itrtemp
        implicit none


! Executable code:
        open(unit=2,file=inname,status='old',
     1          access='sequential',form='formatted')
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(20x,a35)') indat
                read(2,'(21x,a35)') outdat
                read(2,'(18x,a80)') outlol
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(13x,a3)') anoren
                read(2,'(a72)') text
                read(2,'(39x,f5.1)') maxcontrol
                read(2,'(37x,f5.1)') mincontrol
                read(2,'(21x,i8)') nsteps
                read(2,'(21x,i8)') npts
                read(2,'(a72)') text
                read(2,'(11x,f8.1)') eng
                read(2,'(23x,f5.1)') thetmin
                read(2,'(23x,f5.1)') thetmax
                read(2,'(25x,f8.1)') dthet
                read(2,'(a72)') text
                read(2,'(48x,a3)') angdep
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(15x,f5.1)') the
                read(2,'(19x,f8.1)') engmin
                read(2,'(19x,f8.1)') engmax
                read(2,'(21x,f5.1)') deng
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(35x,i5)')  nlayers
                read(2,'(a72)') text
                read(2,'(a72)') text
                read(2,'(25x,a5)') allortwo
                read(2,'(a72)') text
                read(2,'(a72)') text
            read(2,'(a72)') text
      n=3
      call blank(allortwo,n)
      if (allortwo .eq. '2  ') then
      do i=1,3
         read(2,'(a15,f8.1,f8.1,f8.1,f8.1,f8.1,f8.1
     1 ,f8.1,f8.1,f8.1)') sym(i),rho(i),bestthick(i+nlayers),
     1    minthick(i+nlayers),maxthick(i+nlayers),
     1    deltathick(i+nlayers),bestthick(i),minthick(i),
     1    maxthick(i),deltathick(i)
      end do
      do i=4,nlayers,2
          sym(i)=sym(2)
          rho(i)=rho(2)
          bestthick(i+nlayers)=bestthick(2+nlayers)
          minthick(i+nlayers)=minthick(2+nlayers)
          maxthick(i+nlayers)=maxthick(2+nlayers)
          deltathick(i+nlayers)=deltathick(2+nlayers)
          bestthick(i)=bestthick(2)
          minthick(i)=minthick(2)
          maxthick(i)=maxthick(2)
          deltathick(i)=deltathick(2)
          sym(i+1)=sym(3)
          rho(i+1)=rho(3)
          bestthick(i+1+nlayers)=bestthick(3+nlayers)
          minthick(i+1+nlayers)=minthick(3+nlayers)
          maxthick(i+1+nlayers)=maxthick(3+nlayers)
          deltathick(i+1)=deltathick(3)
          bestthick(i+1)=bestthick(3)
          minthick(i+1)=minthick(3)
          maxthick(i+1)=maxthick(3)
          deltathick(i+1)=deltathick(3)
      end do
      end if
      if (allortwo .eq. 'al ') then
      do i=1,nlayers
           read(2,'(a15,f8.1,f8.1,f8.1,f8.1,f8.1,f8.1
     1 ,f8.1,f8.1,f8.1)') sym(i),rho(i),bestthick(i+nlayers),
     1 minthick(i+nlayers),maxthick(i+nlayers),
     1 deltathick(i+nlayers),bestthick(i),minthick(i),
     1 maxthick(i),deltathick(i)
           n=15
           call blank(sym(i),n)
           type *,bestthick(i),bestthick(i+nlayers)
      end do
      end if
      read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(40x,i8)') consskip
      read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(25x,f7.3)') ftol
      read(2,'(50x,i8)') itrtemp
      close(2)
      end

!****************************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

      subroutine wrtfile(outname,npts,xval,yval,y2val,title,xleg,
     1             yleg,nlayers,thick)

      real*4 xval(2001),yval(2001),y2val(2001),thick(3001)
      integer*4 npts,nlayers,n,i
      character*80 title,xleg,yleg
      character*80 outname
      implicit none

      n=80
      call blank(outname,n)
      call blank(title,n)
      call blank(xleg,n)
      call blank(xleg,n)

      open(unit=3,file=outname,status='unknown',access='sequential',
     1      form='formatted')

      write (3,'(a)') title
      do i=1,nlayers
         write (3,*) i,thick(i),thick(i+nlayers)
      end do
      write (3,'(a)') xleg
      write (3,'(a)') yleg
      write (3,'(i6)') npts

      do i=1,npts
         write (3,'(3(1pg14.7))'),xval(i),yval(i),y2val(i)
      end do
      close(3)

      end

C*******The simplex subroutines

      subroutine amebsa(p,y,pb,yb,ftol,funk,iter,temptr,exprefl,
     1             cd_index,energy,theta,init,minp,maxp,bestthick,
     1             thickconst,bestj)

      integer iter,ndim,npts,nlayers,thickconst(2*nlayers),aoe,bestj,
     1        ngood,n_attempts
      real*4 ftol,temptr,yb,p(ndim+1,ndim),pb(ndim),y(ndim+1),
     1   funk,ran,amotsa,exprefl(npts),
     1   energy(((aoe-1)*npts+(aoe+1))/2),!()=1 if ang,=npts if endep
     1   theta(((aoe+1)*npts+(aoe-1))/2),!()=1 if endep, =npts if ang
     1   minp(ndim),maxp(ndim),bestthick(2*nlayers)
C     uses amotsa,funk,ran
      integer i,init,ihi,ilo,inhi,j,m,n
      real rtol,sum,swap,tt,yhi,ylo,ynhi,ysave,yt,ytry,psum(3001*2)
      complex*16 cd_index(nlayers+1,npts)

      common /points/npts,nlayers,ndim,aoe
      common /quality/ ngood,n_attempts
      external funk
      implicit none

C     Executable code:

      tt=-temptr
1     do n=1,ndim
         sum=0.
         do m=1,ndim+1
            sum=sum+p(m,n)
         end do
         psum(n)=sum
      end do
2     ilo=1
      inhi=1
      ihi=2
      ylo=y(1)+tt*log(ran(init))
      ynhi=ylo
      yhi=y(2)+tt*log(ran(init))
      if (ylo.gt.yhi) then
         ihi=1
         inhi=2
         ilo=2
         ynhi=yhi
         yhi=ylo
         ylo=ynhi
      endif
      do i=3,ndim+1
         yt=y(i)+tt*log(ran(init))
         if (yt.le.ylo) then
            ilo=i
            ylo=yt
         end if
         if (yt.gt.yhi) then
            inhi=ihi
            ynhi=yhi
            ihi=i
            yhi=yt
         else if (yt.gt.ynhi) then
            inhi=i
            ynhi=yt
         end if
      enddo
      rtol=abs(yhi-ylo)/(abs(yhi)+abs(ylo)) ! factor 2 diff from orig.
      if (rtol.lt.ftol.or.iter.lt.0) then
         type*,'rtol',rtol
         swap=y(1)
         y(1)=y(ilo)
         y(ilo)=swap
         do n=1,ndim
            swap=p(1,n)
            p(1,n)=p(ilo,n)
            p(ilo,n)=swap
         end do
         return
      end if
      iter=iter-2
      ytry=amotsa(p,y,psum,pb,yb,funk,ihi,yhi,-1.0,
     1      exprefl,cd_index,energy,theta,init,tt,
     1      minp,maxp,bestthick,thickconst,bestj,iter)
      if (ytry.le.ylo) then
         ytry=amotsa(p,y,psum,pb,yb,funk,ihi,yhi,2.0,
     1         exprefl,cd_index,energy,theta,init,tt,
     1         minp,maxp,bestthick,thickconst,bestj,iter)
      else if (ytry.ge.ynhi) then
         ysave=yhi
         ytry=amotsa(p,y,psum,pb,yb,funk,ihi,yhi,.5,
     1         exprefl,cd_index,energy,theta,init,tt,
     1         minp,maxp,bestthick,thickconst,bestj,iter)
         if (ytry.ge.ysave) then
            do i=1,ndim+1
               if (i.ne.ilo) then
                  do j=1,ndim
                     psum(j)=0.5*(p(i,j)+p(ilo,j))
                     p(i,j)=psum(j)
                  end do
                  y(i)=funk(psum,exprefl,cd_index,energy,theta,
     1                      bestthick,thickconst)
               end if
            end do
            iter=iter-ndim
            goto 1
         end if
      else
         iter=iter+1
      end if
      goto 2
      end


      function amotsa(p,y,psum,pb,yb,funk,ihi,yhi,fac,
     1          exprefl,cd_index,energy,theta,init,tt,minp,
     1          maxp,bestthick,thickconst,bestj,iter)

      integer ihi,ndim,thickconst(2*nlayers),aoe,npts,nlayers,
     1        ngood,n_attempts
      real*4 amotsa,fac,yb,yhi,p(ndim+1,ndim),pb(ndim),psum(ndim),
     1   y(ndim+1),funk,ran,exprefl(npts),
     1   energy(((aoe-1)*npts+(aoe+1))/2),!()=1 if ang,=npts if endep
     1   theta(((aoe+1)*npts+(aoe-1))/2),!()=1 if endep, =npts if ang
     1   minp(ndim),maxp(ndim),bestthick(2*nlayers)
C     uses funk,ran
      integer init,j,bestj,iter
      real fac1,fac2,tt,yflu,ytry,ptry(3001*2)
      complex*16 cd_index(nlayers+1,npts)
      common /points/ npts,nlayers,ndim,aoe
      common /quality/ ngood,n_attempts

      implicit none

C     Executable code:

