Modeling Organic Molecule Thin Film Growth

From In Situ X-Ray Reflectivity to Atomistic Growth Processes

Tammo Rukat

Born on the 12th of December 1988 in Berlin

Department of Physics

Humboldt-Universität zu Berlin

Submitted in fulfillment of the requirements for the academic degree of

Bachelor of Science in Physics

23rd of September 2011

1. Reviewer: Prof. Dr. Stefan Kowarik
2. Reviewer: Prof. Dr. Jürgen P. Rabe

Abstract

This thesis delves into the simulation and analysis of reflectivity data obtained by real-time x-ray studies during organic molecular thin film growth. On that account two growth-models are introduced and the highly flexible and modifiable python program *GOfit* for simulation and reflectivity-fitting within those models is developed.
The insights in growth behavior obtainable by in-situ reflectivity measurements and their limits are discussed. Special attention is directed at the island rearrangement behavior (impingement or coalescence) and interlayer transport during growth. The importance of choosing the appropriate observation point in reciprocal space (q-point) in experimental setups is discussed.
It is found, that growth oscillations within their limitations are a powerful tool to examine growth kinetics on scales down to atomistic processes.
On behalf of modeling and fitting *GOfit* proves to be the appropriate tool to exhaust these limitations.

Zusammenfassung

Die vorliegenden Arbeit widmet sich der Simulation und Analyse von Echtzeit-Röngtenreflektvitätsdaten, die während des Wachstums organischer dünner Filme aufgenommen werden. Es wird das äußerst flexible, leicht modifizier- und erweiterbare Python Programm *GOfit* zum Simulieren und Fitten von Reflektivitätsdaten entwickelt. Es fußt auf Wachstumsmodellen, die ausführlich vorgestellt werden.
Die Erkenntnisse bezüglich des Wachstumsverhaltens, die mittels in-situ Reflektivitätsmessungen und deren Simulation gewonnen werden können, sowie ihre Grenzen werden diskutiert. Besonderes Augenmerk liegt dabei auf der Umbildung der Inselkonturen beim Zusammenwachsen von Inseln, sowie auf dem Interlagentransport. Es wird die Bedeutung der Wahl des passenden Beobachtungspunkts im reziproken Raum (q-Punkt) bei der Durchführung von Experimenten verdeutlicht.
Wir zeigen, dass Wachstumsoszillationen, im Rahmen ihrer Mögichkeiten, ein leistungsfähiges Werkzeug zur Untersuchung des Wachstumsverhaltens, bis hin zu atomistischen Prozessen, sind.
Seitens der Simulation und der Anpassungsrechnung erweist sich *GOfit* als geeignetes Analysewerkzeug, diesen Rahmen auszureizen.

Contents

1

Motivation and Purpose

X-ray methods are suitable to obtain information about the structure for the first few monolayers of thin films. This structure is often crucial for the opto-electronic effectivity of whole devices [1–3]. Real-time-methods are applicable, as the structural properties of molecular layers are determined by their growth [4]. This thesis elucidates the capabilities and limitations of real-time x-ray reflectivity to obtain insights in growth kinetics of molecular thin films. We particularly address the following issues:

1. For the analysis of reflectivity data the python program *GOfit* needs to be developed. Based on growth models we introduce, it is to determine the time-dependent x-ray reflectivity and to provide a simple and flexible command-line interface. The feature of fitting experimental data with suitable optimization algorithms is its main objective.

2. Modeled reflectivity boasts several distinct features, which are examined and interpreted. We want to reveal the model behavior under variation of physical parameters and deduce advice for data fitting. The importance of choosing the appropriate observation point in reciprocal space (q-space) for the particular information desired is to be illustrated.

3. Subsequently we want to investigate atomistic growth properties: The influence of the island rearrangement behavior on the molecule-layer evolution and reflectivity is subject of our studies. We examine the influence of *downhill-transport* on the growth process and eventually figure out to what extent such particular

parameters are accessible by reflectivity-fits.

2

Theoretical Background

2.1 X-Ray Scattering Geometry

X-ray scattering for the investigation of thin film growth is predominately referred to as *elastic* photon-electron-scattering. Thus, considering fig. 2.1, we have $|\mathbf{k}| = |\mathbf{k}'|$, where $\mathbf{k}$ and $\mathbf{k}'$ are the wave vector of the incoming and the outgoing wave, respectively. The wave vector transfer is $\mathbf{q} = \mathbf{k}' - \mathbf{k}$ and implies:

$$|\mathbf{q}| = 2\,|\mathbf{k}| \sin\vartheta = \frac{4\pi}{\lambda} sin\vartheta. \tag{2.1}$$

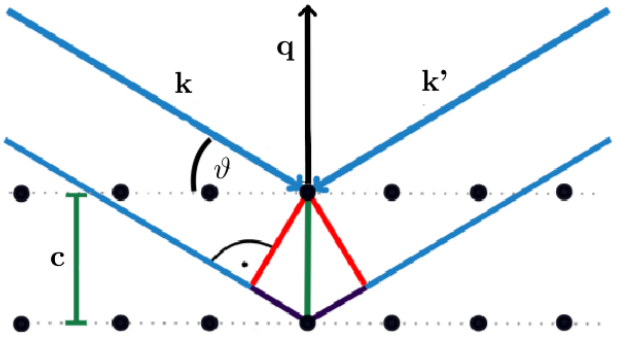


Figure 2.1: Diffraction scheme. Two incoming waves with momentum $\mathbf{k}$ and identical phase are scattered on two different plains of a crystalline solid.

2.2 Kinematical Scattering Theory

To describe such scattering events rigorously, the intensity distribution of the scattered photons $I(\mathbf{q})$ has to be derived.
Far from the Bragg peak or from the critical angle for total external reflection the Born approximation may be used [5]. In this approximation the scattered intensity $I(\mathbf{q})$ is proportional to the square of the scattering amplitude $|A(\mathbf{q})|^2$ and the latter is the Fourier transform of the electron density $\rho(\mathbf{r})$:

$$A(\mathbf{q}) = \int_V d^3r\, \rho(\mathbf{r}) \exp(-i\mathbf{q} \cdot \mathbf{r}) \tag{2.2}$$

Thus, we have a connection between the electron density $\rho(\mathbf{r})$ and the actual measured Intensity $I(\mathbf{q})$. This approximation is also known as *kinematic approximation* and is equivalent to the neglection of multiple scattering events. Therefore it is referred to as *single scattering approximation*, as well.

2.3 Thin Film Growth of Organic Semiconductors

2.3.1 Surface Phenomena

In the last two decades organic molecular beam deposition (OMBD) has become a widely used technique to produce high crystalline organic molecule thin films [6].
Due to several effects occurring simultaneously the process of depositing molecules on surfaces is very complex and to a great extent not well understood.[1] The main processes happening during organic thin film growth, as shown in fig. 2.2, are:

[1] One of the most challenging features, in comparison to single molecule growth are the orientational and vibrational degrees of freedom and the finite size of the molecules. (See [7] for a more detailed discussion)

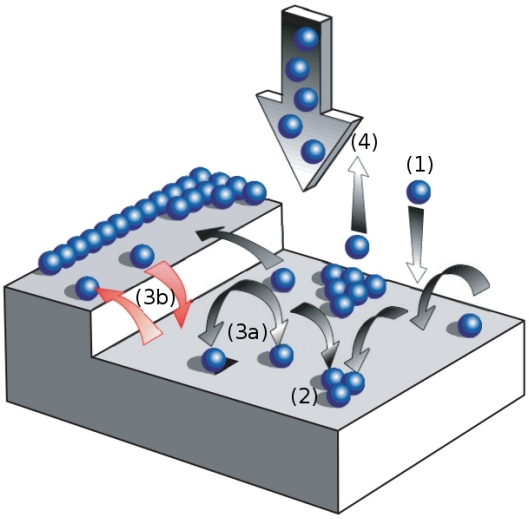


Figure 2.2: Schematic of relevant processes in thin film growth, as discussed in the text. (Figure from [7], further information is provided in [6]).

(1) **Adsorption**: As the evaporated molecules reach the substrate they condense on the surface and are able to interact with vicinal molecules.

(2) **Nucleation**: Particularly caused by Van-der-Waals forces molecules nucleate at energetic favorable sites such as step edges or defects within the substrate surface. A so called *critical island size* yields crystallization, which means that the islands become stable towards dissociation. Depending on the investigated system and the deposition parameters, a *dendritic* or a more *compact* island shape might be favored [8].

(3) **Diffusion**: Surface diffusion can be understood as a random-walk-like process of molecules at a solid surface. *Hetero-diffusion* describes the movement of molecules directly on the substrate, whereas *self-diffusion* is the process of molecules moving alongside each other. A further, for our topic especially important, distinction is made between *intralayer* (3a) and *interlayer diffusion* (3b).[9–11]

(4) **Desorption**: Molecules may have sufficient kinetic energy to exceed the binding energy of the surface. This process depends mainly on the surface temperature and may (for high temperatures) even surpass adsorption.

2.3.2 Morphology

One usually distinguishes the three different growth scenarios [4] illustrated in fig. 2.3:

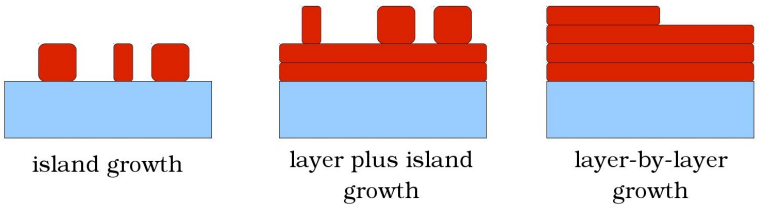


Figure 2.3: Growth modes for organic molecular beam deposition [12]

These are:

- island growth (Volmer-Weber)
- layer-plus-island growth (Stranski-Krastanov)
- layer-by-layer growth (Frank-van-der-Merwe)

Most organic systems exhibit layer-plus-island (LPI) growth, whereas for technical applications often smooth and homogeneous films and therefore layer-by-layer (LBL) growth is desired[1]. The transition of such growth modes, e.g. dependent on temperature or incoming flux is of great interest. A detailed discussion of the transition behavior is given in [14].

[1]Growth behavior can be influenced by use of SAMs (self-assembled monolayers), that modify the substrate surface energy. (See [13] for more details)

2.3.3 Growth Oscillations

Oscillations of the specular x-ray intensity during thin film growth originate from the **phase sensitive** summation of the reflected beam amplitudes. Such reflection is induced by the top surface of the thin film, the crystal lattice surface, the substrate and, potentially, some interfacial layer between film and substrate. [15] (See fig. 2.4).

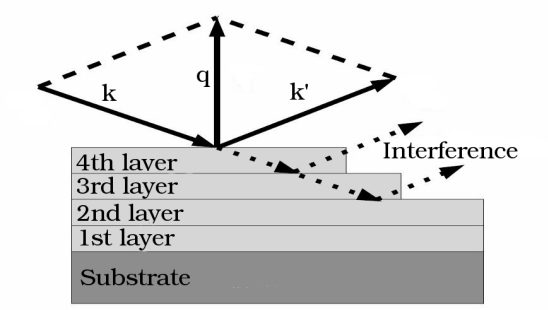


Figure 2.4: Interference causing growth oscillations. The incoming beam is reflected on several layers, reflected beams interfere [12]

For our purpose of determining layer coverages Θ, we utilize the kinematic approximation (2.2) to get an expression for the reflected intensity as a function of this time dependent layer coverage $\Theta(t)$.

