Parratt32
Version 1.5

The Reflectivity Tool

The Manual

current address: ETH Zürich
Laboratorium für Atmosphärenphysik
(LAPETH)
Hönggerberg HPP
8093 Zürich
Schweiz
e–mail: braun@atmos.umnw.ethz.ch

This document was prepared using pdfTEX

Contents

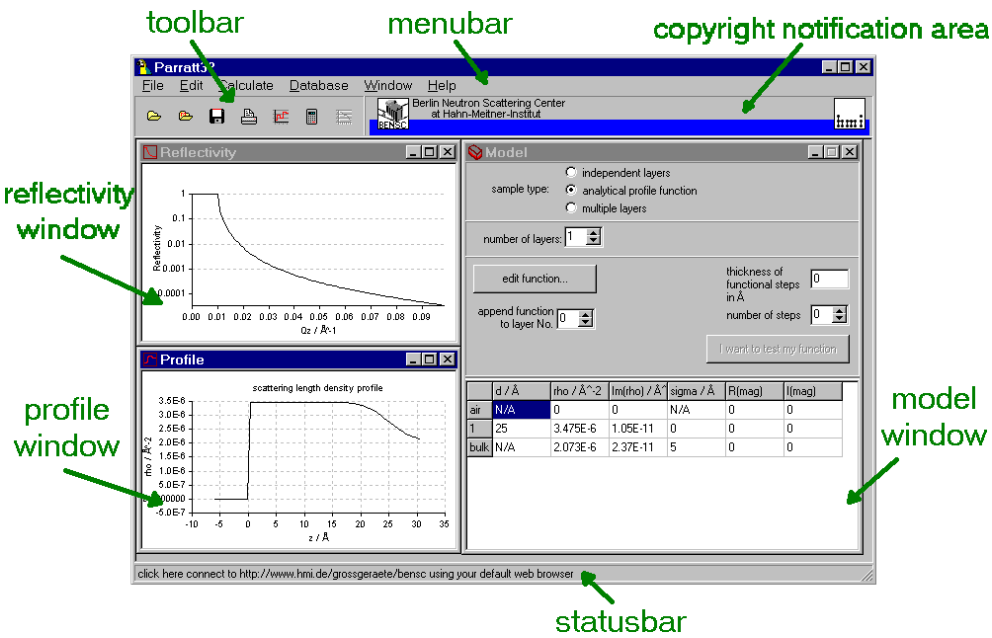
1 Introduction

This manual briefly explains the use of PARRATT32. Most aspects are covered also by the on–line help accessible from within the programme. This document's purpose is to be printed out and to be read separately while or before working with the programme. It also serves as a source of reference in cases where users get stuck with some sort of problem. The second part of the manual describes the vital parts of the source code and gives a brief summary of how to compile it to get the executable. On the screen some of the graphics in this documentation might look ugly, but once printed out the quality increases.

2 The Programme

PARRATT32 is a programme to calculate the optical reflectivity of neutrons or x-rays from flat surfaces. The calculation is based on Parratt's recursion scheme for stratified media [Parratt54]. PARRATT32 is a Microsoft Windows application (either Windows 95, Windows 98 or Windows NT 4.0 — the i386 edition works fine, other platforms, such as FX32 on DEC Alpha could not be tested). The 32 bit extensions for 16bit-Windows (versions 3.1x, also known as Win32s) are not sufficient!

2.1 The Graphical User Interface



PARRATT32's graphical user interface (*GUI* for short, or *main window*) is shown in figure 2.1. The main elements, depicted by the arrows are described in the next sections.

2.1.1 Menu Bar

The Menu Bar.

In the menu bar the different menu entries can be selected with the mouse or with key strokes. Most of the functionality that Parratt32 delivers can be found here.

2.1.2 Tool Bar


The Tool Bar.

Some of the frequently used items from the menu have short–cuts placed in the tool bar as little buttons. This makes accessing these items with the pointing device (i.e. the mouse) very handy.

2.1.3 Reflectivity Window

This window is for displaying the reflectivity (R vs. Q_z) calculated from the given sample. Measured sets of data that were loaded into the programme are displayed here, too.

2.1.4 Profile Window

From the data given in the model window a scattering length density profile can be calculated. The resulting ϱ vs. z is displayed in this window.

2.1.5 Model Window

The model window is the place where the most relevant information of the model sample is stored. Mainly this is a description of how the scattering length density varies with the different layers that make up the sample.

2.1.6 Status Bar

The Status Bar.

Important programme information is displayed here from time to time, i. e. moving the mouse pointer in one of the graphics windows will display the respective coordinates.

2.1.7 Copyright Notification Area

Copyright Notification Area.

This tool bar is just a reminder of where you got the software from and who paid for it...

3 Working with the Programme

3.1 Loading Data

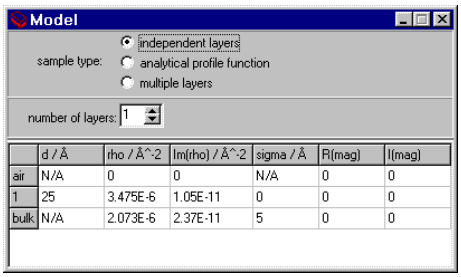
The only kind of data PARRATT32 can handle is ASCII files made up of three or more columns separated by a `TAB` character (ASCII 9). The first column are the Q_z values, the second column are the $R(Q_z)$ values (where $R(0) \equiv 1$) and the third column are $\delta R(Q_z)$. These can be set to zero. Additional columns will be ignored. Since the reflectivity plot is logarithmic, PARRATT32 will ignore negative values in the second column. There is some error handling routine, that lets PARRATT32 ignore certain lines in the file: i. e. header lines, negative R's or other kind of NANs. Once finished loading the data, it will be displayed as small green circles in the reflectivity window.

3.2 Setting up a Model

On start–up PARRATT32 loads a standard set of parameters. Some of these parameters will be stored in the Window's registry when you end working with the programme and will be reloaded the next time you start PARRATT32. In the model window the basic sample start–up parameters are always Si bulk material with a 25 Å SiO_2 layer on top of it (values for 4.66 Å neutrons).
Prior to fitting a model for the layer system has to be set up. There are three basically different sample types in PARRATT32 which will be described in the next sections. For every layer in the model at least two values have to be entered, four more can be entered. The two essential values are: the layer thickness in Ångstrøm (`d/Å`) and the real part of the scattering length density (`rho/Å^-2`). The four optional parameters are the imaginary part of the scattering length density (`Im(rho/Å^-2)`), the roughness of the layer (`sigma/Å`), and the magnetic contribution to the scattering length density (`R(mag)`, `I(mag)`, same units as `rho` and `Im(rho)`).

3.2.1 Simple Models

A simple model is made up of a varying number of essentially different layers. By selecting the radio button labeled "independent layers" the model window rearranges to its most simple form: there is only one edit field for the number of layers and below it is shown a table where one wants to type in the respective properties for the layers. By changing the number of layers in the edit field one can change the number of rows in the table. Up to four hundred layers can make up such a simple sample, but who wants to type in these values...

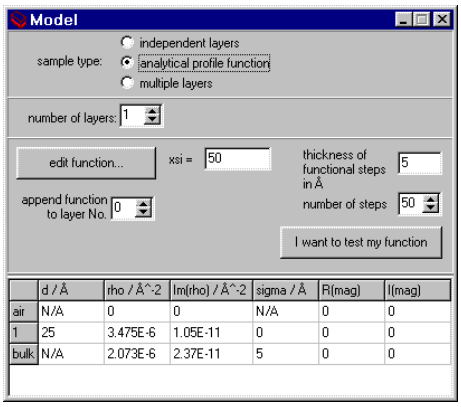


independent layer mode

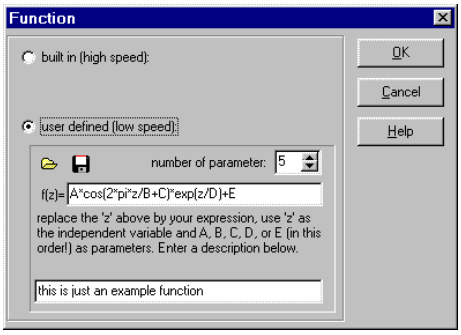
3.2.2 Models With Functions

Sometimes it might be useful to describe the scattering length density profile of interfaces by analytical functions. By pressing the "edit function" button a dialogue shows up where a function can either be chosen from a list of predefined ones or typed as an expression. The latter, so called user–functions, can also be saved to file and reloaded later on. Parratt32 comes with four built–in functions: exponential decay, Liu–Fisher decay [Liu89], oscillating, and oscillating with decaying amplitude.