      fac1=(1.-fac)/ndim
      fac2=fac1-fac
      do j=1,ndim
         ptry(j)=psum(j)*fac1-p(ihi,j)*fac2
         if (ptry(j).lt.minp(j)) then
            ptry(j)=ptry(j)+maxp(j)-minp(j)
         else if (ptry(j).gt.maxp(j)) then
            ptry(j)=ptry(j)-maxp(j)+minp(j)
         end if
      end do
      ytry=funk(ptry,exprefl,cd_index,energy,theta,bestthick,
     1          thickconst)
      if (ytry.le.yb) then
         do j=1,ndim
            pb(j)=ptry(j)
         end do
         yb=ytry
         bestj=iter
      end if
      yflu=ytry-tt*log(ran(init))
      n_attempts=n_attempts+1
      if (yflu.lt.yhi) then
         ngood=ngood+1
         y(ihi)=ytry
         yhi=yflu
         do j=1,ndim
            psum(j)=psum(j)-p(ihi,j)+ptry(j)
            p(ihi,j)=ptry(j)
         end do
      end if
      amotsa=yflu
      return
      end
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        program sasi8
C       =============
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Simulated Annealing Multilayer program, no ensembles, expon. cooling C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C The program performs simulated annealing on a multilayer structure in
C order to optimize the fit to a reflectivity curve. A number of
C subroutines are used:
C
C anneal        Performs the annealing using the subroutines
C rdfile        Reads the input file
C rddata        Reads the data file (experimental reflectivity curve)
C optconan      Finds optical constants, if angle is variable
C optconen      Finds optical constants, if energy is variable
C reflecan      Calculates the reflectivity of a multilayer structure
C reflecen      Same; for energy as variable
C cost          Calculates the cost function of a reflectivity curve
C neighbour     Randomly chooses a neighbour to the current solution
C accept        Decides whether or not to accept the new solution (cost)
C cool          Lowers the temperature exponentially
C wrtfile       Writes the result to a file
C
C The optical method is based on an article by:
C   W.J. Bartels, J. Hornstra and D.J.W. Lobeek, Acta. Cryst. A42, 539 (1986)
C
C The simulated annealing is based on a report by:
C   E. Schroeder, S. Friis-Jensen, Physics Laboratory, U. of Copenhagen (1989)
C
C Elsebeth Schroder, Peter HOghOj, ESRF, June-92
C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

        integer*4 nlayers,npts,size1,size2,byt1,byt2,nsteps,malloc,n,
     1            aoe,consskip
        character*80 inname,indat,outdat,outlo1,sym(3001)
        character*20 anoren,angdep,allortwo
        real*4 thick(2*3001),
     1  rho(3001),thetmin,thetmax,dthet,engmin,
     1  engmax,deng,eng,the,mincontrol,maxcontrol,minthick(2*3001),
     1  maxthick(2*3001),deltathick(2*3001),bestthick(2*3001),
     1  thetamean
        real*4 delta,beta,theta,calcrefl,exprefl,bestrefl,energy
        complex*16 cd_index
        pointer (c,cd_index) ,(d,delta),(b,beta),(t,theta),(r1,calcrefl)
        pointer  (r2,exprefl),(r3,bestrefl),(e,energy)
        parameter (byt1=4,byt2=16)
        data n /3/
        integer status
        integer Expg_st_good
        external Expg_st_good
        Implicit None
        real x_coordinates(3001,5),y_coordinates(3001,5)
        common/points/npts,nlayers,aoe

C       Executable code:

        write(*,'(''Enter input file name: '')')
        read(*,'(a50)') inname

        call rdfile(inname,indat,outdat,outlo1,anoren,mincontrol,
     1  maxcontrol,nsteps,eng,thetmin,thetmax,dthet,angdep,the,
     1  engmin,engmax,deng,nlayers,allortwo,sym,rho,
     1  thick,minthick,maxthick,deltathick,npts,consskip)

C       Graphics
        status=Expg_st_good()
        call EXPG_GR_OPEN_GRAPHICS(status)
!        call EXPG_GR_SET_XLABELSTYLE(.true.,1,3,1.5,1.5,1.0,status)
        call EXPG_GR_SET_CURVELINE(2,1,7,2.0,.false.,0,status)
        call EXPG_GR_SET_CURVELINE(3,1,5,2.0,.false.,0,status)
        call EXPG_GR_SET_CURVESTYLE(1,.true.,.false.,.false.,status)
        call EXPG_GR_SET_CURVESTYLE(2,.true.,.false.,.false.,status)
        call EXPG_GR_SET_CURVESTYLE(3,.true.,.false.,.false.,status)
```

```
        call blank(anoren,n)
        if (anoren .eq. 'a  ') then
            aoe= 1
!            npts = int((thetmax-thetmin)/dthet+1.+1.0e-15)
            size1=npts*byt1
            size2=npts*(nlayers+1)*byt2
            b=malloc(size1)
            d=malloc(size1)
            c=malloc(size2)
            e=malloc(byt1)
            r1=malloc(size1)
            r2=malloc(size1)
            r3=malloc(size1)
            t=malloc(size1)
            energy=eng
            thetamean=(thetmax+thetmin)/2
        else if (anoren .eq. 'e  ') then
            aoe=-1
!           npts = int((engmax-engmin)/deng+1.+1.0e-15)
            size1=npts*byt1
            size2=npts*(nlayers+1)*byt2
            b=malloc(size1)
            d=malloc(size1)
            c=malloc(size2)
            e=malloc(size1)
            r1=malloc(size1)
            r2=malloc(size1)
            r3=malloc(size1)
            t=malloc(byt1)
            theta=the
        else
           type *,'Error in file',inname
           type *,'- neither angle (a) nor energy (e) to vary'
           goto 3
        end if

        call anneal(inname,indat,outdat,outlo1,mincontrol,maxcontrol,
     1  nsteps,energy,angdep,sym,rho,thetamean,
     1  thick,minthick,maxthick,
     1  deltathick,cd_index,delta,beta,theta,
     1  calcrefl,exprefl,bestrefl,bestthick,status,
     1  consskip,x_coordinates,y_coordinates)

3       call free (b)
        call free (d)
        call free (c)
        call free (e)
        call free (r1)
        call free (r2)
        call free (r3)
        call free (t)

C       Graphics
        write(*,*)
        call EXPG_GR_OUT_CGM (status)
        write(*,*)
        write(*,*) ' <return> to end program'
        read(*,*)
        call EXPG_GR_CLOSE_GRAPHICS(status)
        if (status .ne. Expg_st_good()) then
           call EXPG_ST_OUT(status)
        end if

        end
C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine anneal(inname,indat,outdat,outlo1,mincontrol,
     1  maxcontrol,nsteps,energy,angdep,sym,rho,thetamean,
     1  thick,minthick,maxthick,
     1  deltathick,cd_index,delta,beta,theta,
     1  calcrefl,exprefl,bestrefl,bestthick,status,
     1  consskip,x_coordinates,y_coordinates)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       Performs the annealing using the subroutines                   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      integer nsteps,nlayers,npts,good,i,j,bestj,consskip,
     1         init,aoe,n,upgood,
     1         status,start_coordinates(5),end_coordinates(5)
      character*80 inname,indat,outdat,outlol,sym(nlayers),xleg,yleg
      character*20 angdep
      character*6 x_label
      complex*16 cd_index(nlayers+1,npts)
      real*4 control,mincontrol,maxcontrol,thetamean,
     1    thick(2*nlayers),minthick(2*nlayers),maxthick(2*nlayers),
     1    deltathick(2*nlayers),newthick(2*3001),
     1    bestthick(2*nlayers),
     1    energy(((aoe-1)*npts+(aoe+1))/2),!()=1 if ang,=npts if endep
     1    theta(((aoe+1)*npts+(aoe-1))/2),!()=1 if endep, =npts if ang
     1    delta( ((aoe+1)*npts+(aoe-1))/2,((aoe-1)*npts+(aoe+1))/2 ),
     1    beta( ((aoe+1)*npts+(aoe-1))/2 ,((aoe-1)*npts+(aoe+1))/2 ),
     1    rho(nlayers),
     1    oldcost,newcost,bestcost,exprefl(npts),calcrefl(npts),
     1    bestrefl(npts),
     1    rate,uprate,time,time1,time2,secnds,
     1    x_coordinates(npts,5), y_coordinates(npts,5)
      parameter (degrad=0.01745329)
      implicit none
      common/points/npts,nlayers,aoe

C Executable code:
      time=secnds(0.)
      init=nint(time/2)*2+1
      type *,'initvalue  ',init
      time1=secnds(0.)
      if (aoe.eq.1) then
         call rddat(indat,npts,theta,exprefl)
         call optconan(npts,energy,theta,nlayers,sym,rho,cd_index,
     1        delta,beta,angdep,thetamean)
         call reflecan(bestrefl,energy,theta,thick,cd_index)
      else
         call rddat(indat,npts,energy,exprefl)
         call optconen(npts,energy,theta,nlayers,sym,rho,cd_index,
     1        delta,beta)
         call reflecen(bestrefl,energy,theta,thick,cd_index)
      end if
      time2=secnds(time1)
      type *,'sec.s:',time2
      call cost1(exprefl,bestrefl,npts,oldcost)
      bestcost=oldcost
      do i=1,nlayers*2
         bestthick(i)=thick(i)
         newthick(i)=thick(i)
      end do
      control=maxcontrol
      uprate=0
      rate=0

C     Graphics before annealing
      if (aoe.eq.1) then
         do i=1,npts
            x_coordinates(i,2)=theta(i)/degrad
            x_coordinates(i,1)=theta(i)/degrad
         end do
         x_label='angle '
      else
         do i=1,npts
             x_coordinates(i,2)=energy(i)
             x_coordinates(i,1)=energy(i)
         end do
         x_label='energy'
      end if
      do i=1,2
         start_coordinates(i)=1
         end_coordinates(i)=npts
      end do
      do i=1,npts
         y_coordinates(i,2)=Alog10(exprefl(i))
         y_coordinates(i,1)=Alog10(bestrefl(i))
      end do
      call EXPG_GR_XYGRAPH(npts,5,2,
     1     start_coordinates,end_coordinates,
     1     x_coordinates,y_coordinates,
```