To introduce parameters, that are closer related to the morphology of the deposited molecules, the layers can be approximated as uniform density slabs, which is a valid approximation for $\frac{2\pi}{q} \gg$ *interatomic distances*[16]. Thus, we identify the Amplitudes A with the layer coverage Θ and the respective electron density ρ. Similarly, the phase of each layer is identified with its thickness d. Further on, we introduce reciprocal lattice units ($L = \frac{qc}{2\pi}$) and integrate over the semi-infinite substrate, interlayer, inferfacial and molecule layers. The according setup is depicted in fig. 2.5:

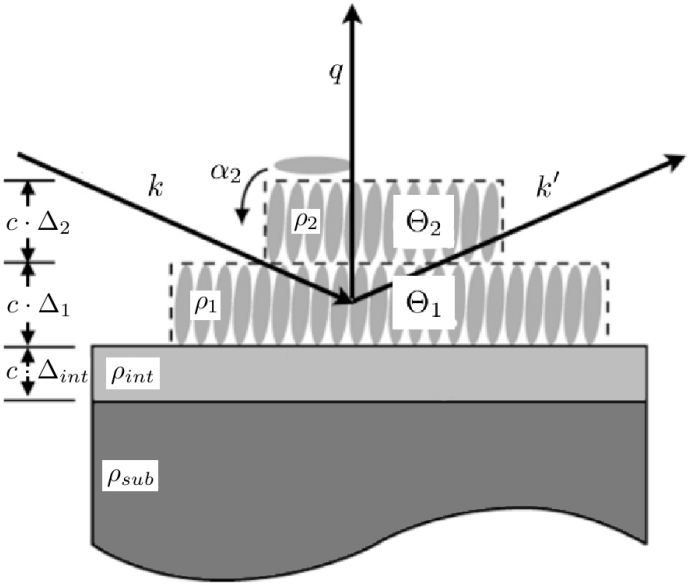


Figure 2.5: Schematic of our thin-film setup with growth parameters. ρ_{sub} is the electron of the substrate (which is assumed to have infinite height). d_{int} is the height of the interfacial layer as fraction of the lattice parameter c, ρ_{int} its electron density of the interfacial layer. d_n accounts for the thickness of the nth layer. ρ_n is the electron density of layer n and α_n is the parameter measuring the downhill transport into layer n. Θ_n denotes the fractional coverage of layer n.

$$\begin{aligned}
\frac{A(q)}{A_0} &= \rho_{sub} \int_{-\infty}^{-c\cdot d_{int}} \mathrm{d}\mathbf{r} e^{-i\mathbf{qr}} + \rho_{int} \int_{-c\cdot d_{int}}^{0} \mathrm{d}\mathbf{r} e^{-i\mathbf{qr}} + \rho_{mol} \sum_n \int_{c\cdot d_{n-1}}^{c\cdot d_n} \mathrm{d}\mathbf{r} e^{-i\mathbf{qr}} \\
&= \frac{1}{q}\rho_{sub} e^{2\pi \mathrm{i}\cdot L\cdot d_{int}} + \rho_{int}\left(1 - e^{2\pi \mathrm{i}\cdot L\cdot d_{int}}\right) + \sum_{n=1} \rho_{film}\Theta_n e^{2\pi \mathrm{i}\cdot L\cdot d_n} \qquad (2.3) \\
\Rightarrow \frac{I(q)}{I_0} &= \frac{1}{q^2}\left|\rho_{sub} e^{2\pi \mathrm{i}\cdot L\cdot d_{int}} + \rho_{int}\left(1 - e^{2\pi \mathrm{i}\cdot L\cdot d_{int}}\right) + \sum_{n=1} \rho_{film}\Theta_n e^{2\pi \mathrm{i}\cdot L\cdot d_n}\right|^2
\end{aligned}$$

For complete layer-by-layer growth the intensity oscillates in time with a periodicity, that depends on the observation point in reciprocal space. For $L = 0.5$ ($q = \frac{1}{2}q_{Bragg}$) it is two-layer-periodic, for $L = \frac{2}{3}$ three-layer-periodic and so forth, as has been shown by Kowarik et al.[17]. A general rule is, that measurements at higher q points provide rather multilayer than single- or duallayer information.

Usually, the unknown parameters in this setup are the amplitudes of the different layers' contributions, the thickness of substrate and layers d and the layer coverages $\Theta_n(t)$. Determining these parameters by means of experimental reflectivity data is one of the main reasons we develop *GOfit*.

Amplitudes and phases can in principle be determined by fitting eq. (2.3) to experimental data. The ensuing task is to provide a *growth model*, that computes the coverages $\Theta_n(t)$.

2.4 Modeling Thin Film Growth

A large variety of growth models has been deployed over the last decades to describe the growth of thin films [19, 20]. The most common approach to model the evolution of layer coverage Θ is the construction of a set of coupled rate equations, that describe the change in coverage of each layer:

$$\frac{d\Theta_n}{dt} = F(k_j, ..., \Theta_{n-1}, \Theta_n, \Theta_{n+1}, ..., t) \tag{2.4}$$

Θ_n is the relative coverage of the nth layer and k_j is a set of parameters determining all Θs on the right-hand side.

We introduce modified versions of the model by Trofimov [14, 18, 21–25] and the model by Cohen [19]. Both are based on rate equations, which have the same structure as 2.4.

2.4.1 The Trofimov Model

The model suggested by Trofimov [18, 25] attempts to model atom-level kinetics. We use a reparametrized (simplified) version of the model according to Woll [16].

To control interlayer transport the model introduces the concept of a *feeding-zone*. This zone can be thought of as a fractional part of each layer with a certain size ξ ($\xi \leq \Theta$) as depicted and described in fig 2.6.

Every molecule that lands on top of the feeding zone of layer n (ξ_n) contributes to

layer n, whereas every molecule, that lands within the λ-band of layer n (λ_n) diffuses downhill and contributes to layer $(n-1)$. The λ-band that is the fraction of the layer, which is not part of the feeding zone and has therefore a relative area of $\lambda = \Theta - \xi$. Its size is a measure of the downhill transport probability α.

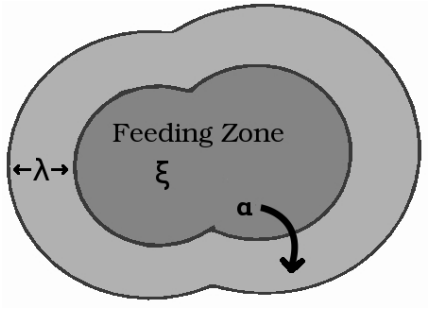


Figure 2.6: Layer with feeding zone ξ, λ-band and implied downhill tranpost α.

It is obvious that ξ is no physical parameter, but represents several atomistic parameters as e.g. the step-edge barrier (Ehrlich-Schwoebel barrier[26]), the diffusivity, the island shape. (See section 4.1 for a detailed study of the latter.)

With this parameter we can write the rate equations (2.4) as:

$$\frac{d\Theta_n}{dt} = \begin{cases} R_1(1-\Theta_1) + R_{n>1}(\Theta_1 - \xi_1) & \text{for } n = 1 \\ R_{n>1}(\xi_{n-1} - \xi_n) & \text{for } n > 1 \end{cases} \tag{2.5}$$

The change in coverage of a certain layer is simply proportional to the size of the corresponding feeding zone and growth rate R. The $n = 1$-case is treated separately, as there is obviously no downhill transport (i.e. no λ-band) for the first layer.

From a rigorous atomistc treatment [14, 18, 25], it can be shown that the feeding zone size of each layer is properly controlled by a *critical coverage* Θ_{cr}. That is the coverage for which nucleation on top of the layer begins (i.e. $\xi > 0$). It is given by [18]:

$$\xi_n = 1 - \exp\left[\sqrt{-\ln(1-\Theta_n)} - \sqrt{-\ln(1-\Theta_{n,cr})}\right]^2$$

for $\Theta_n \geq \Theta_{n,cr}$ and 0 otherwise.

2.4.2 The Cohen Model

For the model suggested by Cohen et al. in 1989 [19] the rate equations have a similar form:

$$\frac{d\Theta_n}{dt} = R_{n-1}\overbrace{(\Theta_{n-1}-\Theta_n)}^{1} - R_{n-1}\alpha_{n-1}\overbrace{(\Theta_{n-1}-\Theta_n)}^{3} + R_n\alpha_n\underbrace{(\Theta_n-\Theta_{n+1})}_{2}. \tag{2.6}$$

The relevant processes as depicted in fig. 2.7 are:

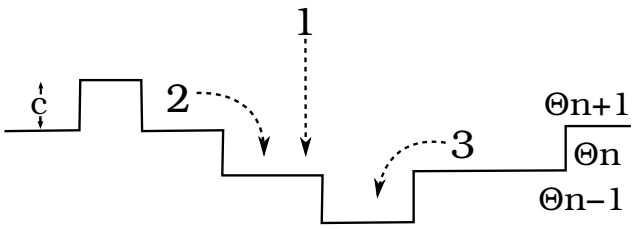


Figure 2.7: Possible ways for an incoming molecule to contribute to a certain layer, as described by eq. 2.6.

(1) The molecule lands and remains in layer (n). The probability of this process is proportional to the exposed coverage of layer $(n-1)$.
(2) The molecule lands in layer $(n+1)$ an diffuses downwards. The probability of this process is proportional to exposed coverage of layer (n).
(3) The molecule lands in layer n and diffuses downwards. The probability of this process is proportional to the exposed coverage of layer (n).
R is a growth rate for each layer, that entails the incoming flux and the adsorption probability.[19]

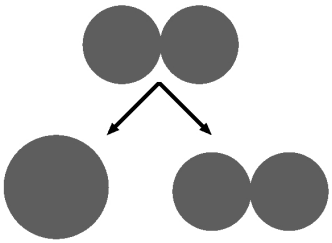
In this model downhill transport is controlled by the parameter α:

$$\alpha_n \propto \frac{d_n(\Theta_n)}{d_n(\Theta_n)+d_{n+1}(\Theta_{n+1})}. \tag{2.7}$$

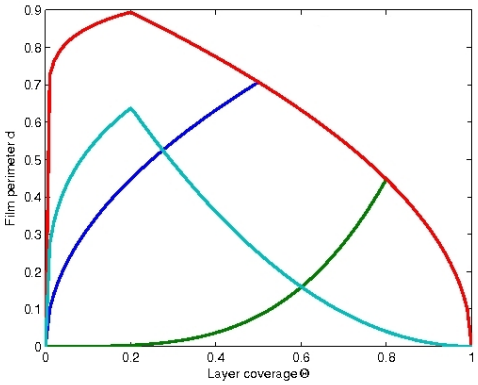
Here $d_n(\Theta)$ is the perimeter of layer n, i.e. a function of coverage and island form. This proportionality is easy to explain, as the likelihood of downhill transport is proportional to the length of possible step edges and inversely proportional to the number

of step edges on top of the respective layer, as those disturb the molecule diffusion.

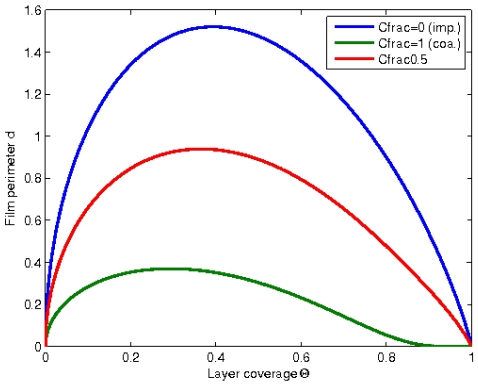
To model the perimeter Cohen suggested to treat the layer as built by same-sized clusters (for a less filled layer) and same-sized holes (for a more filled layer), with parameters accounting for the cluster shape and the coverage for the whole/cluster-transition [19]. This model is obviously unphysical as shown in figure 2.8(b).



(a) Schematic of rearrangement effects



(b) Perimeter according to Cohen



(c) Perimeter according to Tomellini

Figure 2.8: The corresponding perimeter according to Cohen (b) shows an unphysical unsteadiness, whereas we see no such behavior, but an expected strong increase in step edges for strong impingement occurs for the perimeter according to Tomellini (c). (The chosen parameters cover the whole parameter space.)

Based on the JMAK [1] statistical theory Woll[16] modified the Cohen Model according to Tomellini and Fanfoni[29]:

$$\begin{aligned} d_{imp}(\Theta) &= 2\sqrt{\pi N_0}(1-\Theta)\left[\ln\frac{1}{1-\Theta}\right]^{\frac{1}{2}} \\ d_{coa}(\Theta) &= \sqrt{\Theta(1-\Theta)}\,\exp\left[-\frac{\Theta}{2(1-\Theta)}\right], \end{aligned} \tag{2.8}$$

where d_{imp} is the perimeter for complete impingement and d_{coa} is the perimeter for complete coalescence. When two islands hit each other, possible effects are complete rearrangement (*coalescence*, 2.8(a) on the left-hand side) or no rearrangement at all (*impingement*, 2.8(a) on the right-hand side). The case of complete coalescence can be imagined like two drops of water on a glass surface, that hit each other and coalesce to one drop.

To scale between those cases the parameter C_{frac} is introduced:

$$d(\Theta) = C_{frac} \cdot d_{coa} + (1 - C_{frac}) d_{imp}. \tag{2.9}$$

$C_{frac} = 1$ means total coalescence and $C_{frac} = 0$ total impingement. Examples for the consequent perimeters, which lack any unsteadiness, are given in fig 2.8(c).

Other things being equal one could conclude, that C_{frac} has a strong impact on the growth behavior and expect a much smoother surface for low values (impingement). We will examine this hypotheses in section 4.1.

[1] Johnson-Mehl-Avrami-Kolmogoroff, see [27]. For a review by Tomellini and Fanfoni see [28].

3

Designing, Implementing and Using *GOfit*: a Python-Based Fitting Tool

3.1 Introduction to *GOfit*

In order to perform simulations we develop the python program *GOfit*, featuring a command-line interface. It is able to perform coverage and reflectivity simulations and data-fits according to the Trofimov-Model with the described modifications.

The heart of the simulation is the numerical integration of the rate eq. (2.5), which calculates the evolution of layer coverages from the parameters in the upper part of tab. 3.1. Therefore, the LSODA algorithm is used which employs the *Adam multistep method*[30] for non-stiff problems, as the given one.
In a second step the reflectivity is determined from the coverage by using the parameters in the lower part of tab. 3.1. This is done according to eq. 2.3.