These function are *hard–coded*, so the evaluation of the scattering length density is fast, and so is the calculation of the reflectivity, well, at least faster than for the user–functions, that run through the function parser each time a value is requested. Apart from choosing the function there are other parameters to be set in the model. Firstly, the layers between which the functional expression is inserted has to be specified. This can be done in the edit field labeled "append function to layer No.". Append means when you choose 1, the function will be appended to layer one, which is between layer one and two! Secondly, the range of the function with respect to the z–coordinate must be given. This is done with the two edit fields "thickness..." and "number of steps", which, by multiplying give the total depth of the region between the chosen layers. For convenience, there is a button labeled "I want to test my function" which calculates the reflec-



profile function mode

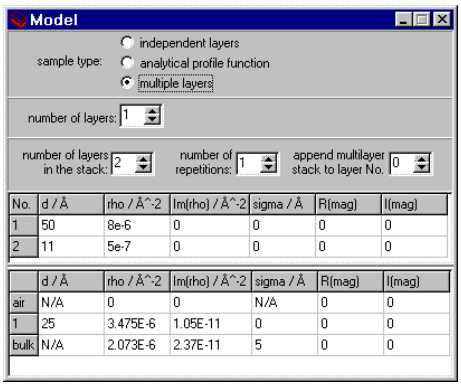


The function edit dialogue

tivity and the profile.

3.2.3 Multi Layer Models

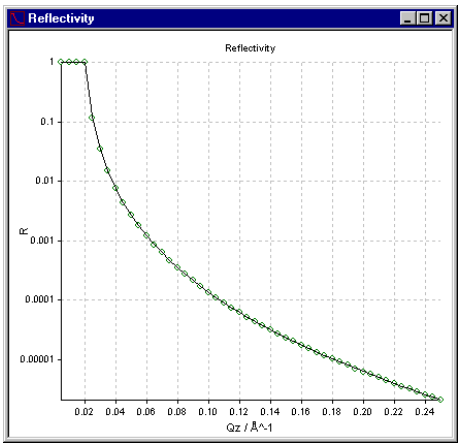
Often samples come with regularly arranged stacks of layers, so called multi–layered systems. To handle these more conveniently the model window can be switch to multi–layer mode. Here a second table shows up: the multi–layer stack. This stack, which behaves in the same way as does the basic sample, can be inserted into the basic sample at an arbitrary position (edit field "append multi layer..."). This means one can have layers of the basic sample above the multi–layer stack and below it. The multi–layer stack itself consists of at least two different materials (edit field "number of layers in the stack"), which can be repeated several times (edit field "number of repetitions"). The total thickness of the sample is then calculated from the thickness of the basic sample plus the thickness of the multi–layer stack multiplied by the number of repetitions.



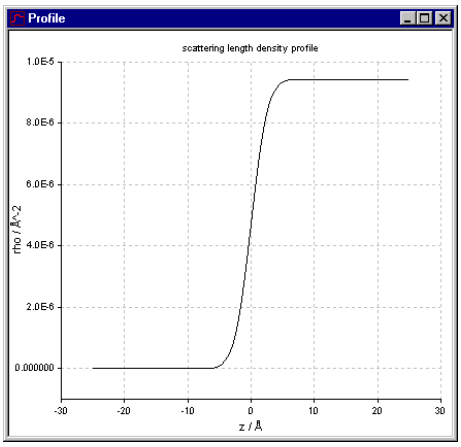
multi–layer mode

3.3 Calculating the Reflectivity

The reflectivity $R(Q_z)$ of the model can be displayed by either selecting "Calculate–Reflectivity" from the main menu or by pressing the calculator button in the tool bar. Note: the tool bar shows so called *tool tips* if the mouse pointer rests over he images for a little while. After a short while — depending of the speed of the computer and the complexity of the model — the calculation will be finished and the reflectivity curve will be displayed in the reflectivity window. The scaling of the graph is done automatically.

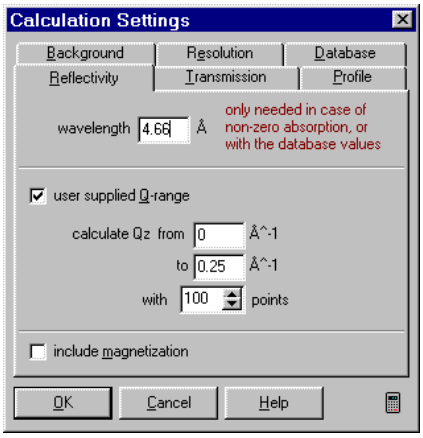


3.4 Calculating SLD Profiles



The scattering length density profile $\varrho(z)$ of the sample can be calculated by either selecting "Calculate–Profile" from the main menu or by pressing the profile button in the tool bar. The progress of the calculation is displayed in the programme's status bar. After the calculation is done the resulting scattering length density profile is displayed in the profile window, which also rescales automatically.
By simultaneously pressing the shift key, the left mouse button, and dragging the mouse a rectangle can be drawn inside the graph windows. This rectangle then defines the new minimal and maximal coordinates of the diagram. For automatic rescaling simply hold shift and press the left mouse button somewhere inside the window, hopefully everything will return to normal.

3.5 Setting Various Calculation Parameters



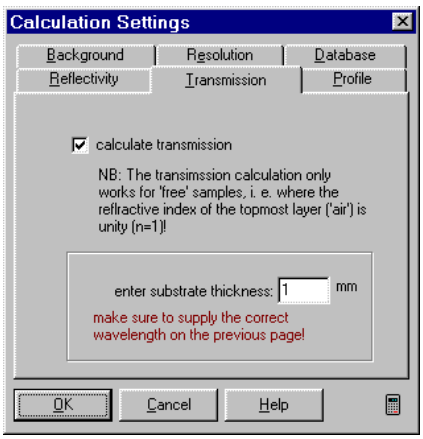
Most calculation relevant settings are accessible via the "Settings Dialogue". It will be shown when selecting "Calculate–Settings" from the main menu. The dialogue box comes with a lot of pages which will be explained in the next sections. On the bottom of the box there are four buttons three of which have the standard meaning ("Ok", "Cancel", and "Help") but the fourth calculates either the reflectivity or the profile depending on what page is selected in the dialogue.

3.5.1 Reflectivity Calculation Settings

On the first page of the dialogue box the settings for reflectivity calculation can be specified. The wavelength edit field is only relevant, when inserting values from the scattering length density database in the model window.
When the chackbox labeled "user supplied Q–range" is checked the reflectivity will be calculated according to the values given in the edit fields. Unchecking the box will disable these fields and either set the values to the default ones or to the values found in the data file.

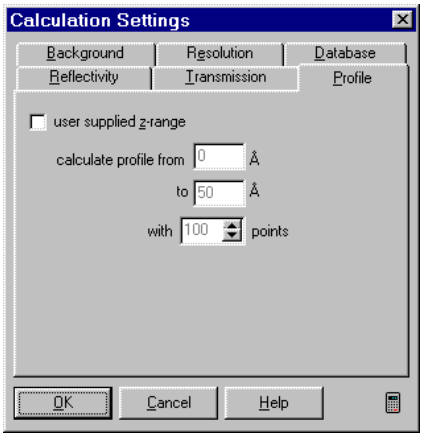
When the checkbox labeled "include magnetization" is checked, the reflectivity will be calculated for three different models:
a) $R(Q_z)$ from the *ordinary* values in the model file (`rho` and `Im(rho)`),
b) $R^+(Q_z)$ from `rho+ R(mag)` and `Im(rho)+I(mag)`,
c) $R^-(Q_z)$ from `rho-R(mag)` and `Im(rho)- I(mag)`.
The three calculations can be distinguished by their different colour in the reflectivity window:
R is plotted **black**
R^+ is plotted **red**
R^- is plotted **blue**.