```
     1        'test graph, sasi8',x_label,'reflectivity',status)

      do i=1,nlayers
         type *,bestthick(i),bestthick(i+nlayers)
      end do
      n=80
      call blank(outlol,n)
      open(unit=2,file=outlol,status='unknown',
     1       access='sequential',form='formatted')
      write(2,'(i8)') int(nsteps/consskip)
C start annealing
      do j=1,nsteps
         call neighbour(thick,newthick,minthick,maxthick,deltathick,
     1                  nlayers,init)
         if (aoe.eq.1) then
            call reflecan(calcrefl,energy,theta,newthick,
     1           cd_index)
         else
            call reflecen(calcrefl,energy,theta,newthick,
     1           cd_index)
         end if
         call cost1(exprefl,calcrefl,npts,newcost)
         call accept(oldcost,newcost,control,good,upgood,init)
         if (good .eq. 1) then
            do i=1,2*nlayers
               thick(i)=newthick(i)
            end do
            oldcost=newcost
            if (oldcost .lt. bestcost) then
               bestcost=oldcost
               bestj=j
               do i=1,npts
                  bestrefl(i)=calcrefl(i)
               end do
               do i=1,nlayers*2
                  bestthick(i)=thick(i)
               end do
            endif
         else
            do i=1,2*nlayers
               newthick(i)=thick(i)
            end do
         endif
         rate=rate+good
         uprate=uprate+upgood
         if (mod(j,consskip).eq.0) then

C        Graphics
            do i=1,3
               start_coordinates(i)=1
               end_coordinates(i)=npts
            end do
            if (aoe.eq.1) then
               do i=1,npts
                  x_coordinates(i,1)=theta(i)/degrad
                  x_coordinates(i,2)=theta(i)/degrad
                  x_coordinates(i,3)=theta(i)/degrad
               end do
            else
               do i=1,npts
                  x_coordinates(i,1)=energy(i)
                  x_coordinates(i,2)=energy(i)
                  x_coordinates(i,3)=energy(i)
               end do
            end if
            do i=1,npts
               y_coordinates(i,1)=Alog10(calcrefl(i))
               y_coordinates(i,2)=Alog10(exprefl(i))
               y_coordinates(i,3)=Alog10(bestrefl(i))
            end do
            call EXPG_GR_XYGRAPH(npts,5,3,
     1           start_coordinates,end_coordinates,
     1           x_coordinates,y_coordinates,
     1           'single, expon. cooling',
     1           x_label,'reflectivity',status)

            rate=rate/consskip
            uprate=uprate/consskip
            write(*,*)
            write(*,'(x,a9,4x,a9,4x,a4,6x,a7,4x,a6)') 'iter. no.',
     1           'best iter.','rate','control','uprate'
            write(*,'(x,i8,5x,i8,3x,g10.5,2x,g10.5,2x,g10.5)') j,
     1           bestj,rate,control,uprate
            write(*,'(x,a15,x,a19)') '   bestcost    ',
     1           '     cost now      '
            write(*,'(x,g15.7,3x,g15.7)') bestcost,oldcost
            write(2,'(a9,2x,a9,3x,a4,3x,a7,3x,a6)') 'iter. no.',
     1           'best iter.','rate','control','uprate'
            write(2,'(x,i8,3x,i8,x,g10.5,g10.5,x,g10.5)') j,
     1           bestj,rate,control,uprate
            write(2,'(x,a15,x,a19)') '   bestcost    ',
     1           '     cost now      '
            write(2,'(x,g15.7,3x,g15.7)') bestcost,oldcost
            write(*,*)

            write(*,'(13x,a13,6x,a15)') 'best till now',
     1           '   layers now  '
            write(*,'(a9,3x,4(a5,5x))') 'layer no.','thick','rough',
     1            'thick','rough'

             do i=1,nlayers
               write(*,'(i8,2x,4(g10.5,g10.5))') i,bestthick(i),
     1         bestthick(i+nlayers),thick(i),thick(i+nlayers)
            end do
            rate=0
            uprate=0
          endif
          call cool(control,maxcontrol,mincontrol,nsteps)
        end do
        close(2)
        if (aoe.eq.1) then
          xleg='angle (degrees)'
          yleg='reflectivity exp  calc'
          call wrtfile(outdat,npts,theta,exprefl,bestrefl,inname,
     1        xleg,yleg,nlayers,bestthick)
        else
          xleg='energy (eV)'
          yleg='reflectivity exp  calc'
          call wrtfile(outdat,npts,energy,exprefl,bestrefl,inname,
     1        xleg,yleg,nlayers,bestthick)
        end if

        return
        end

C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine cool(control,maxcontrol,mincontrol,nsteps)
C       ===================================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Lowers the control parameter (temperature) exponentially       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nsteps
        real*4 control,maxcontrol,mincontrol
        implicit none

C Executable code:

        control=control/exp(log(maxcontrol/mincontrol)/nsteps)

        return
        end

C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine accept(oldcost,newcost,control,good,upgood,init)
C       ===========================================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Decides whether to accept neighbour to the current solution    C
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer good,init,upgood
        real*4 oldcost, newcost,control,deltacost,prob,random,ran
        implicit none

C Executable code:

        deltacost=newcost-oldcost
        if (deltacost .le. 0) then
           good=1
           upgood=0
        else
           random=ran(init)
           prob=exp(-deltacost/control)-random
           if (prob .ge. 0) then
              good=1
              upgood=1
           else
              good=0
              upgood=0
           endif
        end if

        return
        end

C************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92
        subroutine neighbour(thick,newthick,minthick,maxthick,
     1            deltathick,nlayers,init)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Randomly chooses a neighbour to the current solution           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nlayers,changel,init
        real*4 thick(2*nlayers),minthick(2*nlayers),maxthick(2*nlayers),
     1         deltathick(2*nlayers),addthick,random,userough,
     1         newthick(2*nlayers),ran
        parameter (userough=0.0)
        implicit none

C Executable code
        random=ran(init)
        changel=int(random*nlayers-1e-15)+1
        random=ran(init)
        if (random .ge. userough) then
           random=ran(init)
           addthick=(int(random*2-1e-15)*2-1)*deltathick(changel)
           newthick(changel)=thick(changel)+addthick
           if (newthick(changel) .lt. minthick(changel)) then
              newthick(changel)=maxthick(changel)
           else if (newthick(changel) .gt. maxthick(changel)) then
              newthick(changel)=minthick(changel)
           endif
        else
           random=ran(init)
           addthick=(int(random*2-1e-15)*2-1)*
     1                        deltathick(changel+nlayers)
           newthick(changel+nlayers)=thick(changel+nlayers)+addthick
           if (newthick(changel+nlayers) .lt.
     1                            minthick(changel+nlayers)) then
              newthick(changel+nlayers)=maxthick(changel+nlayers)
           else if (newthick(changel+nlayers) .gt.
     1                            maxthick(changel+nlayers)) then
              newthick(changel+nlayers)=minthick(changel+nlayers)
           endif
        end if

        return
        end

C************************************************************************
```

```
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine cost1(expdat,calcdat,npts,kost)
C       =========================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer npts,i
        real*4 expdat(2001),calcdat(2001), kost
        implicit none

C Executable code:

        kost=0
        do i=1,npts
           kost=kost+(1-(calcdat(i)/expdat(i)))**2
     1              +(1-(expdat(i)/calcdat(i)))**2
        end do
        kost=kost/npts

        return
        end

C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine cost2(expdat,calcdat,npts,kost)
C       =========================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer npts,i
        real*4 expdat(2001),calcdat(2001), kost
        implicit none

C Executable code:

        kost=0
        do i=1,npts
           kost=kost+(1-(calcdat(i)/expdat(i)))**2
        end do
        kost=kost/npts

        return
        end


C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine cost4(expdat,calcdat,npts,kost)
C       =========================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer npts,i
        real*4 expdat(2001),calcdat(2001), kost
        implicit none

C Executable code:

        kost=0
        do i=1,npts
           kost=kost+log(1+(1-(calcdat(i)/expdat(i)))**2
     1              +(1-(expdat(i)/calcdat(i)))**2)
        end do
        kost=kost/npts

        return
        end

C**************************************************************************
```

```
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine optconan(npts,energy,theta,nlayers,sym,rho,cd_index,
     1  delta,beta,angdep,thetamean)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Finds optical constants for each layer and for each angle      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        character*20 unit,angdep
        integer nlayers,npts,i,n,k,j
        character*80 sym(3001)
        complex*16 cd_index(nlayers+1,npts)
        real*4 delta(npts,1),beta(npts,1),
     1       energy(1),theta(npts),rho(3001),thetamean
        parameter (degrad=0.01745329)
        implicit none


C Executable code
        type *,'in optconan'
        unit= 'ev'
        do i=1,npts
           theta(i)=theta(i)*degrad
        end do
        n=3
        call blank(angdep,n)
        if (angdep .eq. 'y  ') then
        type *,sym(1)
        call f12(npts,1,sym(1),energy,unit,theta,delta,beta,rho(1))
!       type *,' called f12 (1)'
        do i=1,npts
           cd_index(1,i)= dcmplx(1-delta(i,1),-beta(i,1))
           cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
           do k=1,j-1
              if (sym(j) .eq. sym(k)) then
              do i=1,npts
                 cd_index(j,i)=cd_index(k,i)
!                type *,cd_index(j,i)
              end do
              go to 111
              end if
           end do
           call f12(npts,1,sym(j),energy,unit,theta,delta,beta,rho(j))
!          type *,' called f12'
           do i=1,npts
              cd_index(j,i)= dcmplx(1-delta(i,1),-beta(i,1))
!             type *,cd_index(j,i)
           end do
           go to 111
111     end do
!       type *,'Found optical constants'
        end if

        if (angdep .eq. 'n  ') then
        type *,'thetamean=',thetamean
        call f12(1,1,sym(1),energy,unit,thetamean,delta,beta,rho(1))
!       type *,' called f12 (1)'
        do i=1,npts
           cd_index(1,i)= dcmplx(1-delta(1,1),-beta(1,1))
           cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
           do k=1,j-1
              if (sym(j) .eq. sym(k)) then
              do i=1,npts
                 cd_index(j,i)=cd_index(k,i)
!                type *,cd_index(j,i)
              end do
              go to 112
              end if
           end do
           call f12(1,1,sym(j),energy,unit,thetamean,delta,beta,rho(j))
!          type *,' called f12'
           do i=1,npts
```