Both functions are called by the fitting routine, which is based on the *ralg* algorithm.[1] This optimization algorithm for non-smooth problems is the default choice, but a

[1] http://openopt.org/ralg

large variety of other algorithms from the OpenOpt library [1] can be chosen as well [2]. *ralg* features box-like constraints and adaptive space dilation. It has been extensively tested with experimental data. Even with very imprecise initial conditions and broad constraints, fits as shown in fig. 3.1 are easily derived within a time of typically a few minutes. It is possible to show the χ^2 values and plots during the fitting procedure. For good initial conditions fits can be obtained much faster by fitting only a few parameters several times.

Furthermore, *GOfit* supplies multiple plotting functions based on the matplotlib[3] library with user-scalable fonts. Based on python's *regular expressions* functionality, parameter files and reflectivity or coverage data can easily be save to, or read from a file. A complete documentation of *GOfit* can be found in appendix C, parts of the source code are given in appendix D.

Parameter	**Description**
R_1	Initial growth rate $[\frac{ML}{s}]$
ΔR	Change in growth rate for 2nd layer $[\frac{ML}{s}]$
$\Theta_{cr,1}$	Critical coverage for nucleation of the top-layer
$\Theta_{cr,2}$	[in fractions of layer coverage]
$\Theta_{cr,N}$	
N_0	Factor for exp. scaling between $\Theta_{cr,2}$ & $\Theta_{cr,N}$
C_{frac}	Scales impingement/coalescence (Cohen-model only)
I_0	Scaling factor for the intensity
ρ_{sub}	Electron density: substrate
ρ_{int}	Electron density: interfacial layer
ρ_1	Electron density: 1st layer
$\vdots$	$\vdots$
ρ_N	Electron density: nth (last) layer
d_{int}	thickness of interfacial layer [in fractions of c]
d_1	thickness of 1st layer
$\vdots$	$\vdots$
d_N	thickness of nth layer
L	point in reciprocal space (in reciprocal lattice units, $L = c\frac{q}{2\pi}$)

Table 3.1: Parameters for simulation of coverage-evolution (upper part) and resulting reflectivity (lower part)

[1] `http://openopt.org/`
[2] See appendix C for more information
[3] `http://matplotlib.sourceforge.net/`

3.2 Exemplary Fit

To present the capabilities of the fitting algorithm an example is given in fig. 3.1. With a very broad estimation of initial parameters this result was obtained by fitting Θ_{cr}, $R1$ and Δ_R at first and then fitting only A and d, which both takes about thirty seconds with the default setup. Subsequently, all parameters at one were subject to fit, which takes about ten minutes. The small deviation from the anti-Bragg point ($L = 0.47$) is crucial for the reflectivity characteristics of the 2nd layer.

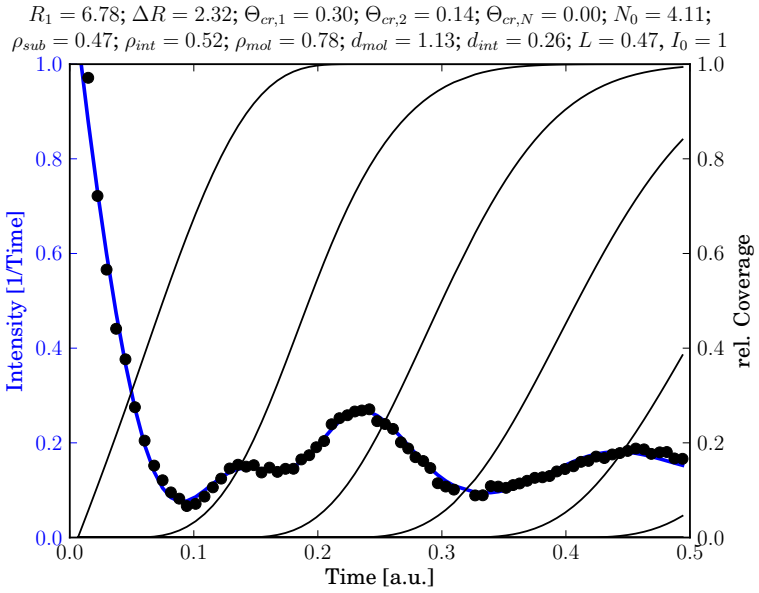


Figure 3.1: Exemplary fit for PDI-8CN2 on SiO_2 at 120°C (black dots)

This example also serves as proof that the algorithm is able to produce good results for more than ten parameters. The data- and parameter file can be found online[1][2].

3.3 Characteristics of Simulated Growth Oscillations

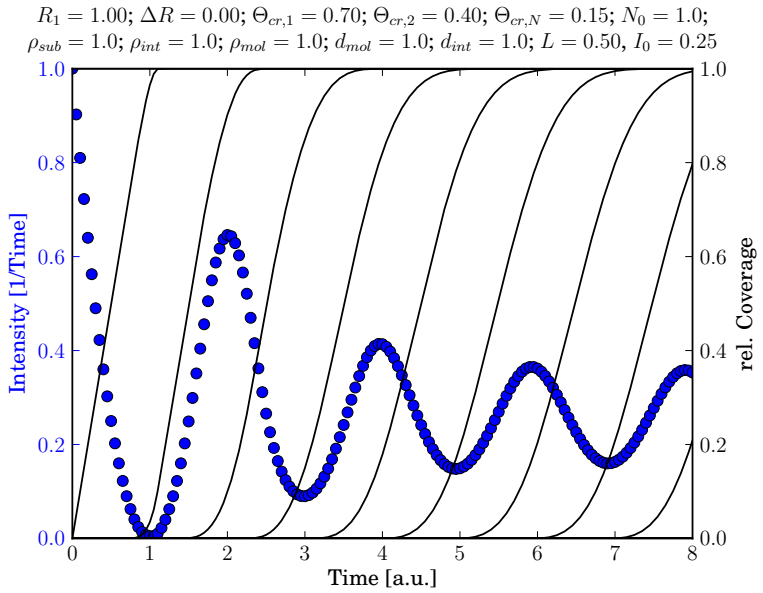
To study the behavior of reflectivity produced by *GOfit* we start from a typical parameter set at the anti-Bragg-point, without any distinct features as depicted in fig. 3.2.

[1] `http://people.physik.hu-berlin.de/~rukat/GOfit.rar`

[2] The measurements were conducted by F. Liscio, S. Milita, K. Broch, F. Schreiber and F. Biscarini at the ESRF beamline ID10B.

At $L = 0.5$ The first layer causes destructive interference, the second one constructive

Figure 3.2: Growth oscillations at the anti-Bragg-point.



and so forth. Thus, in phase space every 2nd layer obeys a phase of $\phi = \pi$, every other layer of $\phi = 0$. With such data the *growth rates* can be read off easily by the x-position of minima and maxima. Then again Θ_{cr} controls the height and sharpness of the peaks. It corresponds closely to the downhill transport (compare section 4.2 and appendix A). To sum up, the peak position, the peak height and the peak sharpness are the characteristics of such data. They are easily linkable with the growth morphology in terms of growth rate and critical coverage .

The most challenging part is to obtain information about the phase and from that information about layer thicknesses and q-point. In what follows we want to examine characteristics of reflectivity data under variation of the observation point in reciprocal space and the thickness.

3.3.1 Growth Oscillations for Varied q-Points

The reflectivity for the typical parameter set with altered q-point is shown in fig. 3.3.

We observe four basic differences:

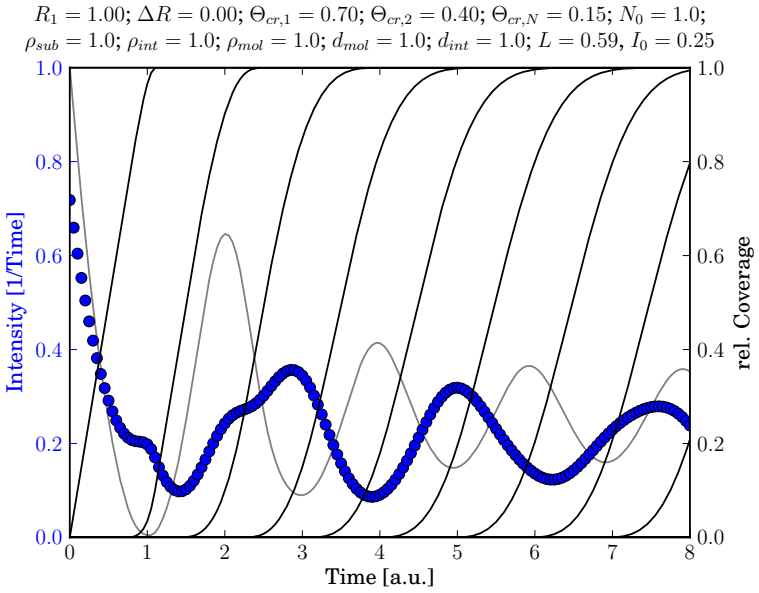


Figure 3.3: Growth oscillations at $L = 0.59$ with additional side-minimum and asymmetry. To be compared with the results for $L = 0.5$ (gray line).

1. The intensity is reduced.
2. The peaks are shifted.
3. The peaks are asymmetric.
4. Side maxima (shoulders) occur.

To understand this eq. (2.3) on page 8 has to be considered.

The reduced initial intensity (1) originates from the factor $\frac{1}{q^2}$. The peak behavior (2,3,4) can in principal be understood by considering the summation of the amplitudes with respect to the phase, as illustrated in fig. 3.4.

The blue arrow is the contribution of the substrate/interlayer, whereas the red-marked contribution of the growing layers moves back and forth during the growth of every other layer. This happens for phase angles of π and 0, respectively. Those angles are e.g. realized for $L = 0.5$ and layer thicknesses $d = 1$ as in fig. 3.2. The actual measured intensity is the square of the absolute value, i.e. the distance from the origin to the current point on the red line. Altering L scales all angles in the complex plain, which explains the shifting of the peaks (2) and their asymmetry (3).

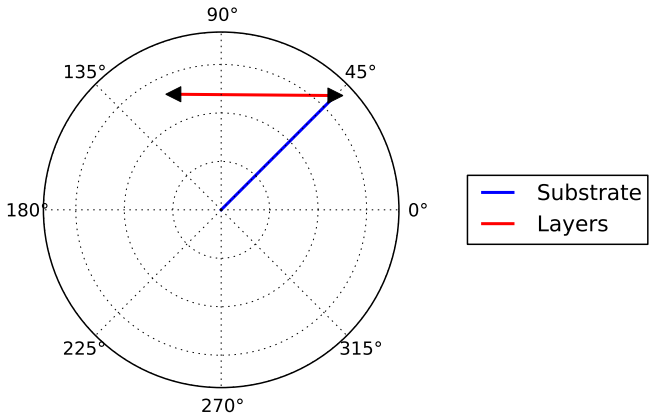


Figure 3.4: Reflectivity amplitudes in the complex plain. The contribution of the growing layers (red) changes its phase during the growth of every other layer.

The side extremes can be understood by considering fig. 3.4 once more. During growth the red contribution moves horizontal and features a minimum when it crosses the vertical (Im $= 0$) axes. We thus have side-extremes even though no layer is completed. These side-extremes can provide information about the current point in the complex plain as we will discuss in sec. 3.3.3.

The effect of an altered q-point in the whole q-space is shown in fig. 3.5. This is of particular interest as fast measurements at synchrotron sources allow to measure real-time-reflectivity for numerous q-points all at once. Further on, *GOfit* provides the capability for fitting such surfaces, as well.

3.3.2 Growth Oscillations for Varied Layer-Thicknesses

Altering the thickness of one or more layers causes effects similar to the altering of L, as both affect the angles in complex plain.

Usually ambiguities are evitable, because the lattice constant is a constant for most of the layers and deviations are usually observed for the first or last layer only. This is typically due to different growth behavior for the first layer on the substrate or for the top layer, which rearranges.

The behavior for a thickness deviation of a single layer is elucidated in fig 3.6.