3.5.2 Transmission Calculation Settings



For *free* samples, i. e. where the substrate is not infinitely thick on a macroscopic scale, the intensity of the transmitted radiation can be calculated. The calculation is mainly based on the length of the path the radiation travels through the substrate. Therefore the substrate's thickness has to be specified (in millimeters). Also, the wavelength of the incident radiation has to be specified (this is done on the "Reflectivity Settings"–page) since the absorption cross section of the substrate's material is corrected for the wavelength.

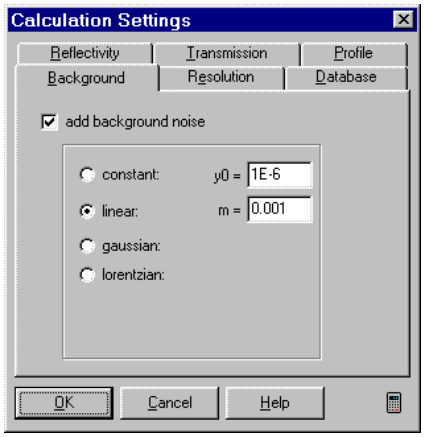
3.5.3 Profile Calculation Settings



The range of the calculation of the scattering length density profile can be set by checking the box marked "user supplied z–range". This will enable the edit fields, so that values may be entered for the lower and upper boundary as well as for the number of steps on the z–Axis. Unchecking this box will result in an automatic recalculation of the minimum and maximum setting for the z–range depending on the thicknesses given in the model window. For a single air/bulk interface the default values are -25 Å to +25 Å for samples with $N_{\text{layer}} \geq 1$ the calculation will range from -25% to +25% of the overall sample thickness. The overall sample thickness is calculated from *all* layers of the given model including the functional region of the model or the including the multi layer stack. In

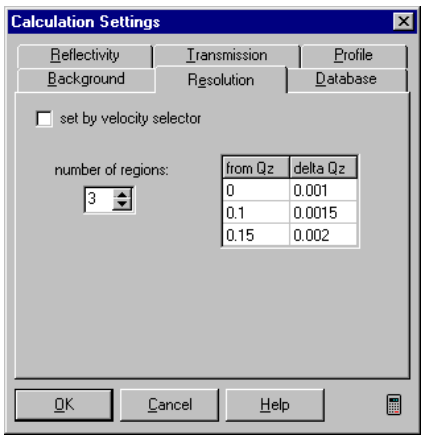
the case of multi layer or function models the default number of points where the scattering length is calculated (one hundred) can get too small to virtually *see* all features of the profile. In this case one is advised to manually choose the calculation region.

3.5.4 Background Settings



Working with real data often involves reflectivities measured down to a certain noise limit, which is also known as *instrumental background*. In the case of uncorrected data, which is, no background substracted from the raw data, this background can be added to the calculated reflectivity. Parratt32 offers different types of background noise to be added to the data: a) the typical noise threshold of the reflectivity instrument (constant y_0), b) linear increasing or decreasing background characterized by a constant y_0 and a slope m, c) and d) peaking background (either of gaussian or lorentzian shape) on a baseline that is characterized by five values: y_0 and m for the baseline and A, w, and x_c for the area, the width, and the center position of the peak.

3.5.5 Resolution Settings

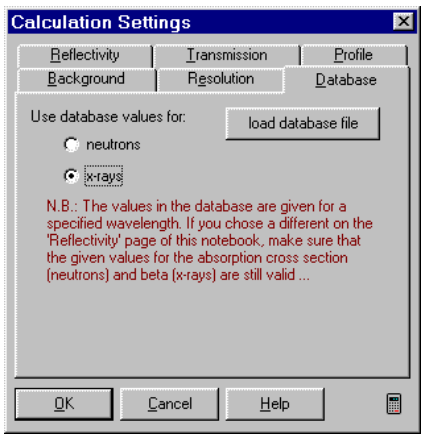


Real reflectometers all have a finite angular resolution. This means that measured $R(Q_z)$–values are virtually $R(Q_z \pm \delta Q_z)$ values. Parratt32 can approximate this resolution by calculating the reflectivity at a position Q_z plus gaussian weighted reflectivities at eight additional positions $Q_z \pm \delta Q_z$, $Q_z \pm 3\delta Q_z/4$, $Q_z \pm \delta Q_z/2$, and $Q_z \pm \delta Q_z/4$. Due to a lack of incoming radiation reflectivity curves are sometimes measured with different resolutions in different Q_z regimes, i. e. different settings of the collimation system. Therefore up to eight regions with different resolutions can be specified in the dialogue's resolution table. The first column is the starting value of the Q_z–region — which obviously should be zero in the first row —, the second column is the δQ_z. Some neutron reflectometers use a velocity selector instead of a crystalline monochromator. The wavelength distribution ($\Delta\lambda/\lambda$)of such a device is constant, hence the resolution decreases with increasing

in Q_z. Clicking the velocity selector checkbox allows you to enter the appropriate values.

3.5.6 The SLD Database Settings

There is a handy way to fill in the values for the different layers in the model window. By clicking with the right mouse button on the first column of a row in the model window a small pop–up menu will appear. Now, element's or compound's properties can be selected to be put into the respective row's `rho` and `Im(rho)` column. These values are taken from a database that Parratt32 loads at programme start–up. To correctly insert the values into the model window, the radiation type (neutrons or x–rays) has to be specified. For details on the database see below.

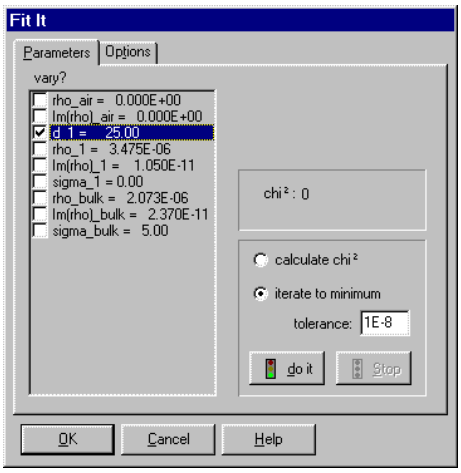


3.6 Fitting Models to Data

Once data is loaded and an appropriate model is set up, this model can then be fitted to the experimental data. By either selecting "Calculate–fit data" or by pressing the fit–button from the tool bar the FitIt dialogue box will show up. This dialogue box comes with two pages.

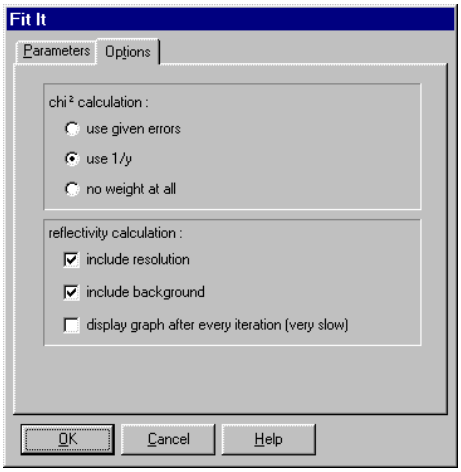
On page one three main elements can be found: the list of parameters, the χ^2 label and the *fitting control center*. Selecting the parameters is simply done by clicking on the checkbox next to it. All values of the parameters that are marked will be varied during the fitting procedure. The χ^2 label shows the momentary goodness–of–fit during the procedure. In the *fitting control center* one can either select a single χ^2 evaluation with the values of parameters given in the list or a complete minimization run with the selected parameters. The tolerance value given in the edit field is read out every time a new χ^2 was calculated. Convergence of a single parameter is reached when the following expression is satisfied: $|\chi^2_{\text{old}} - \chi^2_{\text{new}}| \leq \text{tolerance}$.