```
                cd_index(j,i)= dcmplx(1-delta(1,1),-beta(1,1))
!               type *,cd_index(j,i)
            end do
            go to 112
112     end do
!       type *,'Found optical constants'
        end if

        return
        end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine reflecan(refl,energy,theta,thick,cd_index)


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Calculates reflectivity of a multilayer structure as a function of angle C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nlayers,npts,aoe,i,j
        complex*16 cd_index(nlayers+1,npts), cd_r, cd_x_zero, cd_x_top
     1          , cd_thet1, cd_thet2, cd_exp_phi
        real*8 four_pi_on_lam,cos_sqr_th,sinthe
        real*4 refl(npts),energy(1),theta(npts),thick(nlayers*2)
        parameter (four_pi_hc= 1.013536e-3) ! =4*pi/hc
        common/points/npts,nlayers,aoe
        implicit none

C Executable code:
        four_pi_on_lam= four_pi_hc* energy(1)
        do i=1, npts
            cos_sqr_th= cos( theta(i))**2
            sinthe= sin( theta(i))
            cd_x_zero= (0., 0.) ! reflected amplitude=0 at substrate
            cd_x_top= (0., 0.)
            do j=1,nlayers
                ! njsinthj= sqrt(nj^2-cos^2(th_inc))
                cd_thet2= cdsqrt( cd_index(j,i)**2- cos_sqr_th)
                if(dimag(cd_thet2) .gt. 0.) cd_thet2= dconjg(cd_thet2)
                cd_thet1= cdsqrt( cd_index(j+1,i)**2-cos_sqr_th)
                cd_r = (cd_thet1- cd_thet2)/( cd_thet1+ cd_thet2)
                ! include debye-waller roughness term
             cd_r=cd_r*exp(-(four_pi_on_lam*thick(j+nlayers)*sinthe)**2)
                cd_exp_phi=cdexp( (0.,-1.d0)*
     1                  four_pi_on_lam* thick(j)* cd_thet2)
                cd_x_top= ( cd_r+ cd_x_zero*cd_exp_phi)
     1                  /( 1.+ cd_r* cd_x_zero*cd_exp_phi)
                cd_x_zero= cd_x_top

            end do
            refl(i) = cdabs(cd_x_top)**2
        end do
        return
        end
!***********************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
! sept-91

        subroutine optconen(npts,energy,theta,nlayers,sym,rho,cd_index,
     1  delta,beta)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Finds optical constants for each layer and for each energy      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        character*20 unit
        integer nlayers,npts,i,k,j
        character*80 sym(3001)
        complex*16 cd_index(nlayers+1,npts)
        real*4 delta(1,npts),beta(1,npts),energy(npts)
     1  ,theta(1),rho(3001),degrad
        parameter (degrad=0.01745329)
        implicit none
```

```
C executable code
        type *,'in optconen'
        theta(1)=theta(1)*degrad
        unit= 'ev'
        call f12(1,npts,sym(1),energy,unit,theta,delta,beta,rho(1))
!       type *,' called f12 (1)'
        do i=1,npts
            cd_index(1,i)= dcmplx(1-delta(1,i),-beta(1,i))
            cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
            do k=1,j-1
                if (sym(j) .eq. sym(k)) then
                do i=1,npts
                    cd_index(j,i)=cd_index(k,i)
!                   type *,cd_index(j,i)
                end do
                go to 111
                end if
            end do
        call f12(1,npts,sym(j),energy,unit,theta,delta,beta,rho(j))
!           type *,' called f12'
            do i=1,npts
                cd_index(j,i)= dcmplx(1-delta(1,i),-beta(1,i))
!               type *,cd_index(j,i)
            end do
            go to 111
111     end do
!       type *,'Found optical constants'

        return
        end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine reflecen(refl,energy,theta,thick,cd_index)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Calculates reflectivity of a multilayer structure as a function of energy C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nlayers,npts,aoe,i,j
        complex*16 cd_index(nlayers+1,npts), cd_r, cd_x_zero, cd_x_top
     1          , cd_thet1, cd_thet2, cd_exp_phi
        real*8 four_pi_on_lam, cos_sqr_th,sinthe
        real*4 refl(npts),energy(npts),theta(1),thick(nlayers*2)
        parameter (four_pi_hc= 1.013536e-3) ! =4*pi/hc
        common/points/npts,nlayers,aoe
        implicit none


C executable code:

        cos_sqr_th= cos( theta(1))**2
        sinthe= sin( theta(1))

        do i=1, npts
            four_pi_on_lam= four_pi_hc* energy(i)
            cd_x_zero= (0., 0.) ! reflected amplitude=0 at substrate
            cd_x_top= (0., 0.)
            do j=1,nlayers
                ! njsinthj= sqrt(nj^2-cos^2(th_inc))
                cd_thet2= cdsqrt( cd_index(j,i)**2- cos_sqr_th)
                if(dimag(cd_thet2) .gt. 0.) cd_thet2= dconjg(cd_thet2)
                cd_thet1= cdsqrt( cd_index(j+1,i)**2-cos_sqr_th)
                cd_r = (cd_thet1- cd_thet2)/( cd_thet1+ cd_thet2)
                ! include debye-waller roughness term
                cd_r= cd_r*exp(- (four_pi_on_lam*
     1                          thick(j+nlayers)*sinthe)**2)
                cd_exp_phi=cdexp( (0.,-1.d0)*
     1                  four_pi_on_lam* thick(j)* cd_thet2)
                cd_x_top= ( cd_r+ cd_x_zero*cd_exp_phi)
     1                  /( 1.+ cd_r* cd_x_zero*cd_exp_phi)
                cd_x_zero= cd_x_top

            end do
            refl(i) = cdabs(cd_x_top)**2
         end do
         return
         end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

         subroutine rddat(datname,npts,xval,yval)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C        Reads the data file                                               C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

         real*4 xval(npts),yval(npts)
         integer*4 npts,nskip,erval,i,n,fgetc
         character*80 datname
         character char
         parameter (nskip=4)
         implicit none

! Executable code:
         type  *,'npts in rddat',npts
         n=80
         call blank(datname,n)
         open(unit=3,file=datname,status='old',access='sequential',
      1        form='formatted')
         i=0
         do while (i.lt.nskip)          ! Skip over nskip (text) lines
            erval=fgetc(3,char)
            if  (char.eq.'\n') then     ! If char = return-character
               i=i+1
            end if
         end do
         type  *,'npts in rddat(1)',npts
         do i=1,npts
             read (3,*) xval(i),yval(i)
             type *,xval(i),yval(i),npts
         end do

         close(3)

         return
         end

!*************************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

         subroutine rdfile(inname,indat,outdat,outlol,anoren,mincontrol,
      1  maxcontrol,nsteps,eng,thetmin,thetmax,dthet,angdep,the,
      1  engmin,engmax,deng,nlayers,allortwo,sym,rho,
      1  thick,minthick,maxthick,deltathick,npts,consskip)

         character*80 inname,outdat,indat,outlol,sym(3001)
         character*72 text
         character*5 anoren,angdep,allortwo
         real*4 eng,thetmin,thetmax,dthet,the,engmin,engmax,deng,
      1  thick(2*3001),
      1  rho(3001),mincontrol,maxcontrol,
      1  minthick(2*3001),maxthick(2*3001),deltathick(2*3001)
         integer*4 nlayers,nsteps,npts,n,i,consskip
         implicit none


! Executable code:
         open(unit=2,file=inname,status='old',
      1          access='sequential',form='formatted')
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(20x,a80)') indat
                 read(2,'(21x,a80)') outdat
                 read(2,'(18x,a80)') outlol
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(13x,a3)') anoren
                 read(2,'(a72)') text
                 read(2,'(39x,f5.1)') maxcontrol
                 read(2,'(37x,f5.1)') mincontrol
                 read(2,'(21x,i8)') nsteps
                 read(2,'(21x,i8)') npts
                 read(2,'(a72)') text
                 read(2,'(11x,f8.1)') eng
                 read(2,'(23x,f5.1)') thetmin
                 read(2,'(23x,f5.1)') thetmax
                 read(2,'(25x,f8.1)') dthet
                 read(2,'(a72)') text
                 read(2,'(48x,a3)') angdep
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(15x,f5.1)') the
                 read(2,'(19x,f8.1)') engmin
                 read(2,'(19x,f8.1)') engmax
                 read(2,'(21x,f5.1)') deng
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(35x,i5)')  nlayers
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(25x,a5)') allortwo
                 read(2,'(a72)') text
                 read(2,'(a72)') text
                 read(2,'(a72)') text
         n=3
         call blank(allortwo,n)
         if (allortwo .eq. '2  ') then
         do i=1,3
            read(2,'(a15,f8.1,f8.1,f8.1,f8.1,f8.1,f8.1
      1  ,f8.1,f8.1,f8.1)') sym(i),rho(i),thick(i+nlayers),
      1     minthick(i+nlayers),maxthick(i+nlayers),
      1     deltathick(i+nlayers),thick(i),minthick(i),
      1     maxthick(i),deltathick(i)
         end do
         do i=4,nlayers,2
              sym(i)=sym(2)
              rho(i)=rho(2)
              thick(i+nlayers)=thick(2+nlayers)
              minthick(i+nlayers)=minthick(2+nlayers)
              maxthick(i+nlayers)=maxthick(2+nlayers)
              deltathick(i+nlayers)=deltathick(2+nlayers)
              thick(i)=thick(2)
              minthick(i)=minthick(2)
              maxthick(i)=maxthick(2)
              deltathick(i)=deltathick(2)
              sym(i+1)=sym(3)
              rho(i+1)=rho(3)
              thick(i+1+nlayers)=thick(3+nlayers)
              minthick(i+1+nlayers)=minthick(3+nlayers)
              maxthick(i+1+nlayers)=maxthick(3+nlayers)
              deltathick(i+1)=deltathick(3)
              thick(i+1)=thick(3)
              minthick(i+1)=minthick(3)
              maxthick(i+1)=maxthick(3)
              deltathick(i+1)=deltathick(3)
         end do
         end if
         if (allortwo .eq. 'al ') then
         do i=1,nlayers
              read(2,'(a15,f8.1,f8.1,f8.1,f8.1,f8.1,f8.1
      1  ,f8.1,f8.1,f8.1)') sym(i),rho(i),thick(i+nlayers),
      1  minthick(i+nlayers),maxthick(i+nlayers),
      1  deltathick(i+nlayers),thick(i),minthick(i),
      1  maxthick(i),deltathick(i)
              n=15
              call blank(sym(i),n)
              type *,thick(i),thick(i+nlayers)
         end do
         end if
         read(2,'(a72)') text
         read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(40x,i8)') consskip
      read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(a72)') text