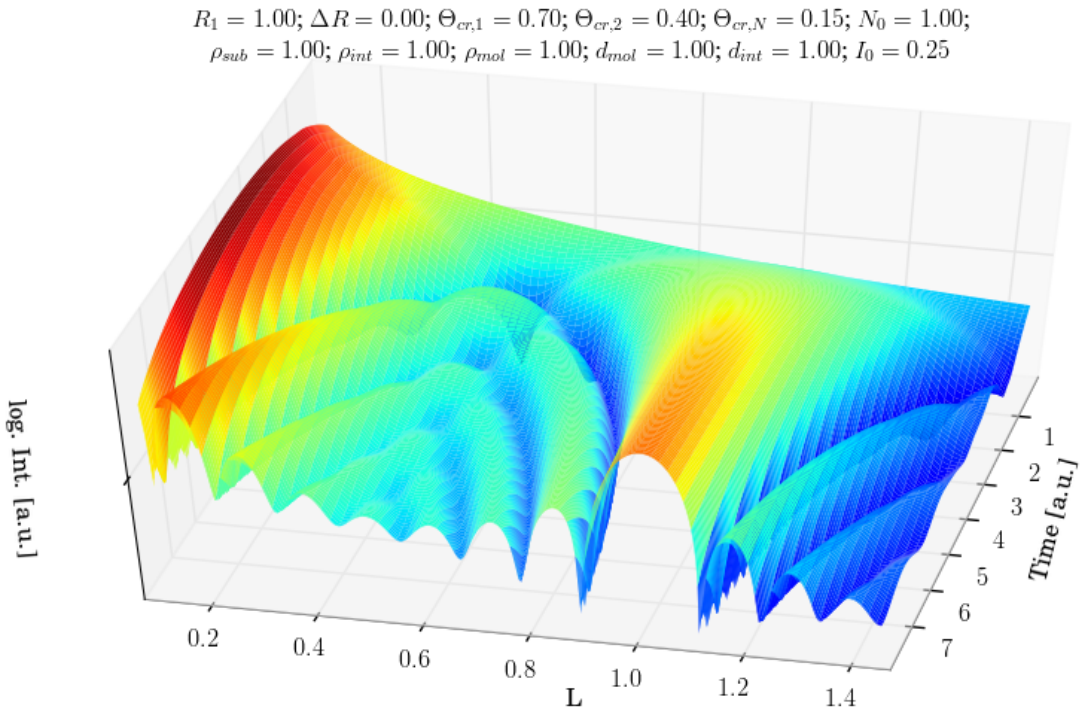


Figure 3.5: Intensity as function of time and observation point in reciprocal space L. If we neglect the dampening of L^{-2}, the reflectivity is symmetric along the $L = 0.5$-axis.

The parameters are the same as in fig. 3.2, but the thickness of the third layer is varied. Due to the change in phase, the contribution of the third layer points in a different direction in phase space therefore has a different contribution to the total intensity. The contributions of further layers thus have an altered starting point in the complex plain and their contribution to the intensity is changed, too.
Altering the thickness of a certain layer allows a complete rotation of the corresponding intensity distribution in the complex plain. This explains the periodicity of $d = 2$ at $L = 0.5$, that we observe in fig. 3.7.

E.g. at $L = 0.75$ the periodicity would be given for $d = \frac{4}{3}$, as eq. (2.3) yields periodicity for $2\pi \cdot L \cdot d = 2\pi n$ with $n \in \mathrm{N}$. In fig. 3.8 we show the reflectivity for an altered thickness of the 3rd layer on multiple q-points. This illustrates the rapid vanishing of overall regularities and intuitive understanding, that occurs if we come further away from our *typical* parameters.
As this is usually the case for experiments, it can be helpful to run such simulations beforehand to explore the reflectivity behavior, that is to be expected.

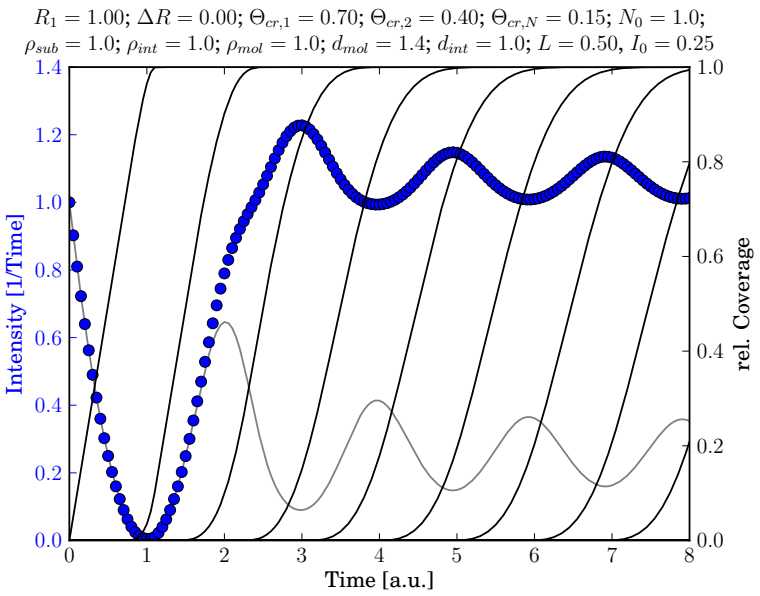


Figure 3.6: Example of typical parameters with altered thickness for the 3rd layer ($d_3 = 1.44$). To be compared with the reflectivity for our *typical* data ($d_3 = 1$, gray line).

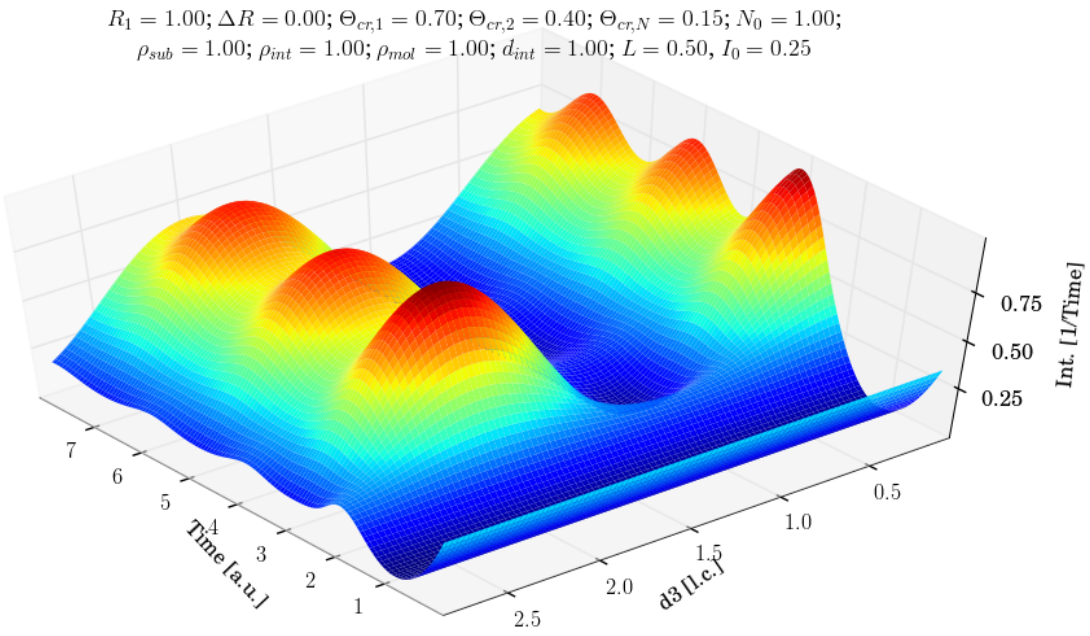


Figure 3.7: Intensity as function of time and thickness of the 3rd layer at L = 0.5.

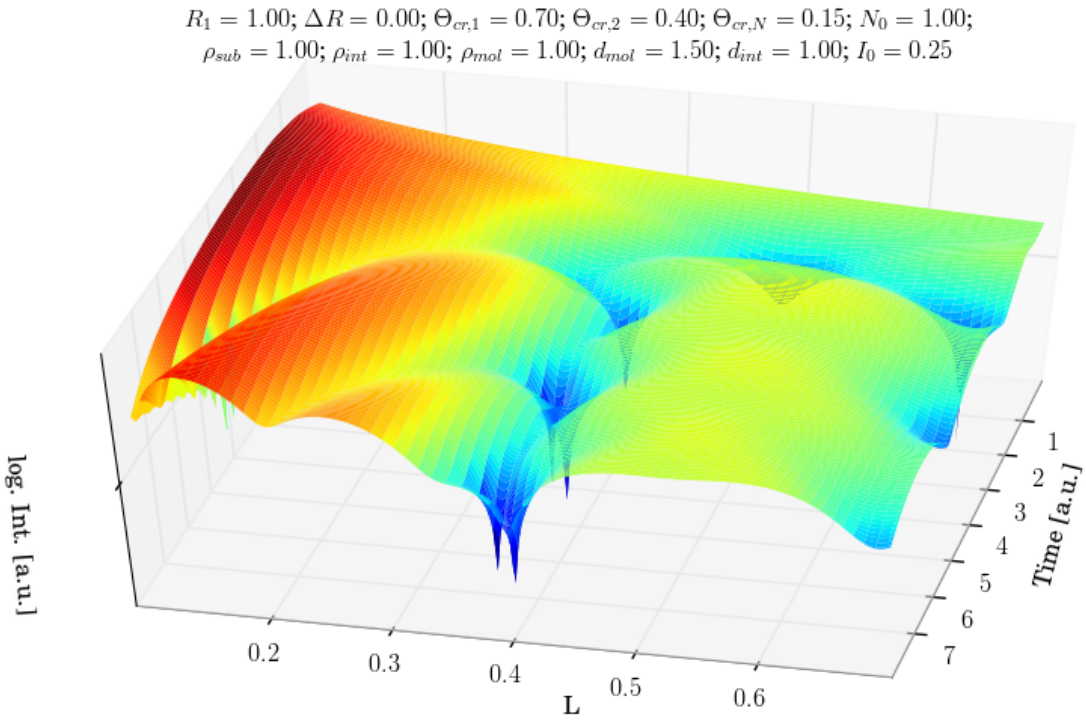


Figure 3.8: Intensity as function of time and observation point in reciprocal space (L) for an altered thickness $d3 = 1.5$

3.3.3 Determining the Phase

Once we understand the basic behavior of how our model-parameters influence the reflectivity, we want to discuss the more application-related task of figuring out information about the phase from reflectivity data.[1]

The midway through the layer's arrow in the complex plain can be roughly determined by the mean reflectivity between the corresponding minimum and maximum. The slope of the oscillations provides further information about the angle. If we consult fig. 3.4 for the first quadrant only. A smaller substrate phase would yield a higher slope for the oscillations, whereas a bigger phase would yield a smaller slope. This is true until the complex amplitude leaves the first quadrant and crosses the Im $= 0$ or the Re $= 0$-axis. In this case side-minima arise.

This case may provide further information about the phase of the contributing amplitudes. E.g. if we examine the substrate reflectivity and the reflectivity slope during

[1] It has been successfully shown, by means of the Trofimov model, that such analysis yields insights in the molecule orientation.[12]

layer growth, in a setup similar to the one depicted in fig. 3.4, the case of a substrate phase of $\frac{1}{4}\pi$ (1st quadrant) and a phase of the first layer of π, the second layer of 0 and so forth is hard to distinguish from a substrate phase of $\frac{5}{4}\pi$ (3rd quadrant) with the same layer-phase. If we have a side minimum we can solve this ambiguity easier, than with examining the reflectivity-slope. Especially as the slope is also affected by the growth rate, which the side-minimum is not.
For our example a side minimum during the growth of the first layer near the anti-Bragg point would clearly tell us, that we are in the 1st, not in the 3rd quadrant and have the corresponding phases for the substrate.

We conclude that the choice of the appropriate q-point can be an issue to achieve most information about growth.

4

Application of Growth Models

In this section we discuss the model's application on reflectivity data. The usually sparse number of distinct non-generic features that are given by typical time-resolved x-ray reflectivity measurements and the vast number of parameters used by the model cast doubt on their unambiguousness and thereby on reliability of those parameter. If one wants to draw conclusions from reflectivity fits e.g. regarding roughness or growth rate, such ambiguousness should be eliminated. At least their characteristics should be fully controlled.
We examine peculiarities of parameters, that are closest to *atomistic* phenomena: In section 4.1 the impingement/coalescence-parameter C_{frac} within the Cohen model is challenged by extensive fitting of reflectivity data for several organic semiconductors on different SAMs. In section 4.2 downhill transport is discussed.