Once the "do it"–button is pressed, the calculation is under way. The progress of the calculation can be monitored in Parratt32's status bar. The first number given there is the number of the parameter which is currently varied, the second number is the current



Page 1 of the FitIt dialogue box

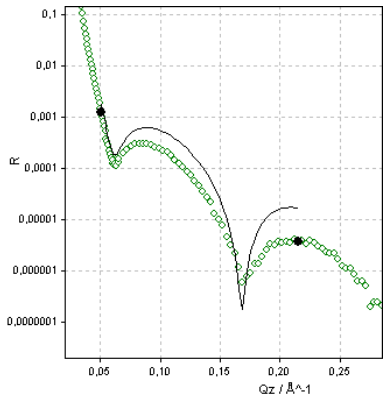
iteration step. The fitting procedure can be stopped by pressing the button labeled "stop".
Some fitting related settings can be made on page two of the FitIt dialogue box. At first the weighting of the χ^2 calculation has to be chosen. Additionally, one can include the background and resolution settings from the "Settings Dialogue" into the fit procedure.



Page 2 of the FitIt dialogue box

3.6.1 Defining a Fitting Region

A fitting region can be defined in the reflectivity window by the following procedure: Data has to be loaded, mark the left border of the region (low Q) by holding down the control key (`Ctrl`) and press the *left* mouse button when the mouse pointer is in the vicinity of the leftmost datapoint of the desired region. The corresponding data point will turn black! Now hold down Ctrl again and press the *right* mouse button on the rightmost datapoint. A message will pop up to tell about the region in Q_z that was selected. Once fitting is done the region is discarded and the reflectivity is calculated for the whole dataset again.



3.7 Saving the Results

The data of all three windows can be saved to file. In the file menu there are entries for saving the data of each window. One can also use the save button from the tool bar which then saves the contents of the currently active window (the one, which has the highlighted title bar).

The data from the model window will be saved in a special Parratt32 format, which contains all the layer information plus the background and resolution settings from the settings dialogue.

The data from the profile window will be saved as two–column ASCII, with the first column as $z/\text{Å}$ and the second column as $\varrho/\text{Å}^{-2}$.

The data of the reflectivity window depends on the magnetization and transmission settings. If none of these is selected the data will be saved as three TAB–delimited column ASCII files: column one are the Q_z–values, column two are the calculated reflectivity values and column three are zeros. If "include magnetization" is checked in the "settings dialogue box" then the file will be expanded to five columns, the last two containing $R^+(Q_z)$ and $R^-(Q_z)$. If "calculate transmission" is checked in the "settings dialogue box" then the file will expand to four columns, the fourth column containing the $T(Q_z)$ values. If both are checked, then there will be six columns in the file: $Q_z, R, 0, R^+, R^-, T$.

Some examples for the different file formats of calculated reflectivity data:

a) bare reflectivity:

Q_z	$R(Q_z)$	
0.000e00	1.000e00	0
2.500e-03	1.000e00	0
5.000e-03	1.000e00	0
7.500e-03	1.000e00	0
1.000e-02	9.999e-01	0
⋮	⋮	⋮

b) reflectivity and transmission:

Q_z	$R(Q_z)$		$T(Q_z)$
0.000e00	1.000e00	0	2.220e-16
2.500e-03	1.000e00	0	4.357e-06
5.000e-03	1.000e00	0	9.922e-06
7.500e-03	1.000e00	0	2.008e-05
1.000e-02	9.999e-01	0	1.030e-04
⋮	⋮	⋮	⋮

c) reflectivity including magnetization:

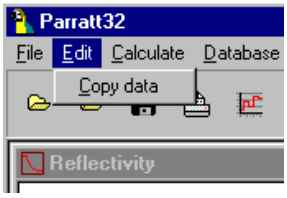
Q_z	$R(Q_z)$		$R^+(Q_z)$	$R^-(Q_z)$
0.000e00	1.000e00	0	1.000e00	1.000e00
2.500e-03	1.000e00	0	1.000e00	1.000e00
5.000e-03	1.000e00	0	1.000e00	1.000e00
7.500e-03	1.000e00	0	1.000e00	4.614e-01
1.000e-02	9.999e-01	0	1.000e00	4.795e-02
⋮	⋮	⋮	⋮	⋮

d) reflectivity including magnetization and transmission:

Q_z	$R(Q_z)$		$R^+(Q_z)$	$R^-(Q_z)$	$T(Q_z)$
0.000e00	1.000e00	0	1.000e00	1.000e00	2.220e-16
2.500e-03	1.000e00	0	1.000e00	1.000e00	4.357e-06
5.000e-03	1.000e00	0	1.000e00	1.000e00	9.922e-06
7.500e-03	1.000e00	0	1.000e00	4.614e-01	2.008e-05
1.000e-02	9.999e-01	0	1.000e00	4.795e-02	1.030e-04
⋮	⋮	⋮	⋮	⋮	⋮

3.8 Copying Data

Since Parratt32's graphical features are limited data from both the reflectivity window and the profile window can be copied to the Window's clipboard and pasted to other Windows applications, such as spreadsheet or scientific graphic applications (*i. e.* Excel or Origin).



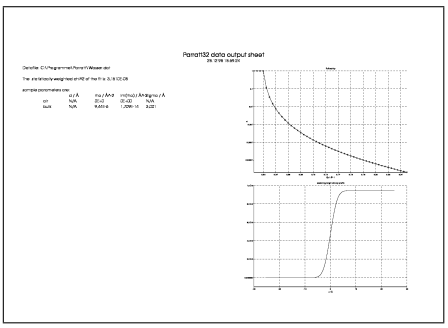
Copying the data from the profile window results in a two column dataset: z and $\varrho(z)$.

Copying the data from the reflectivity window leaves you with at least two columns in the clipboard: Q and $R(Q)$. If you selected to include the calculation of the transmission of the sample, then there

will be three columns: $Q, R(Q)$, and T . If magnetization is included, then there will be four columns: $Q, R(Q), R^{+}(Q)$, and $R^{-}(Q)$. And finally, if transmission and magnetization are included then there are five columns: Q, $R(Q)$, $R^{+}(Q)$, $R^{-}(Q)$, and T.

3.9 Printing the Results

For a *quick–and–dirty* overview of the results from a fitting session one can print out a summary sheet. Either selecting "File-Print..." or clicking the printer button from the tool bar will launch Windows' standard printing dialogue to select the printing device. Best results are achieved on A4 sized paper, but letter size will also work.



4 Some Internals

4.1 The SLD Database

PARRATT32 comes with a user expandable material's properties database. By default the relevant files are located in `<installdir>/database/`. The main file (`Parratt32.db`) is a Paradox7 table. This table can be edited via "Database–edit" from the main menu. The index files — files that provide the sorting order of the table — are automatically regenerated when new entries are send to the database. Elements are sorted according to the periodic system, compounds are sorted by name. From the values given in the database table PARRATT32 calculates the values `rho` and `Im(rho)` according to the following formulae:

$$
\begin{aligned}
\text{neutrons:} \qquad V[\text{Å}^3] &= \frac{M[\text{g/mol}]}{\rho_{\text{mass}}[\text{g/cm}^3] \cdot N_{\text{A}}[\text{mol}^{-1}]} \cdot 10^{24} \\
\rho[\text{Å}^{-2}] &= \frac{b_{\text{coh}}[\text{fm}]}{V[\text{Å}^3]} \cdot 10^{-5} \\
Im(\rho)[\text{Å}^{-2}] &= \frac{\text{abs\_xs[barn]}}{1.789[\text{Å}] \cdot 2 \cdot V[\text{Å}^3]} \cdot 10^{-8} \\
\text{x–rays:} \qquad \rho[\text{Å}^{-2}] &= \frac{2\pi\delta}{1.541^2} \\
Im(\rho)[\text{Å}^{-2}] &= \frac{2\pi\beta}{1.541 \cdot \lambda}
\end{aligned}
$$

Note: 1.789 Å (2200m/s) is the wavelength for which the absorption cross section values are tabulated, and 1.541 Å (Cu–K_α, 8047 eV) is the wavelength for which the δ and β values are supplied in the table.