      close(2)
      end

!*****************************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

      subroutine wrtfile(outname,npts,xval,yval,y2val,title,xleg,
     1           yleg,nlayers,thick)

      real*4 xval(2001),yval(2001),y2val(2001),thick(3001)
      integer*4 npts,nlayers,n
      character*80 title,xleg,yleg
      character*80 outname

      n=80
      call blank(outname,n)
      call blank(title,n)
      call blank(xleg,n)
      call blank(xleg,n)

      open(unit=3,file=outname,status='unknown',access='sequential',
     1      form='formatted')


      write (3,'(a)') title
      do i=1,nlayers
          write (3,*),i,thick(i)
      end do
      write (3,'(a)') xleg
      write (3,'(a)') yleg
      write (3,'(i6)') npts

      do i=1,npts
          write (3,'(3(1pg14.7))'),xval(i),yval(i),y2val(i)
      end do
      close(3)

      end
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        program sasil0
C       ==============
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Simulated Annealing Multilayer program, ensembles, thermodyn cooling C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C The program performs simulated annealing on a multilayer structure in
C order to optimize the fit to a reflectivity curve. A number of
C subroutines are used:
C
C anneal        Performs the annealing using the subroutines
C rdfile        Reads the input file
C rddata        Reads the data file (experimental reflectivity curve)
C optconan      Finds optical constants, if angle is variable
C optconen      Finds optical constants, if energy is variable
C reflecan      Calculates the reflectivity of a multilayer structure
C reflecen      Same; for energy as variable
C cost          Calculates the cost function of a reflectivity curve
C neighbour     Randomly chooses a neighbour to the current solution
C accept        Decides whether or not to accept the new solution (cost)
C cool          Lowers the temperature thermodynamically
C wrtfile       Writes the result to a file
C step_ensemble One Metropolis step per ensemble member
C
C The optical method is based on an article by:
C   W.J. Bartels, J. Hornstra and D.J.W. Lobeek, Acta. Cryst. A42, 539 (1986)
C
C The simulated annealing is based on a report by:
C   E. Schroder, S. Friis-Jensen, Physics Laboratory, U. of Copenhagen (1989)
C
C Elsebeth Schroder, Peter H0gh0j, ESRF, June-92
C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

        integer*4 nlayers,npts,size1,size2,byt1,byt2,nsteps,malloc,n,
     1            aoe,enssize,size3,consskip
        character*80 inname,indat,outdat,outlol,sym(3001)
        character*20 anoren,angdep,allortwo
        real*4 thick(2*3001),
     1  rho(3001),thetmin,thetmax,dthet,engmin,
     1  engmax,deng,eng,the,mincontrol,maxcontrol,minthick(2*3001),
     1  maxthick(2*3001),deltathick(2*3001),bestthick(2*3001),
     1  thetamean,v
        real*4 delta,beta,theta,calcrefl,exprefl,bestrefl,energy,
     1         ensemble
        complex*16 cd_index
        pointer (c,cd_index) ,(d,delta),(b,beta),(t,theta),(r1,calcrefl)
        pointer  (r2,exprefl),(r3,bestrefl),(e,energy),(s,ensemble)
        parameter (byt1=4,byt2=16)
        data n /3/
        integer status
        integer Expg_st_good
        external Expg_st_good
        Implicit None
        real x_coordinates(3001,5),y_coordinates(3001,5)
        common/points/npts,nlayers,aoe,enssize

C       Executable code:

        write(*,'(''Enter input file name: '')')
        read(*,'(a50)') inname

        call rdfile(inname,indat,outdat,outlol,anoren,mincontrol,
     1  maxcontrol,nsteps,eng,thetmin,thetmax,dthet,angdep,the,
     1  engmin,engmax,deng,nlayers,allortwo,sym,rho,
     1  thick,minthick,maxthick,deltathick,npts,consskip,v,enssize)

C       Graphics
        status=Expg_st_good()
        call EXPG_GR_OPEN_GRAPHICS(status)
!        call EXPG_GR_SET_XLABELSTYLE(.true.,1,3,1.5,1.5,1.0,status)
        call EXPG_GR_SET_CURVELINE(2,1,7,2.0,.false.,0,status)
        call EXPG_GR_SET_CURVELINE(3,1,5,2.0,.false.,0,status)
        call EXPG_GR_SET_CURVESTYLE(1,.true.,.false.,.false.,status)
        call EXPG_GR_SET_CURVESTYLE(2,.true.,.false.,.false.,status)
        call EXPG_GR_SET_CURVESTYLE(3,.true.,.false.,.false.,status)
```

```
        call blank(anoren,n)
        if (anoren .eq. 'a  ') then
            aoe= 1
!             npts = int((thetmax-thetmin)/dthet+1.+1.0e-15)
            size1=npts*byt1
            size2=npts*(nlayers+1)*byt2
            size3=2*nlayers*enssize*byt1
            b=malloc(size1)
            d=malloc(size1)
            c=malloc(size2)
            e=malloc(byt1)
            r1=malloc(size1)
            r2=malloc(size1)
            r3=malloc(size1)
            t=malloc(size1)
            s=malloc(size3)
            energy=eng
            thetamean=(thetmax+thetmin)/2
        else if (anoren .eq. 'e  ') then
            aoe=-1
!           npts = int((engmax-engmin)/deng+1.+1.0e-15)
            size1=npts*byt1
            size2=npts*(nlayers+1)*byt2
            size3=2*nlayers*enssize*byt1
            b=malloc(size1)
            d=malloc(size1)
            c=malloc(size2)
            e=malloc(size1)
            r1=malloc(size1)
            r2=malloc(size1)
            r3=malloc(size1)
            t=malloc(byt1)
            s=malloc(size3)
            theta=the
        else
           type *,'Error in file',inname
           type *,'- neither angle (a) nor energy (e) to vary'
           goto 3
        end if

        call anneal(inname,indat,outdat,outlol,mincontrol,maxcontrol,
     1  nsteps,energy,angdep,sym,rho,thetamean,
     1  thick,minthick,maxthick,
     1  deltathick,cd_index,delta,beta,theta,ensemble,
     1  calcrefl,exprefl,bestrefl,bestthick,v,status,
     1  consskip,x_coordinates,y_coordinates)

3       call free (b)
        call free (d)
        call free (c)
        call free (e)
        call free (r1)
        call free (r2)
        call free (r3)
        call free (t)
        call free (s)

C       Graphics
        write(*,*)
        call EXPG_GR_OUT_CGM (status)
        write(*,*)
        write(*,*) ' <return> to end program'
        read(*,*)
        call EXPG_GR_CLOSE_GRAPHICS(status)
        if (status .ne. Expg_st_good()) then
           call EXPG_ST_OUT(status)
        end if

        end
C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*123
C Jan-92

        subroutine anneal(inname,indat,outdat,outlol,mincontrol,
     1  maxcontrol,nsteps,energy,angdep,sym,rho,thetamean,
     1  thick,minthick,maxthick,
     1  deltathick,cd_index,delta,beta,theta,ensemble,
     1  calcrefl,exprefl,bestrefl,bestthick,v,status,
     1  consskip,x_coordinates,y_coordinates)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       Performs the annealing using the subroutines                C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nsteps,nlayers,npts,i,j,bestj,consskip,k,
     1          init,aoe,n,enssize,ens_index,best_index,
     1          status,start_coordinates(5),end_coordinates(5)
        character*80 inname,indat,outdat,outlol,sym(nlayers),xleg,yleg
        character*20 angdep
        character*6 x_label
        complex*16 cd_index(nlayers+1,npts)
        real*4 control,mincontrol,maxcontrol,thetamean,
     1      thick(2*nlayers),minthick(2*nlayers),maxthick(2*nlayers),
     1      deltathick(2*nlayers),newthick(2*3001),
     1      bestthick(2*nlayers),
     1      energy(((aoe-1)*npts+(aoe+1))/2),!()=1 if ang,=npts if endep
     1      theta(((aoe+1)*npts+(aoe-1))/2),!()=1 if endep, =npts if ang
     1      delta( ((aoe+1)*npts+(aoe-1))/2,((aoe-1)*npts+(aoe+1))/2 ),
     1      beta( ((aoe+1)*npts+(aoe-1))/2 ,((aoe-1)*npts+(aoe+1))/2 ),
     1      rho(nlayers),ensemble(enssize,2*nlayers),
     1      cost(1000),bestcost,exprefl(npts),calcrefl(npts),
     1      bestrefl(npts),mint,maxt,delt,v,eqcost,meancost,variance,
     1      oldmeancost,
     1      rate,uprate,time,secnds,ran,random,moment_2,sum1,sum2,
     1      x_coordinates(npts,5), y_coordinates(npts,5)
        parameter (degrad=0.01745329)
        implicit none
        common/points/npts,nlayers,aoe,enssize