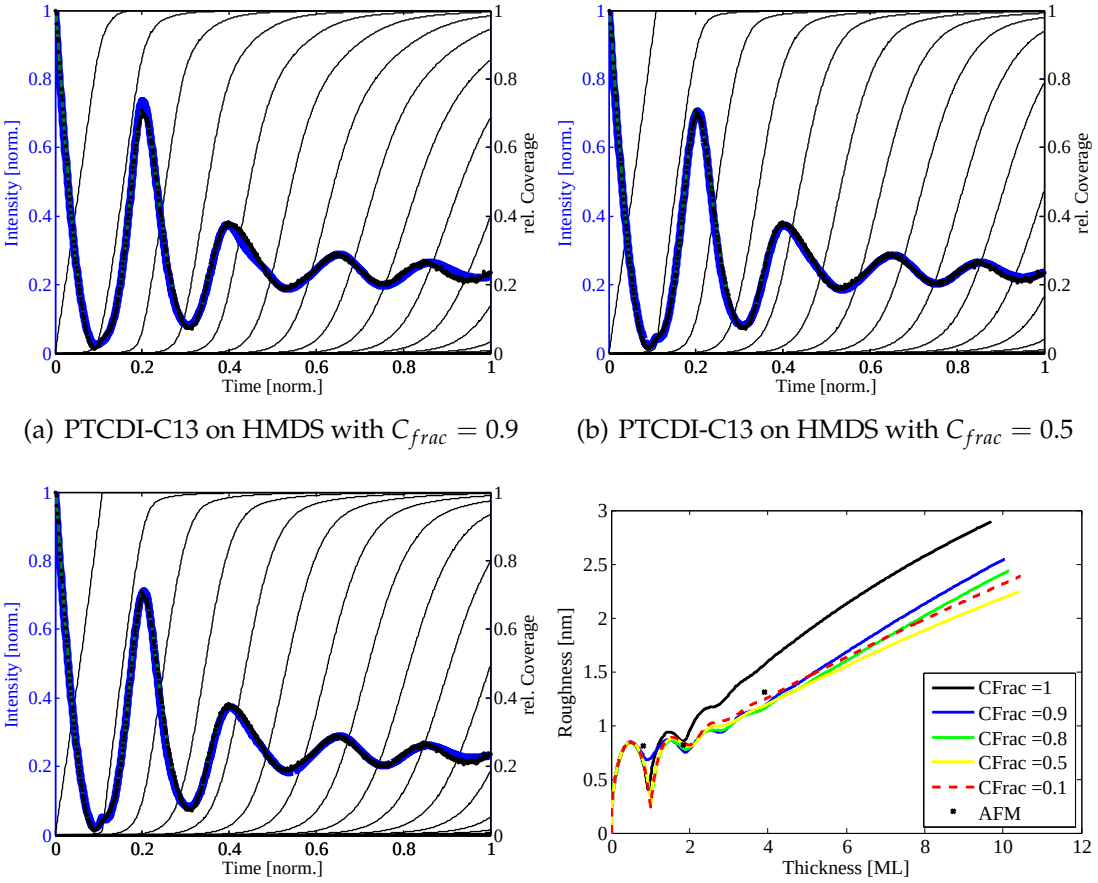
4.1 Impingement and Coalescence

Fits for more than 250 real-time reflectivity data sets, recorded during the growth of the organic semiconductors DIP[1] and PTCDI-C13[2] on different SAMs (PHTS, HMDS, NMTS) and on bare Si02, were performed by least square fitting. The reflectivity and AFM measurements were undertaken at the G3 station at the Cornell Higher Energy

[1] Diindenoperylene
[2] N,N'-ditridecyl-3,4,9,10-perylenetetracarboxylic diimide

Synchrotron Source (CHESS) by the group of J. Engstrom. Both introduced models were deployed and extensive variation of the C_{frac}-parameter within the Cohen-Model was realized. The given figures and results are representative for all the data analyzed.

First of all nearly *all* data could be reproduced by every model and by every fixed C_{frac} value (for $0.1 < C_{frac} < 1$) with satisfying χ^2-values and a variation in growth rate/thickness prediction smaller than 3%. Typical examples for the growth of PTCDI-C13 on HMDS are for different C_{frac}-values are shwon in fig 4.1



(a) PTCDI-C13 on HMDS with $C_{frac} = 0.9$

(b) PTCDI-C13 on HMDS with $C_{frac} = 0.5$

(c) PTCDI-C13 on HMDS with $C_{frac} = 0.1$

(d) Roughness predictions for PTCDI-C13 on HMDS for different C_{frac}-values.

Figure 4.1: Effects of varied impingement/coalescence behavior.

All models fit the reflectivity data, but predict diverse layer coverages and smoothness of growth. As expected the growth is more layer-by-layer wise for impingement , i.e. low C_{frac}-values. This can be seen by comparing fig. 4.1(b) with 4.1(a). The s-curves

of the former, that depict the layer coverage, exhibit considerably more steepness than those of the latter, especially for high coverages. That means the growth is more layer-by-layer wise.
Concerning the thickness predictions all C_{frac} values yield the same results, which agree with match AFM-data (not shown).
Of greater interest are the predictions in roughness. (In appendix B the equations for calculating the rms roughness are presented). As shown in fig. 4.1(d) the model's predictions behave accordingly to our expectations in terms of more impingement yielding smoother growth and lower roughness for $C_{frac} > 0,5$.
One might conclude, that the impingement/coalescence behavior controls the growth-mode and smoothness, but has no effect on the actual reflectivity, i.e. all its impacts on the reflectivity are smoothed out by other parameters. This would be bad news, as it would make it impossible to obtain information concerning the growth mode and roughness from reflectivity data within the Cohen-model.

But surprisingly there is a further observation: C_{frac} values lower than 0.5 (in this example 0.1), that means hardly any island-rearrangement, do not fulfill our expectations, as they yield again a higher roughness. The same behavior is observed for the s-curves in fig. 4.1(c) and in more detail in fig 4.2.

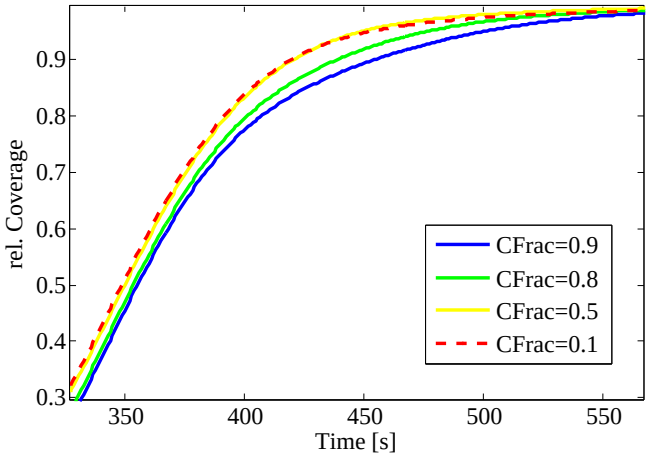


Figure 4.2: Coverage evolution of the fifth layer for various C_{frac}-values.

The coverage evolution of the fifth layer exemplary shows, that more impingement yields more layer-by-layer growth. Though the red dashed line runs contrary to this. $C_{frac} = 0.1$ yields roughly the same s-curve as $C_frac = 0.5$ does, even though it features a much higher step-edge density. There are three conclusions we can take from

this:

1. C_{frac} is clearly not accessible by fitting of time resolved reflectivity data from a single point in reciprocal space.[1]. Concerning reflectivity it is strongly correlated to other system parameters. These can entirely compensate the effect of an altered C_{frac}-value on the reflectivity.

2. Nevertheless C_{frac} is a system parameter with distinct behavior. With regard to roughness and growth-mode it can not be compensated by other model parameters.

3. Values smaller than 0.5 are presumably physically wrong, as they yield inexplicable behavior.

4.2 Downhill Transport

A more promising attempt to gain knowledge about atomistic growth parameters by means of reflectivity data is to choose a broader parameter, e.g. the *interlayer-transport parameter* α (compare eq. (2.7)) accounts for downhill/uphill-transport as well as for desorption and thus for the rearrangement behavior C_{frac}, too. As downhill-transport is by far the most probable process, we refer to α as *downhill-transport parameter*.

If we assume the growth rate R to be constant and neglect any interlayer transport the rate equations for layer coverage (2.6), that were introduced within the Cohen-Model (Section 2.4.2) become

$$\frac{d\Theta_n}{dt} = R(\Theta_{n-1} - \Theta_n). \tag{4.1}$$

[1]This might be seen as an endorsement for the warning Tomellini and Fanfoni stated in their review of the JMAK-theory [28] and its applications.

> Although many theoretical investigators have given a fundamental contribution to extend the range of applicability of the model, the experimentalists keep using the model in its original form, owing to its simplicity, at times risking to overinterpret the experimental results.

This yields the analytical solution

$$\Theta_n(t) = 1 - e^{-Rt} \sum_{k=0}^{n-1} \frac{(Rt)^k}{k!}, \tag{4.2}$$

which is depicted in fig. 4.3.

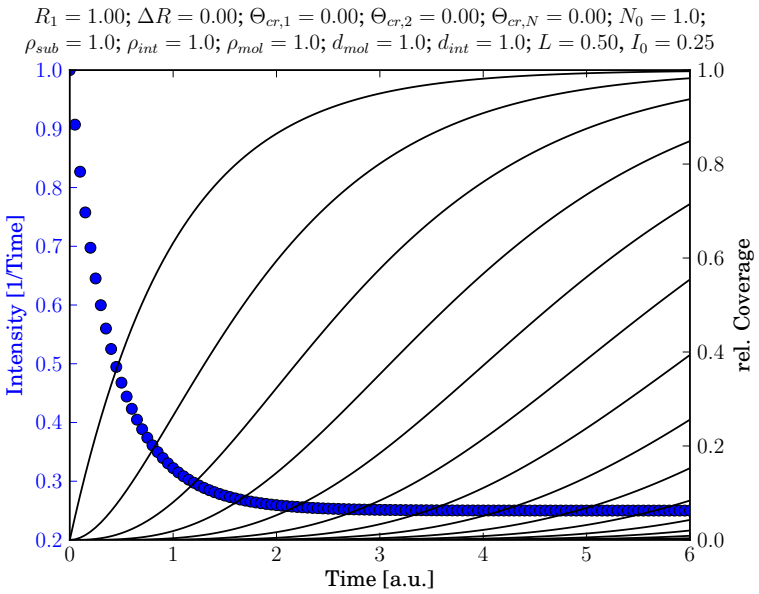


Figure 4.3: Layer coverage without interlayer transfer ($\Theta_{cr,1} = \Theta_{cr,2} = \Theta_{cr,n} = 0$) and with constant growth rate. .

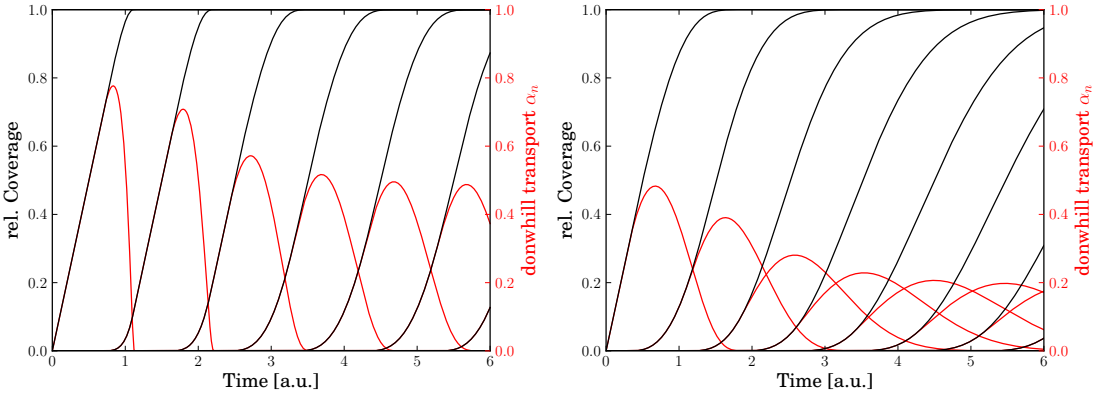
If downhill-transport is not considered even higher layers start to nucleate immediately, which is why no oscillations are observed.
Any deviation from that coverage evolution is then due to downhill transport. For known growth rates the downhill transport might therefore be calculated directly from a given set of layer coverage data, as we show in appendix A.

In the Trofimov model α is given as size of the λ-band:

$$\alpha_n = \lambda_n = \Theta_n - \xi_n \tag{4.3}$$

α_n accounts for the fraction of incoming molecules, that are transported downwards to each layer n. Some typical examples are given in fig. 4.4.

We note that α decreases smoother for a smoother corresponding layer growth (4.4(b)).



(a) A more LBL-growth setup with $\Theta_{cr,1} = 0.7$, $\Theta_{cr,2} = 0.6$, $\Theta_{cr,N} = 0.3$, $N_0 = 1$

(b) A more SF-growth setup with $\Theta_{cr,1} = 0.3$, $\Theta_{cr,2} = 0.2$, $\Theta_{cr,N} = 0.05$, $N_0 = 1$

Figure 4.4: Examples for downhill transport and layer coverage. (Parameters other than the critical are taken from the *typical* parameter set in sec. 3.3.)

If we assume downhill transport to be the only interlayer-activity, nothing but α accounts for the smoothness/roughness of growth. The total number of molecules that diffuse downwoards is given by the integral of α over time.

As mentioned, α is a broad parameter insofar as it covers, among others, the role of the perimeter size and therewith the rearrangement behavior. This is approved by comparison of its shape in fig 4.4 to the island perimeters given in fig. 2.8(c) on page 12 and eq. (2.8). This comparison moreover affirms, that more impingement yields a more LBL-growth (i.e. higher critical coverages).
Determining the perimeter by AFM studies and comparing it with the downhill transport predictions arises as a pending task.

5

Summary and Outlook

We sum up our results in the chronology they were presented in the introduction.

1. The python program *GOfit* to simulate and fit in-situ x-ray reflectivity during thin film growth has been implemented and utilized. It uses growth models in order to examine the reflectivity under variation of any model parameter. The program features high flexibility and a convenient command-line usability. Multiple fit-algorithms are applicable, featuring box constraints and leading to satisfying results, even by fitting all system parameters at once and with inaccurate initial parameters. A framework for comfortable graphical output is provided. This work enables us to analyze x-ray reflectivity data from in situ growth with unrivaled convenience, flexibility and accuracy.