4.2 Hidden Features

4.2.1 Command Line Parameters

For faster loading PARRATT32 can be started without the database file being loaded: simply type "`parratt nodatabase`" in the programme's directory, without the quotes, of course. The database file can then be loaded later on by selecting "Database-load from disk" from the main menu. Also, if you don't like the welcome screen simply type "`parratt nosplash`". Both of these parameters can be specified in the *properties dialog* of the shortcut placed in the *Start* menu by the installation procedure, see the Windows help for details...

4.3 Trouble is...

4.3.1 Installing

Here is the story why you get error messages installing PARRATT32: Since Inprise (formerly known as Borland) changed their database engine (BDE) to version 5.0 and gave it away for free, I thought it might be a good idea to have it on my system. It also had the advantage of being the english language version which was not available to me in version 4.52. But, the

Installshield Express Delphi Edition that came with Delphi 3.0 (which was the development tool for Parratt32) was not working correctly any more with this edition of the BDE. The support site of 'Installshield.com' had a little work–around for about two weeks in the late summer of 1998. It didn't work error free (as you see) but at least is does not prevent the installation of the software, completely. Then they stopped supporting this version of their software and removed the patch from their server. In the meanwhile Inprise came out with Delphi 4 which was *bundled* with a) the BDE 5.0 and b) a new version of Installshield Express Delphi Edition. But this also means that people like me, who do not own a Delphi 4 or an InstallShieldExpress 2.03 will never be able to make correctly working sets of installation media.
All this only concerns the installation of Parratt32. Once installed there are no known problems with BDE 5.0!

4.3.2 The Decimal Separator

What is a decimal separator most people will ask, especially when they are from english speaking countries. However, in Germany for instance the decimals of a real number are separated from the integer part by a comma. Microsoft Windows handles this with a setting in the control panel ("International" or "Systemsteuerung–Ländereinstellungen" in the german version). When you have installed Parratt32 and switch the decimal separator setting later on in either direction you will get some error messages concerning not–valid–floating–point–values: don't be surprised. Parratt32 tries to translate all these numbers depending on the settings you chose on your system with these exceptions: a) Values that were stored in the edit fields of some of the dialogue boxes (i.e. the 'settings' or the 'FitIt' dialogue) cannot be converted on-the-fly since you typed them in! when Parratt32 complains about these values try to look them up and correct them manually. b) Values are stored in the Windows' registry according to the current decimal separator setting, so after changing the separator setting while Parratt32 is not running these values are not longer valid and the programme will load default values instead of the ones found in the registry.
Always remember what decimal separator setting you have on your system and type in numbers according to this setting. Parratt32 is no wizard knowing that you meant "1.541" but Windows said you should have typed "1,541"!

5 The Source Code

Parratt32's language is Pascal. Since Pascal on a PC means Borland Turbo Pascal and programming graphical user interfaces for PCs means Microsoft Windows the right tool is called Delphi. Borland stopped calling its Pascal compiler Pascal after Borland Pascal 7.0, then came Delphi 1 (BP 8.0), Delphi 2 (BP 9.0) and Delphi 3. The command-line compiler says `Delphi for Win32 version 10.0`...
You do not have to know anything about Windows programming to understand the source code of Parratt32 and the use of Delphi — there is nothing like `int APIENTRY WinMain(hinstance, hinstPrev, lpCmdLine, nCmdShow)` like in C++ — , but a little knowledge of Pascal is recommended.

5.1 The Code Itself

5.1.1 Naming conventions

I tried to keep a reasonable naming scheme for the objects in the programme, variables, functions and procedures as well as for classes. The visual components all have a prefix to determine of what type they are:

prefix	component type	Delphi type
`lbl`	label	TLabel
`mnu`	menu item	TMenuItem
`btn`	button	TButton, TSpeedButton
`edt`	edit field	TEdit, TRealEdit, TCardinalEdit, TSpinEdit
`cb`	checkbox	TCheckbox
`rb`	radio button	TRadioButton

5.1.2 Globals

This is the file called `globals.pas` where all the global constants, types and variables are stored. It's not too much I guess...

```
unit Globals;
{*********************************************************************}
{ some constants and some types that are used globally                }
{*********************************************************************}
interface

const { some upper array boundaries and misc. constants }
        maxLayers = 499;
        maxQValues = 999;
        map = maxLayers*4+10;
      { different values for variable 'sampletype' }
        simplesample = 1;
        functionsample = 2;
        multisample = 3;
      { different function types }
        ft_builtin = 0;
        ft_user = 1;
      { Window IDs}
        wnd_Plot = 1;
        wnd_Model = 2;
        wnd_Profile = 3;
     { dataset IDs}
        ds_data = 0;
        ds_calc = 1;
        ds_prof = 0;
        ds_minus = 2;
        ds_plus = 3;
        ds_trans = 4;
```

```
        ds_fitwindow = 9;
     { Help IDs}
        hcIndex = 10;
        hcfrmPlot = 80;
        hcfrmModel = 40;
        hcfrmProfile = 70;
        fittingdialog = 350;
        settingsdialog = 330;
        functiondialog = 340;
        DatabaseEditor = 375;

        RegFilename = '\SOFTWARE\HMI\Parratt32';

type float = double;
     flarray = array[-5..maxLayers*4+5] of float;
     Pflarray = ^flarray;
     flarray1 = array[0..maxQValues] of float;
     PFlarray1 = ^flarray1;
     IntegerArrayMFIT = ARRAY [1..map] OF integer;
     FitParameter = array[1..maxLayers*4+10] of float;

var  a: Pflarray;                    { layer parameters for fitting         }
     q, refl, errors: Pflarray1;     { loaded data                          }
     resltn, bg: Pflarray1;          { resolution and background            }
     ndata: integer;                 { number of datapoints in loaded data  }
     nlparms: integer;               { number of layer parameters           }
     c_MinZ, c_MaxZ: integer;
     dataavail, hasbeenfitted, expert,
     splashing, databaseing: Boolean;
     sampletype, functiontype: Byte;
     fitwindow: array[0..1] of float;
     www_hmi, www_bensc, author_e_mail: String;
     datafilename, fn_databasename: String;

implementation

{*********************************************************************}


end.
```

5.1.3 Fequently Used Functions And Procedures

The file `rtfuncs.pas` contains most of the frequently used things, which are not directly related to events produced by the user interface, such as mouse clicks or menu messages. The following code shows the definition or interface part of this file

```
unit RTFuncs;
{ here are some functions that are used by all modules }
interface

uses Forms, Windows, SysUtils, Controls, Graphics, Dialogs, Printers, Classes,
     Menus, db, dbtables, xyGraph,
     Globals;

procedure LoadReflData;
procedure SaveReflData;
procedure LoadModelData;
procedure SaveModelData;
procedure SaveProfileData;
procedure SaveUserFunction;
procedure LoadUserFunction;
procedure LoadDatabase;
procedure PlotData;
procedure PrintResults;
function EvalFunc(z, rhostart, rhoend, p_A, p_B, p_C, p_D, p_E: double): double;
procedure GetParameters(magnsign: Integer);
procedure PutParameters;

implementation
.
.
.
end.
```

In brief these functions and procedures perform the following tasks:

`LoadReflData`	loads a data file from disk
`SaveReflData`	saves the calculated reflectivity to a file
`LoadModelData`	loads a model file from disk
`SaveModelData`	save the current model to disk
`SaveProfileData`	saves the calculated SLD profile to disk
`SaveUserFunction`	saves the user function to disk
`LoadUserFunction`	loads a user function from disk
`LoadDatabase`	loads the element/compound database from disk
`PlotData`	plots the calculated reflectivity
`PrintResults`	prints a summary sheet of the programme output
`EvalFunc(...)`	calculates the SLD at a given coordinate z
`GetParameters(...)`	puts data from model window in fitting vector `a`
`PutParameters`	put data from the fitting vector `a` into the model window