C Executable code:
        time=secnds(0.)
        init=nint(time/2)*2+1
        type *,'initvalue  ',init
        do i=1,nlayers*2
           bestthick(i)=thick(i)
        end do

        if (aoe.eq.1) then
           call rddat(indat,npts,theta,exprefl)
           call optconan(npts,energy,theta,nlayers,sym,rho,cd_index,
     1         delta,beta,angdep,thetamean)
           call reflecan(bestrefl,energy,theta,bestthick,cd_index)
        else
           call rddat(indat,npts,energy,exprefl)
           call optconen(npts,energy,theta,nlayers,sym,rho,cd_index,
     1         delta,beta)
           call reflecen(bestrefl,energy,theta,bestthick,cd_index)
        end if
        call cost1(exprefl,bestrefl,npts,bestcost)

C*******Prepare ensemble and costs
        do i=1,2*nlayers
           ensemble(1,i)=thick(i)
        end do
        cost(1)=bestcost
        do ens_index=2,enssize
           do i=1,2*nlayers
              random=ran(init)
              maxt=maxthick(i)
              mint=minthick(i)
              delt=deltathick(i)
              thick(i)=mint+int(random*(1+(maxt-mint)/delt))*delt
              ensemble(ens_index,i)=thick(i)
           end do
           if (aoe.eq.1) then
              call reflecan(calcrefl,energy,theta,thick,cd_index)
           else
              call reflecen(calcrefl,energy,theta,thick,cd_index)
           end if
           call cost1(exprefl,calcrefl,npts,cost(ens_index))
        end do

C*******Find equilibrium cost value at "temperature" maxcontrol
        sum1=0
        do ens_index=1,enssize
           sum1=sum1+cost(ens_index)
        end do
        oldmeancost=sum1/enssize
        do i=1,10
           call step_ensemble(ensemble,minthick,maxthick,deltathick,
     1         bestthick,energy,theta,cd_index,exprefl,bestrefl,calcrefl,
     1         cost,bestcost,maxcontrol,0,bestj,rate,uprate,init)
        end do
        sum1=0
        sum2=0
        do ens_index=1,enssize
           sum1=sum1+cost(ens_index)
           sum2=sum2+cost(ens_index)**2
        end do
        meancost=sum1/enssize
        moment_2=sum2/enssize
        variance=moment_2-meancost**2
        type *,'mean',oldmeancost
        type *,'mean',meancost,' variance',variance
        type *,'start finding eqcost'
        do k=1,100
           if ((oldmeancost-meancost)**2 .gt. v*v*variance/100.) then
              oldmeancost=meancost
              do i=1,10
                 call step_ensemble(ensemble,minthick,maxthick,
     1                deltathick,bestthick,energy,theta,cd_index,
     1                exprefl,bestrefl,calcrefl,
     1                cost,bestcost,maxcontrol,0,bestj,rate,uprate,init)
              end do
              sum1=0
              sum2=0
              do ens_index=1,enssize
                 sum1=sum1+cost(ens_index)
                 sum2=sum2+cost(ens_index)**2
              end do
              meancost=sum1/enssize
              moment_2=sum2/enssize
              variance=moment_2 - meancost**2
              type *,'mean',meancost,' variance',variance
           else
              type *,'eqcost found after',k*10,' steps per member'
              goto 7
           end if
        end do
        type *,'eqcost found after 1000 (max) steps per member'
7       eqcost=meancost
        control=maxcontrol
        uprate=0
        rate=0

C       Graphics before annealing
        if (aoe.eq.1) then
           do i=1,npts
              x_coordinates(i,2)=theta(i)/degrad
              x_coordinates(i,1)=theta(i)/degrad
           end do
           x_label='angle '
        else
           do i=1,npts
               x_coordinates(i,2)=energy(i)
               x_coordinates(i,1)=energy(i)
           end do
           x_label='energy'
        end if
        do i=1,2
           start_coordinates(i)=1
           end_coordinates(i)=npts
        end do
        do i=1,npts
           y_coordinates(i,2)=Alog10(exprefl(i))
           y_coordinates(i,1)=Alog10(bestrefl(i))
        end do
        call EXPG_GR_XYGRAPH(npts,5,2,
     1      start_coordinates,end_coordinates,
     1      x_coordinates,y_coordinates,
     1      'ensemble, thermodyn. cooling',x_label,'reflectivity',
     1      status)
```

```
      do i=1,nlayers
         type *,bestthick(i),bestthick(i+nlayers)
      end do
      n=80
      call blank(outlo1,n)
      open(unit=2,file=outlo1,status='unknown',
     1     access='sequential',form='formatted')
      write(2,'(i8)') int(nsteps/consskip/enssize)
C start annealing
      do j=1,int(nsteps/enssize)
         call step_ensemble(ensemble,minthick,maxthick,deltathick,
     1      bestthick,energy,theta,cd_index,exprefl,bestrefl,calcrefl,
     1      cost,bestcost,control,j,bestj,rate,uprate,init)

         if (mod(j,consskip).eq.0) then
C******** Graphics
            best_index=1
            do ens_index=2,enssize
               if (cost(best_index).gt.cost(ens_index)) then
                  best_index=ens_index
               end if
            end do
            do i=1,2*nlayers
               newthick(i)=ensemble(best_index,i)
            end do
            do i=1,3
               start_coordinates(i)=1
               end_coordinates(i)=npts
            end do
            if (aoe.eq.1) then
               do i=1,npts
                  x_coordinates(i,1)=theta(i)/degrad
                  x_coordinates(i,2)=theta(i)/degrad
                  x_coordinates(i,3)=theta(i)/degrad
               end do
               call reflecan(calcrefl,energy,theta,newthick,
     1              cd_index)
            else
               do i=1,npts
                  x_coordinates(i,1)=energy(i)
                  x_coordinates(i,2)=energy(i)
                  x_coordinates(i,3)=energy(i)
               end do
               call reflecen(calcrefl,energy,theta,newthick,
     1              cd_index)
            end if
            do i=1,npts
               y_coordinates(i,1)=Alog10(calcrefl(i))
               y_coordinates(i,2)=Alog10(exprefl(i))
               y_coordinates(i,3)=Alog10(bestrefl(i))
            end do
            call EXPG_GR_XYGRAPH(npts,5,3,
     1           start_coordinates,end_coordinates,
     1           x_coordinates,y_coordinates,
     1           'ensemble, thermodyn. cooling',
     1           x_label,'reflectivity',status)

            rate=rate/consskip/enssize
            uprate=uprate/consskip/enssize

            write(*,*)
            write(*,'(x,a9,4x,a9,4x,a4,6x,a7,4x,a6)') 'iter. no.',
     1           'best iter.','rate','control','uprate'
            write(*,'(x,i8,5x,i8,3x,g10.5,2x,g10.5,2x,g10.5)')
     1           j*enssize,bestj*enssize,rate,control,uprate
            write(*,'(x,a15,x,a19,x,a8,x,a8)') '    bestcost     ',
     1           'bestcost in ensemb.','meancost','variance'
            write(*,'(x,g15.7,3x,g15.7,4x,g15.7,5x,g15.7)') bestcost,
     1           cost(best_index),meancost,variance
            write(2,'(a9,2x,a9,3x,a4,3x,a7,3x,a6)') 'iter. no.',
     1           'best iter.','rate','control','uprate'
            write(2,'(x,i8,3x,i8,x,g10.5,g10.5,x,g10.5)') j*enssize,
     1          bestj*enssize,rate,control,uprate
            write(2,'(x,a15,x,a19)') '    bestcost     ',
     1           'bestcost in ensemb.'
            write(2,'(x,g15.7,3x,g15.7)') bestcost,cost(best_index)
            write(*,*)
```

```
            write(*,'(13x,a13,6x,a15)') 'best till now',
     1           'best in ensemb.'
            write(*,'(a9,3x,4(a5,5x))') 'layer no.','thick','rough',
     1            'thick','rough'

            do i=1,nlayers
               write(*,'(i8,2x,4(g10.5,g10.5))') i,bestthick(i),
     1         bestthick(i+nlayers),newthick(i),newthick(i+nlayers)
            end do
            rate=0
            uprate=0
         endif
         call cool(control,cost,eqcost,v,meancost,variance)
      end do
      close(2)
      if (aoe.eq.1) then
         xleg='angle (degrees)'
         yleg='reflectivity exp  calc'
         call wrtfile(outdat,npts,theta,exprefl,bestrefl,inname,
     1        xleg,yleg,nlayers,bestthick)
      else
         xleg='energy (eV)'
         yleg='reflectivity exp  calc'
         call wrtfile(outdat,npts,energy,exprefl,bestrefl,inname,
     1        xleg,yleg,nlayers,bestthick)
      end if

      return
      end

C**********************************************************************
C June 92

      subroutine step_ensemble(ensemble,minthick,maxthick,deltathick,
     1      bestthick,energy,theta,cd_index,exprefl,bestrefl,calcrefl,
     1      cost,bestcost,control,j,bestj,rate,uprate,init)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     Takes one step for each member of the ensemble                   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      integer npts,nlayers,aoe,enssize,ens_index,init,i,good,upgood
      real*4 ensemble(enssize,2*nlayers),minthick(2*nlayers),
     1    maxthick(2*nlayers),deltathick(2*nlayers),
     1    bestthick(2*nlayers),newthick(2*3001),
     1    energy(((aoe-1)*npts+(aoe+1))/2),!()=1 if ang,=npts if endep
     1    theta(((aoe+1)*npts+(aoe-1))/2),!()=1 if endep, =npts if ang
     1    exprefl(npts),calcrefl(npts),bestrefl(npts),control,
     1    cost(enssize),bestcost,newcost,j,bestj,rate,uprate

      complex*16 cd_index(nlayers+1,npts)

      common/points/npts,nlayers,aoe,enssize
      implicit none

C Executable code:
      do ens_index=1,enssize
         do i=1,2*nlayers
            newthick(i)=ensemble(ens_index,i)
         end do
         call neighbour(newthick,minthick,maxthick,deltathick,
     1                 nlayers,init)
         if (aoe.eq.1) then
            call reflecan(calcrefl,energy,theta,newthick,
     1           cd_index)
         else
            call reflecen(calcrefl,energy,theta,newthick,
     1           cd_index)
         end if
         call cost1(exprefl,calcrefl,npts,newcost)
         call accept(cost(ens_index),newcost,control,good,
     1                upgood,init)
         if (good .eq. 1) then
            do i=1,2*nlayers
               ensemble(ens_index,i)=newthick(i)
            end do
            cost(ens_index)=newcost
            if (newcost .lt. bestcost) then
```