2. Such analysis of growth oscillations leads to knowledge about growth rates, layer thicknesses and roughness evolution. Especially to obtain phase information, i.e. information about the thickness or molecule orientation of certain layers the choice of the appropriate observation point in reciprocal space has been shown to be of importance.

3. The analysis of impingement and coalescence revealed limits of insights into atomistic properties by reflectivity data and suggests to use broader parameters, as it has been done for the downhill transport α.
   α exhibits a shape that is similar to the introduced step edge density (based on JMAK theory). It combines further parameters like the intra- and interlayer dif-

fusivity to one, unambiguous parameter, suitable for fitting experimental data.

In-situ x-ray reflectivity during organic thin-film growth and its extensive analysis have been shown to be an appropriate tool to investigate the growth behavior of organic molecules on scales down to atomistic processes, up to a certain extent. *GOfit* is an important step to exploit this extent.

Proposed improvements in *GOfit* are the more straightforward modification of $\Theta_{cr,n}$ and d_n for arbitrary layers n and the computation and investigation of parameter correlations.

For future work it seems promising to model the coverage evolution and the downhill-transport on atomistic scales to reproduce the behavior shown in section 4.2 with insightful parameters like the perimeter or the Ehrlich-Schwoebel barrier [26]. Random walk simulations could be deployed.
On behalf of experiments a comparison of the results with AFM data could be insightful.

Appendix A

Calculating Downhill Transport from Layer Coverage

Suppose one has given layer coverages $\Theta_n(t)$, from a reflectivity fit. We want to give a simple approach to calculate the downhill transport α_n that describes the transport rate of molecules from layer $(n+1)$ into layer n.

The crucial fact for this calculation is the simultaneous growth of many layers. During the beginning growth of layer n typically layer $(n-2)$, $(n-1)$ are still growing, whereas layer $(n+1)$, $(n+2)$, etc. are already growing while layer n is not finished. To determine α we will compare two stages of growth for every layer. Stage a for a certain layer n is the time span where this layer is growing but layer $(n+1)$ is not growing yet, time points from this stage will be denoted as t_n^a and respective coverages as $\Theta_n(t_n^a) =: \Theta_n^a$. Stage b is the time span, between the beginning growth of layer $(n+1)$ and the finishing of layer n (denoted accordingly: $\Theta_n(t_n^b) =: \Theta_n^b$). This approach aims to compare the growth rate $\dot{\Theta}_n^a$ (without downhill transport into layer n, as there is no layer $(n+1)$) with the growth rate $\dot{\Theta}_n^b$, that also accounts for downhill transport.

With the obvious idea, that molecules have to land on top of layer $(n-1)$ to contribute to layer n, and that a higher coverage of layer n lowers the probability for an incoming

molecule to contribute to that layer, one denotes:

$$\dot{\Theta}_n^i(t) = R_n \cdot \Theta_{n-1}(t) \cdot (1 - \Theta_n(t)) + \beta_n(t) \tag{A.1}$$

R_n is a proportionality factor, that describes a hypothetical growth rate layer n would have with a completely filled layer $(n-1)$ and a non-existing layer $(n+1)$. Thus any change between the growth rates in different stages $\dot{\Theta}_n^a$ and $\dot{\Theta}_n^b$, which can not be explained by the coverages of vicinal layers as given in eq. A.1, is described by $\beta_n(t)$. With a few reasonable assumptions[1] $\beta_n(t)$ is a rate accounting for downhill-transport. It is given by

$$\beta_n = \alpha_n - \alpha_{n-1} \tag{A.2}$$

With this expression it is not clear how to distinct between transport into layer n (α_n) and transport from layer n into layer $(n-1)$ (α_{n-1}).
This problem is solved by iterating over all layers beginning with the first one as described in more detail in section A.1. There is no downhill transport α_0 down from the first layer[2]. Thus $\alpha_1 = \beta_1$ can be determined. With α_1, it is possible to derive α_2 from β_2 and iteratively any α_n as described in section A.2.

A.1 Transport from the 2nd to the 1st Layer

Starting point is eq. A.1:

$$\dot{\Theta}_n(t) = R_1 \Theta_{n-1}(t) \cdot (1 - \Theta_n(t)) + \alpha_n(t) - \alpha_{n-1}(t) \tag{A.3}$$

For the first layer's growth stage eq. (A.3) becomes:

$$\begin{aligned} \dot{\Theta}_1^a &= R_1 \underbrace{\Theta_0(t)}_{=1} \cdot \underbrace{(1 - \Theta_1(t))}_{\neq 1} + \underbrace{\beta_1}_{=0} \\ &= R_1(1 - \Theta_1(t)) \end{aligned} \tag{A.4}$$

[1] Uphill steps, multiple downhill steps and desorption are neglected. Different sticking coefficients are covered by different rates R_n.

[2] This is evident from the $\dot{\Theta}_1(t_n^1) = R_1 (= \text{const.})$, which is only true for the first layer.

That yields the analytical solution[1]

$$\begin{aligned}\Theta_1^a(t) &= -e^{-R_1 t} + 1 \ (\neq \text{const}) \\ \dot{\Theta}_1^a(t) &= R_1 e^{-R_1 t},\end{aligned} \tag{A.5}$$

which allows to determine R_1 by fitting the expression to the given data Θ_1^a. Alternatively one may use the series expansion

$$-e^{-R_1 t} = \sum_{i=0}^{\infty} \frac{(-R_1 t)^i}{i!}.$$

This yields

$$\Theta_1^a \approx R_1 t \tag{A.6}$$

and is valid for small t, as usually given.

For stage b follows

$$\begin{aligned}\dot{\Theta}_1^b &= R_1 \underbrace{\Theta_0(t)}_{=1} \cdot \underbrace{(1 - \Theta_1(t))}_{\neq 1} + \underbrace{\beta_1}_{=\alpha_1} \\ &= R_1 \cdot (1 - \Theta_1(t)) + \alpha_1(t).\end{aligned}$$

Obviously α is given by the difference of the growth rates

$$\alpha_1(t) = \dot{\Theta}_1^b(t) - \dot{\Theta}_1^a(t) \tag{A.7}$$

and can be calculated by comparing the coverages:

$$\begin{aligned}\Delta\Theta_1 =& \Theta_1^b(t) - \Theta_1^a(t) \\ & \Theta_1^b(t) - (-e^{-Rt} + 1) \\ \alpha_1(t) =& \Delta\dot{\Theta}_1(t).\end{aligned} \tag{A.8}$$

[1]Further calculations have to be done numerically, otherwise it will be explicitly mentioned.

A.2 Transport from Layer $(n+1)$ to Layer n

Examining $\dot{\Theta}_n$ for $n > 1$ one notes, that there is also downhill transport from layer n into layer $(n-1)$ influencing the growth. Eq. A.1 becomes:

$$\begin{aligned}
\dot{\Theta}_n^a =& R_n \cdot \underbrace{\Theta_{n-1}(t)}_{\neq 1} \cdot \underbrace{(1-\Theta_n(t))}_{\neq 1} + \underbrace{\beta_n(t)}_{=-\alpha_{n-1}(t)} \\
=& R_n \cdot \Theta_{n-1}(t) \cdot (1-\Theta_n(t)) - \alpha_{n-1}(t) \\
\rightarrow R_n =& \frac{\dot{\Theta}_n^a + \alpha_{n-1}(t_n^a)}{\Theta_{n-1}(t_n^a)(1-\Theta_n(t_n^a))} \ (\stackrel{!}{=} \text{const}).
\end{aligned} \tag{A.9}$$

For stage b follows:

$$\begin{aligned}
\dot{\Theta}_n^b =& R_n \cdot \Theta_{n-1}(t) \cdot (1-\Theta_n(t)) + \beta_n(t) \\
=& R_n \cdot \Theta_{n-1}(t) \cdot (1-\Theta_n(t)) + \alpha_n(t) - \alpha_{n-1}(t).
\end{aligned}$$

That yields:

$$\alpha_n(t_n^b) = \dot{\Theta}_n^b + \alpha_{n-1}(t_n^b) - R_n \cdot \Theta_{n-1}(t_n^b)(1-\Theta_n(t_n^b)). \tag{A.10}$$

Here the expressions, that are functions of t, have to be evaluated only in the appropriate time stage a or b for the appropriate layer n. With recursion of eq. A.10 α can be determined up to any layer.

Appendix B

Calculating the RMS Roughness from Layer Coverage

We briefly describe our approach to calculate the film's rms roughness.

We imagine the film to consist of discrete boxes like Lego bricks as depicted in fig. B.1. The dashed line is the mean thickness, which can be written as

$$\langle thick \rangle = \sum_n n(\Theta_n - \Theta_{n+1}). \tag{B.1}$$

Here every addend accounts for the *exposed* coverage of one layer and is multiplied

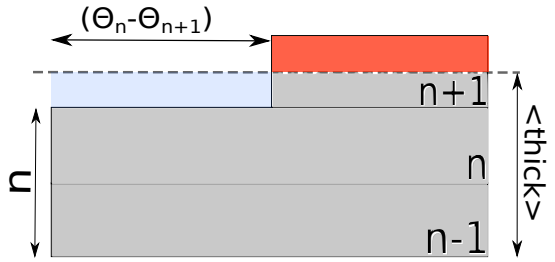


Figure B.1: This very simple example of a layer coverage setup illustrates the rms roughness. The blue and red boxes are the deviation from the mean, with height $|\langle thick \rangle - n| = \frac{1}{2}$ and $|\langle thick \rangle - (n+1)| = \frac{1}{2}$ respectively.

by the number of (filled) layers underneath it.
To calculate the rms roughness we sum up the squares of each *exposed* layer's height-

deviation from the mean thickness. In other words: we sum up the squares of the height of the bricks, that have to be added (removed) to fill (to empty) each particular layer up to the mean thickness. This is illustrated in fig. B.1.

The height of such a brick is the difference of the mean thickness and the thickness of the respectively exposed layer:

$$\Delta height = |n - \langle thick \rangle| \tag{B.2}$$

Eventually we sum up the squares of all $\Delta height$s and weight each with the the corresponding width. This weighting accounts for the likelihood to measure a certain thickness for a random measurement:

$$\begin{aligned} var &= \sum_n \underbrace{(\Theta_n - \Theta_{n+1})}_{width} (\Delta height)^2 \\ &= \sum_n (\Theta_n - \Theta_{n+1}) \left(n - \sum_n n \, (\Theta_n - \Theta_{n+1}) \right)^2 \end{aligned} \tag{B.3}$$

This sum can be understood as variance, thus its square root is the standard deviation of the surface height from its mean and the sought rms roughness.
With Θ_0 representing the surface ($\Theta_0 = 1$) and $n_0 = 0$ it can easily be shown, that this equals the following - easier to compute - equation:

$$\sum_n (\Theta_n - \Theta_{n+1}) n^2 - t^2. \tag{B.4}$$

Appendix C

GOfit: Documentation

C.1 Introduction

GOfit is a python program, which performs simulations and fits of thin film growth and x-ray reflectivity using the growth model described in section 2.4.1. Reflectiviy amplitudes are calculated according to eq. (2.3).

It features a convenient command-line based user interface. Due to its object-oriented design the program-code is easy to change and adding new features or modifying the model is simple. Python offers a very good readability and as interpreted language a good cross-platform compatibility. *GOfit* consists of two files *GOfit.py* and *GOfitBib.py* and is written with the purpose to be easy to use, easy to modify and easy to extend. The complete source code can be found here: `people.physik.hu-berlin.de/~rukat/GOfit.rar`, parts of it are given in appenix D

The program is tested and developed with python[1] 2.6.6 and utilizes numpy 1.3.0, scipy[2] 0.7.2, OpenOpt[3] 0.34 and matplotlib[4] 0.99.

[1] `www.python.org`
[2] `numpy.scipy.org`
[3] `www.openopt.org`
[4] `pypi.python.org/pypi/matplotlib`

C.2 Getting Started

A list of all model parameters is given in tab. 3.1 on page 16.

To start the programm the *GOfit* needs to be imported, e.g. by typing in your python command line:

```
>>> from GOfit import *
```

To perform an analysis or a fit an instance of GOfit with arbitrary name (in this example *mysim* is used) has to be created. A set of experimental data (two columns text file) is taken as optional argument:

```
>>> mysim = GOfit('datafile.txt')
```

All further commands are realised as built-in methods of this class. To guarantee convenient parameter handling all variables are attributes, global within instance.

C.3 Data Fitting

The main purpose of the program is to fit time resolved experimental reflectivity data to the introduced models. Parameters can arbitrarily be chosen as constant (`par.fit=False`) or as fit-parameters (`par.fit=True`) with simple constraints (`par.limits`=[lower contraint, upper constraint]). Various optimization algorithms are provided[1], *ralg* from OpenOpt library [2] is the default.

`mysim.fit` takes `maxIter=int`, `maxFunEval=int` and `solver='string'` as optional arguments (`1e3`, `1e5` and `'ralg'` are default, respectively). Recommended optimization algorithms are:

- `ralg` - Based on an r-algorithm with adaptive space dilation.
- `pswarm` - Combines pattern search and particle swarm (see [31]).