5.1.4 The Calculation of the Reflectivity

The Parratt formalism is coded in the file `uparratt.pas` according to the following formula:

$$
\begin{aligned}
\text{wave vector inside layer } n\text{:} \quad k_{z,n} &= \sqrt{k_{z,0}^2 - 4\pi\rho_n} \\
\text{reflectivity from layer } n\text{:} \quad r_{n,n+1} &= \frac{k_{z,n} - k_{z,n+1}}{k_{z,n} + k_{z,n+1}} \cdot e^{-2\sigma_{n+1}^2 k_{z,n} k_{z,n+1}} \\
\text{reflectivity of the system:} \quad R &= |R_0|^2 \\
R_{N+1} &= 0 \\
R_N &= r_{N,N+1} \\
R_n &= \frac{r_{n,n+1} + R_{n+1} e^{2id_{n+1}k_{z,n+1}}}{1 + r_{n,n+1} R_{n+1} e^{2id_{n+1}k_{z,n+1}}}
\end{aligned}
$$

NB: ρ_n is complex: $\rho_n = \varrho_n + i\mathrm{Im}(\varrho_n)$, where ϱ_n and $\mathrm{Im}(\varrho_n)$ are the values given in the model window.
The source is somewhat documented. Only a few remarks: procedures `kadd`, `ksub`, `kmul`, `kdiv`, `kexp`, and `ksqrt` all work on complex numbers (type `kompl`) and are defined in `cxmath.pas`. Since Pascal knowns no way of passing back user defined types in functions these all had to be made procedures that pass back the result in the last argument. For this reason there is heavy use of temporary variables in the code. The *bare* reflectivity is calculated by `cParratt(...)`, whereas the plotting routine calls `CalcParratt(...)`, where things like background and resolution are handled.

```
unit Uparratt;
{*********************************************************************}
{ This Unit calculates the reflectivity of a n-layer system at given Qz  }
{ R(Qz) := cParratt(Qz, resolution, nlayers, parameters)                }
{                                                                       }
{                                                                       }
```

```
{    input is as follows:                                                  }
{            double Qz: z-component of the scattering vector               }
{     double Resolution: Qz-resolution                                     }
{       integer nlayers: number of layers (no bulk, no vacuum)             }
{    pointer parameters: points to an array of the following parameters: }
{      parameters[1] := rho(air)                                           }
{                [2] := Imrho(air)                                         }
{                [3] := d(1st layer)                                       }
{                [4] := rho(1st layer)                                     }
{                [5] := Imrho(1st layer)                                   }
{                [6] := sigma(1st layer)                                   }
{                  ...                                                     }
{       [4*nlayer+3] := d(nth layer)                                       }
{                  ...                                                     }
{       [4*nlayer+6] := sigma(nth layer)                                   }
{       [4*nlayer+7] := rho(bulk)                                          }
{       [4*nlayer+8] := Imrho(bulk)                                        }
{       [4*nlayer+9] := sigma(bulk)                                        }
{ example:                                                                 }
{ for a single interface there will be 5 parameters:                       }
{          rho(air), Imrho(air), rho(bulk), Imrho(bulk), sigma(bulk)       }
{                                                                          }
{ d is in Angstroms (AA)                                                   }
{ rho is in AA^-2: rho_el * r_0 for x-rays,                                }
{                  Nb for neutrons                                         }
{ Imrho is in AA^-2: calculated from beta for xrays,                       }
{                    calculated from abs. cross section for neutrons       }
{ sigma is in AA                                                           }
{*************************************************************************}

interface

uses Globals, cxmath;

function calcParratt(qz, rsltn, bg: float;
                     nlayers: integer; parameters: Pflarray;
                     UseRsltn, UseBG: Boolean): float;

implementation

var rrn: array[0..maxLayers+1] of kompl;

function cParratt(qz: float; nlayers: integer; parameters: Pflarray): float;
var
    d, rho, imrho, sigma: array[0..maxLayers] of float;
    rcc, rnn1, arg, temp1, temp2, num, den: kompl;
    n : integer;
    e2idk, rme: kompl;
    kz0: float;

  procedure kzn(kz0: float; ni: integer; var res: kompl);
  var radix: kompl;
  begin
    radix[1] := kz0*kz0-4*pi*rho[ni];
    radix[2] := 4*pi*imrho[ni];                  // changed 08-02-98, cb
    ksqrt(radix, res);
  end;

  procedure rn(kz0: float; ni: integer; var res: kompl);
  var kz_n, kz_np1, rnum, rden, e2kks, rfres, arg1, arg2: kompl;
  begin
    kzn(kz0, ni, kz_n);
    kzn(kz0, ni+1, kz_np1);
    ksub(kz_n, kz_np1, rnum);
    kadd(kz_n, kz_np1, rden);
    kdiv(rnum, rden, rfres);
      { include roughness according to Nevot & Croce }
    kmul(kz_n, kz_np1, arg1);
    rmul(-2*sqr(sigma[ni+1]), arg1, arg2);
    kexp(arg2, e2kks);
    kmul(rfres, e2kks, res);
  end;

begin
  { map parameters^[] to d[], rho[], my[], sigma[] }
  rho[0]:= parameters^[1];
  imrho[0]:= parameters^[2];
  { calculate wavevector kz0': 1. vacuum, air -> Qz/2                      }
  {                            2. n0 <> 0 -> sqrt(Qz0^2/4+4*pi*rho0+4*pi*mu0)}
  temp1[1] := qz*qz/4+4*pi*rho[0];
  temp1[2] := 4*pi*imrho[0];
  ksqrt(temp1, temp2);
  kz0 := kabs(temp2);
  for n := 1 to nlayers do
```

```
    begin
      d[n] := abs(parameters^[(n-1)*4+3]);         // abs() added 22-01-98
      rho[n] := parameters^[(n-1)*4+4];
      imrho[n] := parameters^[(n-1)*4+5];
      sigma[n] := parameters^[(n-1)*4+6];
    end;
  rho[nlayers+1] := parameters^[nlayers*4+3];
  imrho[nlayers+1] := parameters^[nlayers*4+4];
  sigma[nlayers+1] := parameters^[nlayers*4+5];
  { done with remapping }
  rrn[nlayers+1, 1] := 0;
  rrn[nlayers+1, 2] := 0;
  { calculate rrn[nlayers]: bulk interface }
  rn(kz0, nlayers, rrn[nlayers]);
  { calculate the rrn[...] for the remaining layers }
  for n := nlayers-1 downto 0 do
    begin
      rn(kz0, n, rnn1);                     { r_n,n+1   }
      kzn(kz0, n+1, temp1);                 { k_z,n+1   }
      arg[1] := 0;
      arg[2] := 2*d[n+1];
      kmul(temp1, arg, temp2);              { 2 * i * k_z,n+1 * z_n+1        }
      kexp(temp2, e2idk);
      kmul(e2idk, rrn[n+1], rme);           { X_n+1 * exp(2 i k_z,n+1 z_n+1) }
      kadd(rnn1, rme, num);                 { r_n,n+1 + X_n+1 * exp(---)     }
      kmul(rnn1, rme, den);                 { r_n,n+1 * X_n+1 * exp(---)     }
      den[1] := den[1] + 1;
      kdiv(num, den, rrn[n]);
    end;
  rcc[1] := rrn[0,1]; rcc[2] := -1*rrn[0,2];
  kmul(rrn[0], rcc, temp1);
  cParratt := temp1[1];
end;

function calcParratt(qz, rsltn, bg: float;
                     nlayers: integer; parameters: Pflarray;
                     Usersltn, UseBG: Boolean): float;
var refl: float;
begin
  refl := cParratt(qz, nlayers, parameters);
  if UseRsltn then
    begin
      { approximate Gaussian beam profile by 8 pivot points }
      refl := refl + 0.135*cParratt(qz-rsltn, nlayers, parameters);
      refl := refl + 0.135*cParratt(qz+rsltn, nlayers, parameters);
      refl := refl + 0.325*cParratt(qz-rsltn*3/4, nlayers, parameters);
      refl := refl + 0.325*cParratt(qz+rsltn*3/4, nlayers, parameters);
      refl := refl + 0.605*cParratt(qz-rsltn/2, nlayers, parameters);
      refl := refl + 0.605*cParratt(qz+rsltn/2, nlayers, parameters);
      refl := refl + 0.88*cParratt(qz-rsltn/4, nlayers, parameters);
      refl := refl + 0.88*cParratt(qz+rsltn/4, nlayers, parameters);
      refl := refl/4.89;
    end;
  if UseBG then refl := refl + bg;
  calcParratt := refl;
end;
end.
```