```
                bestcost=newcost
                bestj=j
                do i=1,npts
                   bestrefl(i)=calcrefl(i)
                end do
                do i=1,nlayers*2
                   bestthick(i)=newthick(i)
                end do
             endif
          endif
          rate=rate+good
          uprate=uprate+upgood
       end do


       return
       end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

       subroutine cool(control,cost,eqcost,v,meancost,variance)
C      ====================================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Lowers the control parameter (temperature) according to costs  C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       integer npts,nlayers,aoe,enssize,ens_index
       real*4 cost(enssize),v,meancost,moment_2,variance,eqcost,
     1         neweqcost,control,sum1,sum2,oldvariance
       common/points/npts,nlayers,aoe,enssize
       implicit none

C Executable code:
       sum1=0.
       sum2=0.
       oldvariance=variance
       do ens_index=1,enssize
          sum1=sum1+cost(ens_index)
          sum2=sum2+cost(ens_index)**2
       end do
       meancost=sum1/enssize
       moment_2=sum2/enssize
       variance=moment_2-meancost**2
       neweqcost=meancost-v*sqrt(variance)
       control=control*(1+(neweqcost-eqcost)*control/oldvariance)
       eqcost=neweqcost
       if (control .lt. 1e-10) then
          control=1e-10
       end if
       return
       end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

       subroutine accept(oldcost,newcost,control,good,upgood,init)
C      =======================================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Decides whether to accept neighbour to the current solution   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       integer good,init,upgood
       real*4 oldcost, newcost,control,deltacost,prob,random,ran
       implicit none

C Executable code:

       deltacost=newcost-oldcost
       if (deltacost .le. 0) then
          good=1
          upgood=0
       else
          random=ran(init)
          prob=exp(-deltacost/control)-random
```

```
          if (prob .ge. 0) then
             good=1
             upgood=1
          else
             good=0
             upgood=0
          endif
       end if

       return
       end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92
       subroutine neighbour(newthick,minthick,maxthick,
     1           deltathick,nlayers,init)


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Randomly chooses a neighbour to the current solution         C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       integer nlayers,changel,init
       real*4 minthick(2*nlayers),maxthick(2*nlayers),
     1        deltathick(2*nlayers),addthick,random,userough,
     1        newthick(2*nlayers),ran
       parameter (userough=0.0)
       implicit none

C Executable code
       random=ran(init)
       changel=int(random*nlayers-1e-15)+1
       random=ran(init)
       if (random .ge. userough) then
           random=ran(init)
           addthick=(int(random*2-1e-15)*2-1)*deltathick(changel)
           newthick(changel)=newthick(changel)+addthick
           if (newthick(changel) .lt. minthick(changel)) then
              newthick(changel)=maxthick(changel)
           else if (newthick(changel) .gt. maxthick(changel)) then
              newthick(changel)=minthick(changel)
           endif
       else
           random=ran(init)
           addthick=(int(random*2-1e-15)*2-1)*
     1                       deltathick(changel+nlayers)
           newthick(changel+nlayers)=newthick(changel+nlayers)+addthick
           if (newthick(changel+nlayers) .lt.
     1                         minthick(changel+nlayers)) then
               newthick(changel+nlayers)=maxthick(changel+nlayers)
           else if (newthick(changel+nlayers) .gt.
     1                         maxthick(changel+nlayers)) then
               newthick(changel+nlayers)=minthick(changel+nlayers)
           endif
       end if

       return
       end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

       subroutine cost1(expdat,calcdat,npts,kost)
C      ========================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       integer npts,i
       real*4 expdat(2001),calcdat(2001), kost
       implicit none

C Executable code:
        kost=0
        do i=1,npts
            kost=kost+(1-(calcdat(i)/expdat(i)))**2
     1              +(1-(expdat(i)/calcdat(i)))**2
        end do
        kost=kost/npts

        return
        end

C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine cost2(expdat,calcdat,npts,kost)
C       ============================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer npts,i
        real*4 expdat(2001),calcdat(2001), kost
        implicit none

C Executable code:

        kost=0
        do i=1,npts
            kost=kost+(1-(calcdat(i)/expdat(i)))**2
        end do
        kost=kost/npts

        return
        end


C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine cost4(expdat,calcdat,npts,kost)
C       ============================================
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   Calculates the cost function of a calculated reflectivity curve     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer npts,i
        real*4 expdat(2001),calcdat(2001), kost
        implicit none

C Executable code:

        kost=0
        do i=1,npts
            kost=kost+log(1+(1-(calcdat(i)/expdat(i)))**2
     1              +(1-(expdat(i)/calcdat(i)))**2)
        end do
        kost=kost/npts

        return
        end

C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine optconan(npts,energy,theta,nlayers,sym,rho,cd_index,
     1  delta,beta,angdep,thetamean)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Finds optical constants for each layer and for each angle        C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        character*20 unit,angdep
        integer nlayers,npts,i,n,k,j
        character*80 sym(3001)
        complex*16 cd_index(nlayers+1,npts)
        real*4 delta(npts,1),beta(npts,1),
     1         energy(1),theta(npts),rho(3001),thetamean
        parameter (degrad=0.01745329)
        implicit none

C Executable code
        unit= 'ev'
        do i=1,npts
            theta(i)=theta(i)*degrad
        end do
        n=3
        call blank(angdep,n)
        if (angdep .eq. 'y  ') then
        type *,sym(1)
        call f12(npts,1,sym(1),energy,unit,theta,delta,beta,rho(1))
!       type *,' called f12 (1)'
        do i=1,npts
            cd_index(1,i)= dcmplx(1-delta(i,1),-beta(i,1))
            cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
            do k=1,j-1
                if (sym(j) .eq. sym(k)) then
                do i=1,npts
                    cd_index(j,i)=cd_index(k,i)
!                   type *,cd_index(j,i)
                end do
                go to 111
                end if
            end do
            call f12(npts,1,sym(j),energy,unit,theta,delta,beta,rho(j))
!           type *,' called f12'
            do i=1,npts
                cd_index(j,i)= dcmplx(1-delta(i,1),-beta(i,1))
!               type *,cd_index(j,i)
            end do
            go to 111
111     end do
!       type *,'Found optical constants'
        end if

        if (angdep .eq. 'n  ') then
        type *,'thetamean=',thetamean
        call f12(1,1,sym(1),energy,unit,thetamean,delta,beta,rho(1))
!       type *,' called f12 (1)'
        do i=1,npts
            cd_index(1,i)= dcmplx(1-delta(1,1),-beta(1,1))
            cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
            do k=1,j-1
                if (sym(j) .eq. sym(k)) then
                do i=1,npts
                    cd_index(j,i)=cd_index(k,i)
!                   type *,cd_index(j,i)
                end do
                go to 112
                end if
            end do
            call f12(1,1,sym(j),energy,unit,thetamean,delta,beta,rho(j))
!           type *,' called f12'
            do i=1,npts
                cd_index(j,i)= dcmplx(1-delta(1,1),-beta(1,1))
!               type *,cd_index(j,i)
            end do
            go to 112
112     end do
!       type *,'Found optical constants'
        end if

        return
        end

C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

        subroutine reflecan(refl,energy,theta,thick,cd_index)
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Calculates reflectivity of a multilayer structure as a function of angle C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        integer nlayers,npts,aoe,i,j,enssize
        complex*16 cd_index(nlayers+1,npts), cd_r, cd_x_zero, cd_x_top
     1          , cd_thet1, cd_thet2, cd_exp_phi
        real*8 four_pi_on_lam,cos_sqr_th,sinthe
        real*4 refl(npts),energy(1),theta(npts),thick(nlayers*2)
        parameter (four_pi_hc= 1.013536e-3) ! =4*pi/hc
        common/points/npts,nlayers,aoe,enssize
        implicit none

C Executable code:
        four_pi_on_lam= four_pi_hc* energy(1)
        do i=1, npts
            cos_sqr_th= cos( theta(i))**2
            sinthe= sin( theta(i))
            cd_x_zero= (0., 0.) ! reflected amplitude=0 at substrate
            cd_x_top= (0., 0.)
            do j=1,nlayers
                ! njsinthj= sqrt(nj^2-cos^2(th_inc))
                cd_thet2= cdsqrt( cd_index(j,i)**2- cos_sqr_th)
                if(dimag(cd_thet2) .gt. 0.) cd_thet2= dconjg(cd_thet2)
                cd_thet1= cdsqrt( cd_index(j+1,i)**2-cos_sqr_th)
                cd_r = (cd_thet1- cd_thet2)/( cd_thet1+ cd_thet2)
                ! include debye-waller roughness term
             cd_r=cd_r*exp(-(four_pi_on_lam*thick(j+nlayers)*sinthe)**2)
                cd_exp_phi=cdexp( (0.,-1.d0)*
     1                  four_pi_on_lam* thick(j)* cd_thet2)
                cd_x_top= ( cd_r+ cd_x_zero*cd_exp_phi)
     1                  /( 1.+ cd_r* cd_x_zero*cd_exp_phi)
                cd_x_zero= cd_x_top

            end do
            refl(i) = cdabs(cd_x_top)**2
        end do
        return
        end