Further testing the numerous further optimization algorithms should be undertaken.

[1] See code and comments for details
[2] `OpenOpt.org`

C.4 Examples and Summary

An example datafile and corresponding parameter file is provided online[1] and discussed in section 3.2 on page 17. A typical worksheet could look like this:

```
>>> mysim = GOfit('datafile.txt')
>>> mysim.read('params.txt')
>>> mysim.parsPrint()
R1       = 0.01   limits=[0,None] fit:False
deltaR   = 0.00   limits=[-.1,.1] fit:False
thCr1    = 0.80   limits=[0,0.99] fit:False
thCr2    = 0.60   limits=[0,0.99] fit:False
thCrN    = 0.30   limits=[0,0.99] fit:False
N0       = 0.10   limits=[0,None] fit:False
ASub     = 1.00   limits=[0,None] fit:False
AInt     = 2.00   limits=[0,None] fit:False
AMol     = 1.00   limits=[0,None] fit:False
dMol     = 1.00   limits=[0,None] fit:False
dInt     = 0.50   limits=[0,None] fit:False
L        = 0.50   limits=[0,1]    fit:False
I0       = 0.01   limits=[0,None] fit:False
>>> mysim.I0.value = 0.02
>>> mysim.calcCov()
>>> mysim.calcInt
>>> mysim.plotSim(plotPar=True)
... Graphic output ...
```

Model parameters, their fitting boundaries and behaviour are implemented each as a atttribute of the parameter class. They are acessable as shown in the following example:

```
>>> mysim.R1.value
0.01
>>> mysim.R1.fit
False
>>> mysim.R1.limits
[0, None]
>>> mysim.L.value = 0.3
>>> mysim.fit(maxIter=500)
```

[1] http://people.physik.hu-berlin.de/~rukat/GOfit.rar

The last command starts the fitting routine. The following example shows how to load and save parameters and simulated data:

```
>>> mysim.save('parameter.parms')
Saved parameters to file.
>>> mysim.load('parameter.parms')
Read parameters from file.
>>> mysim.simSave('intensity.txt','coverage.txt')
Saved intensity and coverage to file
```

A summary of all available commands is given in table C.1.

command	abbr.	description
`fit(method)`	-	fit int to exp. data
`calcCov()`	`cc()`	calc. cov. array: `mysim.cov`
`calcInt()`	`ci()`	calc. intensity: `mysim.int`
`calfSim()`	`cs()`	calc. cov. and int.
`calcFit()`	`cf()`	fit int. to datafile
`plotCov()`	`ps()`	plot coverage
`plotInt()`	`pi()`	plot intensity
`plotSim()`	`ps()`	plot cov. and int.
`plotExp()`	`pe()`	plot exp. data
`plotAlphaCov()`	`pac()`	plot cov. and downh. coeff.
`plotFit()`	`pf()`	plot cov., int., exp. data
`setTime(ti,tf,step)`	`st()`	set t_i and t_f manually
`parsPrint('datafile')`	`pp()`	print parameters
`parsLoad('datafile')`	`load()`	load parameters from file
`parsSave('datafile')`	`save()`	save paramters to file
`simSave('int','cov')`	`ssave()`	save simulation to file
`par.value`	-	access parameter value
`par.limits`	-	access fit limits
`par.fit`	-	access fit value (boolean)
`par.parsPrint()`	`par.pp()`	print selected paramter
`chiSquare()`	`chis()`	print χ^2

Table C.1: Overview of GOfit methods . Every fit parameter (as listed in tab. 3.1) is realised as instance of the subclass `par`.

All plot commands take the optional keyword argument `plotPar`, which equals either `True` to print the model parameters in the title or `False` (default). The arguments of `setTime` are required, the user will be asked for appropriate parameters, anyway. Further commands and explanation is provided in the programm-code (appendix D).

Appendix D

GOfit: Source Code

We show parts of the program code, including all function definitions, except the lengthy plot functions. The program consists of the main module *GOfit.py*, with definitions of the parameter class *par*(appendix D.1 line 63) and the main class *GOfit* (appendix D.2 line 83).
It is completed, by the additional library module *GOfitBib.py*. It contains the set of differential equations for calculating the coverage evolution (appendix D.2 line 76), the function that determines the intensity from the coverage data (appendix D.2 line 22) and the function that determines the downhill transport.

D.1 *GOfit.py*

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  #
4  #       GO-Fit.py
5  #
6  #               Version 0.91 (2011-09-20)
7  #
8  #       Copyright 2011 Tammo Rukat <tammorukat@gmail.com>
9  #
10 #       This program is distributed in the hope that it will be useful,
11 #       but WITHOUT ANY WARRANTY; without even the implied warranty of
12 #       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
13 #
14 #       To be done:
15 #       - Interrupt fitting function by user input (and store parameter)
16 #       - More convenient handling of setting the fit values
17 #       - Add easier handling for arbitrary thickness and ThCr modifications.
18 #       - Correlation matrix
19 #
```

```
print\
'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\
                \n This is GOfit 0.9\
                \n by Tammo\
                \n rukat@physik.hu-berlin.de\
                \n \n To get started try e.g.:\
                \n \t >>> test=GOfit()\
                \n \t >>> test.st(0,500,1)\
                \n \t >>> test.ps()\n\
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'

#~ Import essential modules
import scipy
import numpy
import pylab
import sys, re
from scipy import integrate
from scipy import optimize
from scipy import inf
import matplotlib.pyplot as plt

#~ Import further optimization tools (optional)
from pswarm_py import pswarm
#~ from FuncDesigner import interalg

# Import optimization tools
from openopt import NLP, GLP, NSP

#~ Import dedicated library
import GOfitBib

#~  Set general font properties for plots
plt.rcParams['ps.useafm'] = True
plt.rcParams['pdf.use14corefonts'] = True
plt.rcParams['text.usetex'] = True

font = {'family' : 'serif',
                'size' : '12'}
        #~ 'weight' : 'bold',
        #~ 'size'   : 'larger'}
plt.rc('font', **font)


class par:
        ''' Class to store system variables, their constraints,
                fitting behaviour, name and LaTex-name. '''
        def __init__(self, name='unnamed', texname='unnamed',\
                                value=0, fit=True, limits=[None,None]):
                self.name = str(name)
                self.texname = texname
                self.value = value
                self.fit = fit
                self.limits = limits

        def parPrint(self):
                ''' Print parameter information '''
                print self.name,'=',self.value, '\t fit = ',self.fit,\
                        '\t limist = ',self.limits

        def pp(self):
                ''' Abbr. for parPrint() '''
                self.parPrint()

class GOfit:
        ''' Main class for modelling and fitting '''
        def __init__(self,datafile=0):

                #~ Intialize fit parameters
                self.R1 = par(name='R1',texname='R_1',\
```

```
89                              value=0.01,fit=False,limits=[0,None])
90              self.deltaR = par(name='deltaR',texname='\Delta_R',\
91                              value=0.001,fit=False,limits=[None,None])
92              self.thCr1 = par(name='thCr1',texname='\Theta_{cr,1}',\
93                              value=0.6,fit=False,limits=[0,0.99])
94              self.thCr2 = par(name='thCr2',texname='\Theta_{cr,2}',\
95                              value=0.6,fit=False,limits=[0,0.99])
96              self.thCrN = par(name='thCrN',texname='\Theta_{cr,N}',\
97                              value=0.2,fit=False,limits=[0,0.99])
98              self.N0 = par(name='N0',value=1,texname='N_0$',\
99                              fit=False,limits=[0,None])
100             self.rhoSub = par(name='rhoSub',texname='\rho_{sub}',\
101                             value=1,fit=False,limits=[0,None])
102             self.rhoInt = par(name='rhoInt',texname='\rho_{int}',\
103                             value=1,fit=False,limits=[0,None])
104             self.rhoMol = par(name='rhoMol',texname='\rho_{mol}',\
105                             value=1,fit=False,limits=[0,None])
106             self.dMol = par(name='dMol',texname='d_{mol}',\
107                             value=1,fit=False,limits=[0,None])
108             self.dInt = par(name='dInt',texname='d_{int}',\
109                             value=0.5,fit=False,limits=[0,None])
110             self.L = par(name='L',texname='L',\
111                             value=0.5,fit=False,limits=[0,1])
112             self.I0 = par(name='I0',texname='I_0',\
113                             value=1,fit=False,limits=[0,None])
114             self.N = 10
115             self.tInitial = 0
116             self.tFinal = 100
117             self.tStep = 1
118             self.time = numpy.arange(self.tInitial,\
119                             self.tFinal+self.tStep, self.tStep)
120             self.datafile = datafile
121
122             #~ Write fit parameters to array for index accesability.
123             self.pars = [self.R1, self.deltaR, self.thCr1, self.thCr2,\
124                             self.thCrN, self.N0, self.rhoSub, self.rhoInt,\
125                             self.rhoMol, self.dMol, self.dInt, self.L, self.I0]
126
127             #~ Initialize data file
128             if datafile != 0:
129                     l=0
130                     for line in open(datafile):
131                             l=l+1
132                     self.xData = numpy.zeros(l)
133                     self.yData = numpy.zeros(l)
134                     k = 0
135                     for line in open(datafile):
136                             columns = line.split("\t")
137                             self.xData[k] = columns[0]
138                             self.yData[k] = columns[1]
139                             k = k+1
140                     self.xData_full = scipy.array(self.xData)
141                     self.yData_full = scipy.array(self.yData)
142                     self.yData = self.yData_full
143                     self.xData = self.xData_full
144                     self.time = scipy.array(self.xData)
145
146
147     def setTime(self,tInitial=None,tFinal=None,tStep=None):
148             ''' Modify time limits with optinal arguments.'''
149
150                     #~  Ask for user input
151             if tInitial == None and tFinal == None and tStep == None:
152                     print 'old values:\n\ntInitial =', self.tInitial, \
153                         '\ntFinal =', self.tFinal, \
154                         '\ntStep =', self.tStep
155
156                     while True:
157                             try:
```

```
158                                     tInitial = float(raw_input('tInitial: '))
159                                     tFinal = float(raw_input('tFinal: '))
160                                     tStep = float(raw_input('tStep'\
161                                     '(only effective, there is no exp data present): '))
162
163                                     break
164                             except ValueError:
165                                     print "Please, choose only float values"
166
167
168             if tFinal == 0:
169                     # set Time array according xData
170                     self.time = scipy.array(self.xData)
171
172             elif self.datafile == 0:
173
174                     self.time = numpy.arange( tInitial, tFinal+tStep, tStep )
175                     print 'New time intervall succesfully chosen.'
176
177             elif self.datafile != 0:
178
179                     self.time = scipy.array( self.xData )
180
181                     if self.xData[-1] < tFinal:
182                             print 'Please choose tFinal to be smaller than %f!'\
183                                             % max(self.time)
184
185                     tFinal = min ( self.xData[-1], tFinal )
186                     self.time = filter( lambda t: t < tFinal, self.xData )
187
188                     self.yData = self.yData_full[0:len(self.time)]
189
190
191
192     def calcCov(self):
193             '''Integrates Theta to calculate the layer coverage.'''
194
195             #~ Initial conditions for coverage
196             Theta_initial = numpy.zeros( self.N )
197             Theta_initial[0] = 1
198
199             self.cov = integrate.odeint (
200                                     GOfitBib.rateqs_real,
201                                     Theta_initial,
202                                     self.time,
203                                     args = ( self.N, self.pars[:] ) )
204                             #~ Further optinal parameters for coverage calculation:
205                             #~ (full_output = True, hmin=1e-4, h0=.0005)
206                             #~ print '...Coverage calculated...'\
207                             #~                       accesable via object.cov'
208
209
210     def calcInt(self):
211             ''' Calculates Intensity for
212                     a given set of parameters and coverage. '''
213
214             self.int = GOfitBib.calcIntSimple(self.cov, self.time,\
215                                     self.N, self.pars[:])
216             #~ print '...Intensity calculated... accesable via object.int'
217
218
219     def calcSim(self):
220             ''' Calculates coverage and intensity '''
221             self.calcCov()
222             self.calcInt()
223
224
225     def calcAlpha(self):
226             ''' Calculates downhill transport. '''
```