5.1.5 The Calculation of the Transmission

Calculation of the transmission of "free samples" is done according to the following formula:

$$T(Q_z) = (1 - R(Q_z))T_{\text{subst}}(Q_z)$$

where $T_{\text{subst}}(Q_z)$ is the transmission through the substrate, which is calculated by the following code:

```
function calcTrans(qz: float; nlayers: integer; parameters: Pflarray;
                   wavelength, substratethickness: float): float;
var pathlength: float;
    d, rho, imrho: float;
    temp1, temp2, temp3: kompl;
    n: integer;
    currentk: kompl;
begin
  pathlength := 0;
  for n := 1 to nlayers do
    begin
```

```
      d := abs(parameters^[(n-1)*4+3]);
      rho := parameters^[(n-1)*4+4];
      imrho := parameters^[(n-1)*4+5];
      temp1[1] := d*2*pi/wavelength;
      temp1[2] := 0;
      temp2[1] := qz*qz/4-4*pi*rho;
      temp2[2] := 4*pi*imrho;
      ksqrt(temp2, currentk);
      kdiv(temp1, currentk, temp3);
      pathlength := pathlength + temp3[1];
    end;
  rho := parameters^[nlayers*4+3];
  imrho := parameters^[nlayers*4+4];
  d := substratethickness*1e7;    // thickness is in millimeters, we need AA
  temp1[1] := d*2*pi/wavelength;
  temp1[2] := 0;
  temp2[1] := qz*qz/4-4*pi*rho;
  temp2[2] := 4*pi*imrho;
  ksqrt(temp2, currentk);
  kdiv(temp1, currentk, temp3);
  pathlength := pathlength + temp3[1];
  if pathlength > 0 then result := exp(-2*wavelength*imrho*pathlength)
                    else result := 0;
end;
```

5.1.6 Calculation of the Scattering Length Density Profile

The calculation of the scattering length density profile from the given layers in the model is done by evaluating the following formula:

$$\varrho(z) = \sum_{i=1}^{N} \frac{\varrho_i - \varrho_{i+1}}{2} \left(1 + \operatorname{erf}\left(\frac{z - z_i}{\sqrt{2}\sigma_i}\right)\right),$$

where N is the overall number of layers, ϱ_i is the scattering length density of the ith layer at position z_i and with a gaussian roughness σ_i. The error function erf(x) is approximated by a 5th order polynomial given by Press et al.[Press88].

```
procedure TfrmProfile.ShowProfile;
var i, j, s_index, r_index, d_index,
    lastlayer, zstart, zend, temp: integer;
    z, zinc, sum,
    deltarho, sigma, zi: float;

  function erfcc(x: float): float; { according to 'NumRecip' }
  var
     t,z,ans: extended;
  begin
     z := abs(x);
     t := 1.0/(1.0+0.5*z);
     try
       ans := t*exp(-z*z-1.26551223+t*(1.00002368+t*(0.37409196+t*(0.09678418
                +t*(-0.18628806+t*(0.27886807+t*(-1.13520398+t*(1.48851587
                +t*(-0.82215223+t*0.17087277)))))))));
     except
       ans := 0;
     end;
     if x >= 0.0 then erfcc := ans
     else erfcc := 2.0-ans
  end;

begin
  Screen.Cursor := crHourGlass;
  GetParameters(0);
  xyGraph1[ds_prof].Free;
  xyGraph1.Plotting := False;
  with xyGraph1[ds_prof] do
    begin
      DrawPoints := False;
      LineColor := clBlack;
    end;
  if SettingsDlg.cbUserZ.Checked then
    with SettingsDlg do
      begin
        zstart := edtMinZ.Value;
```

```
        zend := edtMaxZ.Value;
        if zstart > zend then
          begin
            temp := zstart;
            zstart := zend;
            zend := temp;
          end;
      end
    else CalcZBounds(zstart, zend);
  z := zstart;
  zinc := (zend-zstart)/SettingsDlg.edtNumPointsZ.Value;
  while z <= zend do
    begin
      lastlayer := 0;
      case sampletype of
        simplesample:
          lastlayer := frmModel.edtLayers.Value;
        functionsample:
          lastlayer := frmModel.edtLayers.Value+frmModel.edtNumCalcLayers.Value;
        multisample:
          lastlayer := frmModel.edtLayers.Value
                       +frmModel.edtRepts.Value*frmModel.edtStackedLayers.Value;
      end;
      sum := a^[1];
      for i:= 1 to lastlayer+1 do
      begin
        if i = lastlayer+1 then
          begin
            r_index := 4*i-1;             { rho_bulk }
            s_index := r_index+2;         { sigma_bulk }
            if lastlayer = 0
              then deltarho := a^[r_index] - a^[r_index-2]
              else deltarho := a^[r_index] - a^[r_index-3];
          end else
            begin
              s_index := 4*i+2;
              r_index := 4*i;
              if i = 1
                then deltarho := a^[r_index]-a^[r_index-3]
                else deltarho := a^[r_index]-a^[r_index-4];
            end;
          zi := 0;
          j := 1;
          if i>1 then while j<i do
            begin
              d_index := 4*(j-1)+3;
              zi := zi + a^[d_index];
              inc(j);
            end;
          if a^[s_index] = 0 then sigma := 1e-3
            else sigma := a^[s_index];
          sum := sum + deltarho/2*(2-erfcc((z-zi)*sqrt(2)/2/sigma)); // changed on 24-NOV-99
        end;
      xyGraph1[ds_prof][z] := sum;
      Form1.Statusbar1.Panels[0].Text :=
          Format('calculating rho(z), z = %f', [z]);
      Form1.Statusbar1.Refresh;
      z := z + zinc;
    end;
  Screen.Cursor := crDefault;
  xyGraph1.Dimensions.YAxisTitleOffset := 20;
  xyGraph1.Plotting := True;
  Form1.Statusbar1.Panels[0].Text := '';
end;
```

5.1.7 Fitting of A Model to Reflectivity Data

The code for the optimization of the layers's parameters is a little too complex to be described in just one formula. The minimization is done by least squares fitting:

$$\chi^2 = \sum_{i=1}^{M} \left(\frac{R^{\text{calc}}_{Q_{z,i}} - R^{\text{meas}}_{Q_{z,i}}}{\text{weighting}} \right)^2 ,$$

where *weighting* is either 1 ("no weighting") or $R^{\text{meas}}_{Q_{z,i}}$ ("statistical weighting") or $\delta R^{\text{meas}}_{Q_{z,i}}$ ("error weighting"), see function `CalcChisq(weight: integer)` for details, and M is the num-