!**************************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
! sept-91

        subroutine optconen(npts,energy,theta,nlayers,sym,rho,cd_index,
     1  delta,beta)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Finds optical constants for each layer and for each energy     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        character*20 unit
        integer nlayers,npts,i,k,j
        character*80 sym(3001)
        complex*16 cd_index(nlayers+1,npts)
        real*4 delta(1,npts),beta(1,npts),energy(npts)
     1  ,theta(1),rho(3001),degrad
        parameter (degrad=0.01745329)
        implicit none

C executable code
        theta(1)=theta(1)*degrad
        unit= 'ev'
        call f12(1,npts,sym(1),energy,unit,theta,delta,beta,rho(1))
!       type *,' called f12 (1)'
        do i=1,npts
            cd_index(1,i)= dcmplx(1-delta(1,i),-beta(1,i))
            cd_index(nlayers+1,i) =(1.,0.)
        end do
        do j=2,nlayers
            do k=1,j-1
                if (sym(j) .eq. sym(k)) then
                do i=1,npts
                    cd_index(j,i)=cd_index(k,i)
!                   type *,cd_index(j,i)
                end do
                    go to 111
                    end if
                end do
            call f12(1,npts,sym(j),energy,unit,theta,delta,beta,rho(j))
!               type *,' called f12'
                do i=1,npts
                    cd_index(j,i)= dcmplx(1-delta(1,i),-beta(1,i))
!                   type *,cd_index(j,i)
                end do
                go to 111
111         end do
!           type *,'Found optical constants'

            return
            end

C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

            subroutine reflecen(refl,energy,theta,thick,cd_index)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Calculates reflectivity of a multilayer structure as a function of energy C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

            integer nlayers,npts,aoe,i,j,enssize
            complex*16 cd_index(nlayers+1,npts), cd_r, cd_x_zero, cd_x_top
        1           , cd_thet1, cd_thet2, cd_exp_phi
            real*8 four_pi_on_lam, cos_sqr_th,sinthe
            real*4 refl(npts),energy(npts),theta(1),thick(nlayers*2)
            parameter (four_pi_hc= 1.013536e-3) ! =4*pi/hc
            common/points/npts,nlayers,aoe,enssize
            implicit none


C executable code:

            cos_sqr_th= cos( theta(1))**2
            sinthe= sin( theta(1))

            do i=1, npts
                four_pi_on_lam= four_pi_hc* energy(i)
                cd_x_zero= (0., 0.) ! reflected amplitude=0 at substrate
                cd_x_top= (0., 0.)
                do j=1,nlayers
                    ! njsinthj= sqrt(nj^2-cos^2(th_inc))
                    cd_thet2= cdsqrt( cd_index(j,i)**2- cos_sqr_th)
                    if(dimag(cd_thet2) .gt. 0.) cd_thet2= dconjg(cd_thet2)
                    cd_thet1= cdsqrt( cd_index(j+1,i)**2-cos_sqr_th)
                    cd_r = (cd_thet1- cd_thet2)/( cd_thet1+ cd_thet2)
                    ! include debye-waller roughness term
                    cd_r= cd_r*exp(- (four_pi_on_lam*
        1                           thick(j+nlayers)*sinthe)**2)
                    cd_exp_phi=cdexp( (0.,-1.d0)*
        1                   four_pi_on_lam* thick(j)* cd_thet2)
                    cd_x_top= ( cd_r+ cd_x_zero*cd_exp_phi)
        1                   /( 1.+ cd_r* cd_x_zero*cd_exp_phi)
                    cd_x_zero= cd_x_top

                end do

                refl(i) = cdabs(cd_x_top)**2
            end do
            return
            end

C**************************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345
C Jan-92

            subroutine rddat(datname,npts,xval,yval)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       Reads the data file                                              C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

            real*4 xval(npts),yval(npts)
```

```
      integer*4 npts,nskip,erval,i,n,fgetc
      character*80 datname
      character char
      parameter (nskip=4)
      implicit none

! Executable code:
      n=80
      call blank(datname,n)
      open(unit=3,file=datname,status='old',access='sequential',
     1      form='formatted')
      i=0
      do while (i.lt.nskip)         ! Skip over nskip (text) lines
         erval=fgetc(3,char)
         if  (char.eq.'\n') then    ! If char = return-character
            i=i+1
         end if
      end do
      type  *,'npts ',npts
      do i=1,npts
          read (3,*) xval(i),yval(i)
          type *,xval(i),yval(i),npts
      end do

      close(3)

      return
      end

C***********************************************************************
C23456789*123456789*123456789*123456789*123456789*123456789*123456789*12

      subroutine rdfile(inname,indat,outdat,outlo1,anoren,mincontrol,
     1 maxcontrol,nsteps,eng,thetmin,thetmax,dthet,angdep,the,
     1 engmin,engmax,deng,nlayers,allortwo,sym,rho,
     1 thick,minthick,maxthick,deltathick,npts,consskip,v,enssize)

      character*80 inname,outdat,indat,outlo1,sym(3001)
      character*72 text
      character*5 anoren,angdep,allortwo
      real*4 eng,thetmin,thetmax,dthet,the,engmin,engmax,deng,
     1 thick(2*3001),v,
     1 rho(3001),mincontrol,maxcontrol,
     1 minthick(2*3001),maxthick(2*3001),deltathick(2*3001)
      integer*4 nlayers,nsteps,npts,n,i,consskip,enssize
      implicit none


! Executable code:
      open(unit=2,file=inname,status='old',
     1         access='sequential',form='formatted')
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(20x,a80)') indat
               read(2,'(21x,a80)') outdat
               read(2,'(18x,a80)') outlo1
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(13x,a3)') anoren
               read(2,'(a72)') text
               read(2,'(39x,f5.1)') maxcontrol
               read(2,'(37x,f5.1)') mincontrol
               read(2,'(21x,i8)') nsteps
               read(2,'(21x,i8)') npts
               read(2,'(a72)') text
               read(2,'(11x,f8.1)') eng
               read(2,'(23x,f5.1)') thetmin
               read(2,'(23x,f5.1)') thetmax
               read(2,'(25x,f8.1)') dthet
               read(2,'(a72)') text
               read(2,'(48x,a3)') angdep
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(15x,f5.1)') the
               read(2,'(19x,f8.1)') engmin
               read(2,'(19x,f8.1)') engmax
```

```
               read(2,'(21x,f5.1)') deng
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(35x,i5)')  nlayers
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(25x,a5)') allortwo
               read(2,'(a72)') text
               read(2,'(a72)') text
               read(2,'(a72)') text
      n=3
      call blank(allortwo,n)
      if (allortwo .eq. '2  ') then
      do i=1,3
         read(2,'(a15,f8.1,f8.1,f8.1,f8.1,f8.1,f8.1
     1 ,f8.1,f8.1,f8.1)') sym(i),rho(i),thick(i+nlayers),
     1   minthick(i+nlayers),maxthick(i+nlayers),
     1   deltathick(i+nlayers),thick(i),minthick(i),
     1   maxthick(i),deltathick(i)
      end do
      do i=4,nlayers,2
          sym(i)=sym(2)
          rho(i)=rho(2)
          thick(i+nlayers)=thick(2+nlayers)
          minthick(i+nlayers)=minthick(2+nlayers)
          maxthick(i+nlayers)=maxthick(2+nlayers)
          deltathick(i+nlayers)=deltathick(2+nlayers)
          thick(i)=thick(2)
          minthick(i)=minthick(2)
          maxthick(i)=maxthick(2)
          deltathick(i)=deltathick(2)
          sym(i+1)=sym(3)
          rho(i+1)=rho(3)
          thick(i+1+nlayers)=thick(3+nlayers)
          minthick(i+1+nlayers)=minthick(3+nlayers)
          maxthick(i+1+nlayers)=maxthick(3+nlayers)
          deltathick(i+1)=deltathick(3)
          thick(i+1)=thick(3)
          minthick(i+1)=minthick(3)
          maxthick(i+1)=maxthick(3)
          deltathick(i+1)=deltathick(3)
      end do
      end if
      if (allortwo .eq. 'al ') then
      do i=1,nlayers
          read(2,'(a15,f8.1,f8.1,f8.1,f8.1,f8.1,f8.1
     1 ,f8.1,f8.1,f8.1)') sym(i),rho(i),thick(i+nlayers),
     1 minthick(i+nlayers),maxthick(i+nlayers),
     1 deltathick(i+nlayers),thick(i),minthick(i),
     1 maxthick(i),deltathick(i)
          n=15
          call blank(sym(i),n)
          type *,thick(i),thick(i+nlayers)
      end do
      end if
      read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(a72)') text
      read(2,'(40x,i8)') consskip
      read(2,'(35x,i8)') enssize
      read(2,'(25x,f7.3)') v
      read(2,'(a72)') text
      read(2,'(a72)') text
      close(2)
      end

!***********************************************************************
!23456789*123456789*123456789*123456789*123456789*123456789*123456789*12345

      subroutine wrtfile(outname,npts,xval,yval,y2val,title,xleg,
     1           yleg,nlayers,thick)

      real*4 xval(2001),yval(2001),y2val(2001),thick(3001)
      integer*4 npts,nlayers,n
      character*80 title,xleg,yleg
      character*80 outname

      n=80
      call blank(outname,n)
      call blank(title,n)
      call blank(xleg,n)
      call blank(xleg,n)

      open(unit=3,file=outname,status='unknown',access='sequential',
     1      form='formatted')


      write (3,'(a)') title
      do i=1,nlayers
          write (3,*),i,thick(i)
      end do
      write (3,'(a)') xleg
      write (3,'(a)') yleg
      write (3,'(i6)') npts

      do i=1,npts
          write (3,'(3(1pg14.7))'),xval(i),yval(i),y2val(i)
      end do
      close(3)

      end
```