ber of datapoints. The parameter minimization is done in the procedure `Newton(parindex: integer)`, which is a simple Newton method for finding minima in parametric functions. It in(de)creases the parameter value until the difference in χ^2 between two successive steps falls below a given threshold (`edtTolerance.Value`). The rest of the code deals with chosing of parameters, fitting regions in Q_z etc.

```
procedure TFitDlg.btn1iterClick(Sender: TObject);
var i, j, nvarpar, RB: integer;
    lista, listadone: IntegerArrayMFIT;
    chisq, chiold, chidiff: real;
    finished: Boolean;
    iter: longint;
    nol, weightingmethod : integer;
    lastvarpar: integer;
    FitndataStart, FitndataEnd: integer;

  function CalcChisq(weight: integer): float;
  var cref, cs: float;
      i: integer;
  begin
    cs := 0;
    if cbDisplayGraph.Checked then
      begin
        frmPlot.xyGraph1.Plotting := False;
        frmPlot.xyGraph1[ds_calc].Free;
        with frmPlot.xyGraph1[ds_calc] do
          begin
            DrawPoints := false;
            LineColor := clBlack;
          end;
      end;
    for i := fitndataStart to fitndataEnd do
      begin
        cref := calcParratt(q^[i], resltn^[i], bg^[i],
                            nol, a, cbRsltn.Checked, cbBackground.Checked);
        case weight of
          1: cs := cs + sqr((refl^[i]-cref)/errors^[i]);
          2: cs := cs + sqr((refl^[i]-cref)/refl^[i]);
          3: cs := cs + sqr(refl^[i]-cref);
        end;
        if cbDisplayGraph.Checked then frmPlot.xyGraph1[ds_calc][q^[i]] := cref;
      end;
    CalcChisq := cs/(fitndataEnd-fitndataStart);
    if cbDisplayGraph.Checked then
      begin
        frmPlot.xyGraph1.Plotting := True;
        lblChisq.Caption := FormatFloat('0.0000E+00',
                                        cs/(fitndataEnd-fitndataStart));
      end;
  end;

  procedure Newton(parindex: integer);
  var incr, fac, newchisq: float;
  begin
    chisq := CalcChisq(weightingmethod);
    if P[parindex]=0 then incr := 1e-7 else incr := P[parindex] * 0.1;
    fac := 1;
    iter := 0;
    repeat
      repeat
        inc(iter);
        Form1.StatusBar1.Panels[0].Text := IntToStr(parindex)+' : '
                                           +IntToStr(iter);
        Form1.Statusbar1.Refresh;
        P[parindex] := P[parindex]+incr*fac;
        CalcAfromP;
        if cbDisplayGraph.Checked then ListBoxUpdate;
        newchisq := CalcChisq(weightingmethod);
        chidiff := chisq-newchisq;
        if chidiff<0 then
          begin
            fac := -fac;
            P[parindex] := P[parindex]+incr*fac;
          end else
            begin
              chisq := newchisq;
              fac := fac*2;
            end;
        Application.ProcessMessages;
      until (chidiff<0) or (iter>99);
```

```
      fac := fac/2;
      Application.Processmessages;
      if ModalResult = mrCancel then stopfitting := True;
    until ((abs(chidiff)<=edtTolerance.Value) and (iter>2)) or stopfitting;
  end;

begin
  stopfitting := False;
  converged := False;
  lblconverged.Visible := False;
  btnStop.Enabled := True;
  Screen.Cursor := crHourGlass;
  { set weighting method }
  if RBNone.Checked then weightingmethod := 3;
  if RBStat.Checked then weightingmethod := 2;
  if RBErrors.Checked then weightingmethod := 1;
  { get variable parameters from CheckListBox1 }
  nvarpar := 0;
  i := 0;
  while i <= CheckListBox1.Items.Count-1 do
    begin
      if CheckListBox1.Checked[i] then
        begin
          lista[nvarpar+1] := i+1;
          inc(nvarpar);
        end;
      inc(i);
    end;
  { set overall number of layers }
  nol := frmModel.edtLayers.Value;
  case sampletype of
    functionsample: nol := nol + frmModel.edtNumCalcLayers.Value;
    multisample:    nol := nol + frmModel.edtStackedLayers.Value
                               *frmModel.edtRepts.Value;
  end;
  { get fitting region }
  if (fitwindow[0] = -1) and (fitwindow[1]=-1) then
    begin     { no user region specified }
      fitndataStart := 1;
      fitndataEnd := ndata;
    end else
      begin     { user has specified a region }
        i := 0;
        repeat
          inc(i);
        until (i > ndata) or (q^[i] > fitwindow[0]);
        fitndataStart := i-1;
        i := 0;
        repeat
          inc(i);
        until (i > ndata) or (q^[i] > fitwindow[1]);
        fitndataEnd := i-1;
      end;
  { check if user wants to fit or only the chisq }
  RB := 0;
  if RBchisq.Checked then RB := 1;
  if RBfulliter.Checked then RB := 2;
  case RB of
    1: lblChisq.Caption := FormatFloat('0.0000E+00', CalcChisq(weightingmethod));
    2: begin
         { 'make' all parameters are still unminimized }
         for i := 1 to nvarpar do listadone[i] := 1;
         finished := False;
         lastvarpar := -1;
         repeat
           if nvarpar > 1 then
             repeat
               j := Trunc(Random*nvarpar)+1;
             until j <> lastvarpar
             else j := 1;
           lastvarpar := j;
           if listadone[j]=1 then
             begin
               chiold := CalcChisq(weightingmethod);
               Newton(lista[j]);
                if chisq<chiold then for i:=1 to nvarpar do listadone[i]:=1
                                else listadone[j] := 0;
             end;
           lblChisq.Caption := FormatFloat('0.0000E+00', chisq);
           { replot the graph }
           frmPlot.xyGraph1.Plotting := False;
           frmPlot.xyGraph1[ds_calc].Free;
           with frmPlot.xyGraph1[ds_calc] do
             begin
```

```
              DrawPoints := false;
              LineColor := clBlack;
            end;
          for i := fitndataStart to FitndataEnd do
            frmPlot.xyGraph1[ds_calc][q^[i]] :=
              calcParratt(q^[i], resltn^[i], bg^[i], nol, a,
                          cbRsltn.Checked, cbBackground.Checked);
          frmPlot.xyGraph1.Plotting := True;
          { update the parameters in CheckListBox1 }
          ListBoxUpdate;
          { check for 'un-minimized' parameters }
          j := 0;
          for i := 1 to nvarpar do j := j+listadone[i];
          if j=0 then converged := True;
          if (ModalResult = mrCancel) or stopfitting then finished := True;
        until (chisq<edtTolerance.Value) or finished or converged;
        if converged then lblConverged.Visible := True;
        CalcAfromP;
        Form1.StatusBar1.Panels[0].Text := '';
      end;
  end;
  btnStop.Enabled := False;
  Screen.Cursor := crDefault;
  MessageBeep(0);
  Application.ProcessMessages;
end;
```

5.2 Compiling

5.2.1 What you need...

1. The sources. These can be obtained from Christian Braun by e–mail: braun@atmos.umnw.ethz.ch or ask Roland Steitz (steitz@hmi.de) for my whereabouts.

2. A complete installation of Delphi. Version 3 professional is recommended, version 2 could work, version 4 does work.

3. The following non–standard Delphi components:

   a) TXYGraph: a component for xy-diagrams from Grahame Grieve at Kestral Computing

   b) RZNEdit: an advanced edit field component from Robert Kratz

   c) TParser 10.1: a function parsing component from Stefan Hoffmeister based on previous work of Renate Schaaf and Alin Flaider

   d) TGotoWeb: a browser launching component by Sylvain Cresto

   e) TVersionInfo: a non-visual component that reads the version information of a file, by an unknown author

   All of them are freeware and can be found on the Delphi Super Page.

6 Appendix

Acknowledgements

Holger Rhan (rhan@desy.de) for ymath62.pas
The Technische Universität Berlin for the hardware (IBM 486DX2/66 Mhz boosted to 80 Mhz) and software (Windows NT 4.0, Borland Delphi 3)
The Hahn–Meitner–Institut Berlin (HMI) for funding
and to all those who complained about missing features, buggy procedures, and gave lots of helpful hints.

References

[Parratt54] L. G. Parratt, *Phys. Rev*, **95**(2), 359-369, 1954

[Nevot80] L. Névot, P. Croce, *Revue de physique appliquée*, 15 (1980), 761

[Press88] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling: "Numerical Recipes", Cambridge University Press, 1988

[Liu89] A. J. Liu, M. E. Fisher, *Phys. Rev. A*, **40** (1989), 7202

Index