```
227                 self.alpha = GOfitBib.alphas(self.cov, self.pars)
228
229
230         def plotInt(self, plotPar=False, figsize=(6, 4.5), dpi=80):
231                 ''' Create intensity plot '''
232                 self.calcSim()
233                 plt.figure(1,figsize=figsize, dpi=dpi)
234                 plt.xlabel('Time [a.u.]')
235                 plt.ylabel('Intensity [1/Time]')
236                 plt.ylim((0,1))
237                 plt.plot(self.time,self.int,'black')
238                 print '... Intensity plot created ...'
239
240                 if plotPar:
241                         print '... Create plot with parameters in title ...'
242                         plt.suptitle(r'$R_1 = %.2f$; $\Delta R= %.2f$;'\
243                         r'$\Theta_{cr,1} = %.2f$; $\Theta_{cr,2} = %.2f$;'\
244                         r'$\Theta_{cr,N} = %.2f$; $N_0 = %.2f$;' '\n' \
245                         r'$\rho_{sub}=%.2f$; $\rho_{int}=%.2f$;'\
246                         r'$\rho_{mol}=%.2f$; $d_{mol}=%.2f$; $d_{int}=%.2f$;'\
247                         r'$L=%.2f$, $I_0=%2.0f$'\
248                          %(self.R1.value, self.deltaR.value, self.thCr1.value,\
249                            self.thCr2.value,self.thCrN.value,self.N0.value,\
250                            self.rhoSub.value,self.rhoInt.value,self.rhoMol.value,\
251                            self.dMol.value,self.dInt.value,self.L.value,\
252                            self.I0.value), fontsize=12)
253
254                 #~ The following is a more elegant, flexible way to print the
255                 #~ parameters. Not sufficently tested yet.
256
257                 #~ def formatParam( param ):
258                         #~ return r'$%s = %.2f$;' % ( param.name, param.value )
259                 #~
260                 #~ plt.suptitle (
261                         #~ " ".join(
262                                 #~ map( formatParam,\
263                                 #~                 self.pars[0 : (len(self.pars)-1)/2] ) + ['\n'] +
264                                 #~ map( formatParam,\
265                                 #~                 self.pars[(len(self.pars)-1)/2+1 : -1] )),\
266                                 #~                 fontsize=13 )
267
268                 else:
269                         print '... To creat plot with parameters in title,'\
270                                         'choose option: plotPar=True'
271
272                 plt.show()
273
274
275
276         def parsSave(self, datafile=0):
277                 ''' Save parameters to file. '''
278
279                 datei = file(datafile,'w')
280
281                 for i in range(len(self.pars)):
282                         par_line = '%s = %.4f \t limits = [%s, %s] \t fit: %d\n'\
283                         %(self.pars[i].name,self.pars[i].value,\
284                                 self.pars[i].limits[0],self.pars[i].limits[1],\
285                                 bool(self.pars[i].fit))
286                         datei.write(str(par_line))
287                 datei.close()
288
289
290         def simSave(self, intfile='int', covfile='cov'):
291                 ''' Save Simulation (Intensity and Coverage) to file. '''
292
293                 datei_int = file(intfile, 'w')
294
295                 datei_int.write( "\n".join(
```

```
296                         [ '%.4f' % intensity for intensity in self.int ]
297                         ) )
298
299                 datei_cov = file(covfile,'w')
300
301
302                 datei_cov.write( "\n".join(
303                         [ "␣".join( [ '%.4f' % element for element in row ] )\
304                                 for row in self.cov ]  ) )
305
306                 datei_cov.close()
307
308
309         def parsLoad(self, datafile=0):
310                 ''' Load parameters from file by use of regular expressions. '''
311
312                 datei = file(datafile)
313
314                 i=0
315                 for line in datei.readlines():
316                         mo = re.search(
317 r"\s*([\w\d]+)\s*=\s*(\d*\.?\d*)\s*limits\s*=\s*" +
318 "\[\s*(None|\d*\.?\d*)\s*,\s*(None|\d*\.?\d*)\s*\]\s*" +
319 "fit\:\s*([\d\w]+)\s*", line )
320
321                         #~ self.pars[i].name = mo.group(2)
322                         self.pars[i].value = eval(mo.group(2))
323                         self.pars[i].limits[0] = eval(mo.group(3))
324                         self.pars[i].limits[1] = eval(mo.group(4))
325                         self.pars[i].fit = eval(mo.group(5))
326                         i = i+1
327
328                 datei.close()
329
330
331         def fit(self, maxIter = 1000, maxFunEvals = 100000, solver = 'ralg'):
332                 ''' Calculate intensity fit. '''
333                 self.SplitPar() # Creates or updates self.fpars and self.cpars
334                 ub = self.ub
335                 lb = self.lb
336                 #~ NSP, NLP are also possible (NSP and ralg runs fine)
337                 if solver == 'pswarm':
338                         p = GLP(self.fitfun, self.fpars, maxIter = maxIter,\
339                                         maxFunEvals = maxFunEvals, name = 'GLP␣InfFit',\
340                                         lb=lb, ub=ub)
341                 else:
342                         p = NSP(self.fitfun, self.fpars, maxIter = maxIter,\
343                                         maxFunEvals = maxFunEvals, name = 'GLP␣InfFit',\
344                                         lb=lb, ub=ub)
345                 p.plot = True
346                 #~ solver =
347                 #~ 'ralg', 'pswarm','ralg', 'ipopt', 'scipy_slsqp'
348
349                 r = p.solve(solver, plot=0)
350
351                 print 'Final␣parameters␣are:'
352
353                 self.fpars = r.xf
354                 self.MergePar()
355                 self.pp()
356                 self.chiSquare()
357                 self.pf()
358
359
360         def fitfun(self, fpars):
361                 ''' Merges fit and non fit parameter,
362                         calculates intensity and returns chi_square '''
363                 self.fpars = fpars
364                 print self.fpars
```

```
365                 self.MergePar()
366                 self.calcSim()
367
368                 err = sum((self.yData - self.int)**2)
369
370                 print 'Chi2 * #datapoints = %f' %err
371                 return err
372
373
374         def chiSquare(self):
375                 self.calcSim()
376
377                 chis = (sum((self.yData - self.int)**2))/(len(self.time))
378
379                 print 'chi2 = %f' %chis
380
381
382         def SplitPar(self):
383                 ''' Splits Parameter from pars array
384                         and creates two vectos (fpar, cpar) with values only '''
385                 self.cpars = []
386                 self.fpars = []
387
388                 for i in range(len(self.pars)):
389                         if self.pars[i].fit == False:
390                                 self.cpars.append(self.pars[i].value)
391                         else:
392                                 self.fpars.append(self.pars[i].value)
393
394                 ''' Create vector with constraints '''
395                 self.ub = []
396                 self.lb = []
397
398                 for i in range(len(self.pars)):
399
400                         if self.pars[i].fit != False:
401
402                                 if self.pars[i].limits[0] != None:
403                                         self.lb.append(self.pars[i].limits[0])
404                                 elif self.pars[i].limits[0] == None:
405                                         self.lb.append(-inf)
406                                 if self.pars[i].limits[1] != None:
407                                         self.ub.append(self.pars[i].limits[1])
408                                 elif self.pars[i].limits[1] == None:
409                                         self.ub.append(inf)
410
411
412         def MergePar(self):
413                 ''' Merges fpar and cpar to intial array
414                         with elements of the subclass par. '''
415
416                 cpar_index = 0
417                 fpar_index = 0
418
419                 for i in range(len(self.pars)):
420
421                         if self.pars[i].fit == False:
422                                 self.pars[i].value = self.cpars[cpar_index]
423                                 cpar_index = cpar_index + 1
424                         else:
425                                 self.pars[i].value = self.fpars[fpar_index]
426                                 fpar_index = fpar_index + 1
427
428
429
430         #~ Abbreviations for commands above.
431         def cs(self):
432                 self.calcSim()
433         def pi(self, plotPar=False, figsize=(6, 4.5), dpi=80):
```

```
434                 self.plotInt(plotPar=plotPar, figsize=figsize, dpi=dpi)
435         def pe(self, figsize=(6, 4.5), dpi=80):
436                 self.plotExp(figsize=figsize, dpi=dpi)
437         def ps(self, plotPar=False, figsize=(6, 4.5), dpi=80):
438                 self.plotSim(plotPar=plotPar, figsize=figsize, dpi=dpi)
439         def psr(self, plotPar=False, figsize=(6, 4.5), dpi=80):
440                 self.plotSimRel(plotPar=plotPar, figsize=figsize, dpi=dpi)
441         def pc(self, plotPar=False, figsize=(6, 4.5), dpi=80):
442                 self.plotCov(plotPar=plotPar, figsize=figsize, dpi=dpi)
443         def pa(self, plotPar=False, figsize=(6, 4.5), dpi=80):
444                 self.plotAll(plotPar=plotPar, figsize=figsize, dpi=dpi)
445         def pac(self, plotPar=False, figsize=(6, 4.5), dpi=80):
446                 self.plotAlphaCov(plotPar=plotPar, figsize=figsize, dpi=dpi)
447         def pp(self):
448                 self.parsPrint()
449         def pf(self, plotPar=False, figsize=(6, 4.5), dpi=80):
450                 self.plotFit(plotPar=plotPar, figsize=figsize, dpi=dpi)
451         def st(self,tInitial=None,tFinal=None,tStep=None):
452                 self.setTime(tInitial=tInitial,tFinal=tFinal,tStep=tStep)
453         def load(self, datafile = 0):
454                 self.parsLoad(datafile)
455         def save(self, datafile = 0):
456                 self.parsSave(datafile)
457         def chis(self):
458                 self.chiSquare()
459         def ssave(self, datafile = 0):
460                 self.simSave(datafile)
```

D.2 *GOfitBib.py*

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3   #
4   #       GO-FitBib.py
5   #
6   #       Copyright 2011 Tammo Rukat <tammorukat@gmail.com>
7   #
8   #       This program is distributed in the hope that it will be useful,
9   #       but WITHOUT ANY WARRANTY; without even the implied warranty of
10  #       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
11
12
13  #~ Import basic modules
14  import math
15  import cmath
16  from numpy import *
17  import scipy as sp
18  import pylab
19
20
21
22  def calcIntSimple(cov, time, N, pars):
23          ''' Calculate Intensity from layers matrix '''
24          rho_sub = pars[6].value
25          rho_int = pars[7].value
26          rho_mol = pars[8].value
27          delta = pars[9].value
28          delta_int = pars[10].value
29          L         = pars[11].value
30          I0        = pars[12].value
31
32          thick_vec = sp.zeros(N-1)
33          thick_vec[:] = 1
34          thick_vec[1] = 1
35          thick_vec[3] = delta
```

```
36
37          rho_vec = sp.zeros(N-1)
38          rho_vec[:] = rho_mol
39
40          Intensity = zeros(len(time))
41          ampSub = rho_sub*cmath.exp(complex(0,2*math.pi*L*delta_int))
42          ampInter = rho_int*(1-cmath.exp(complex(0,2*math.pi*L*delta_int)))
43
44          #~ Write to file for testing
45                  #~ datei = open('phase.txt', 'w')
46                  #~ datei.write(str(cmath.phase(ampSub)))
47                  #~ datei.write('\n')
48                  #~ datei.write(str(cmath.phase(ampInter)))
49                  #~ datei.write('\n \n')
50
51          for t in range(0,len(time)):
52
53                  ampLayer = 0
54                  thick = 0
55
56                  for i in range(1,N-1):
57                          thick = thick + thick_vec[i]
58                          ampLayer = ampLayer + rho_vec[i]*cov[t,i]*\
59                                                  cmath.exp(complex(0,2*math.pi*L*(thick)))
60
61                  Intensity[t] = (((abs((1/L)*(ampLayer+ampInter+ampSub))))**2)
62
63          Intensity[:] = I0*(Intensity[:])
64
65          #~ for i in range(0,len(Intensity)):
66                  #~ Intensity[i] = (Intensity[i]/Lfac)*(I0/Intensity[0])
67
68          #~ Write intensity to file for testing:
69                  #~ datei.write(str(ampSub))
70                  #~ datei.write(str(ampInter))
71                  #~ datei.close()
72          return Intensity
73
74
75
76  def rateqs_real(Theta, t_0, N, pars):
77          ''' returns dTh for a given set of Thetas '''
78          R1 = pars[0].value
79          deltaR = pars[1].value
80          th_cr1 = pars[2].value
81          th_cr2 = pars[3].value
82          th_crn = pars[4].value
83          N0 = pars[5].value
84
85          dTh = zeros(N)
86          xi_1 = 1
87          for n in range(1, N):
88                  if n == 1:
89                          th_cr = th_cr1
90                  elif n == 2:
91                          th_cr = th_cr2
92                  else:
93                          th_cr = th_cr2 + (th_crn-th_cr2)*(1.0-exp(-(n-2)/N0));
94
95                  if (n > 1) and Theta[n-1] <= 0:
96                                  xi_2 = 0
97                                  dTh[n] = 0
98                  if (Theta[n-1] < Theta[n]): # Prevent overhangs
99                          dTh[n] = 0
100                 if Theta[n] >= 1:
101                         Theta[n] = 1
102                         dTh[n] = 0
103                         xi_2 = 1
104                 else:
```

```
                        if Theta[n] <= 0:
                                xi_2 = 0
                        else:
                                xi_2_temp = -math.log(1-Theta[n])
                                xi_2_temp_cr = -math.log(1-th_cr)
                                if xi_2_temp > xi_2_temp_cr:
                                        xi_2 = 1 - math.exp(-math.pow((sqrt(xi_2_temp)-\
                                                        sqrt(xi_2_temp_cr)),2))
                                else:
                                        xi_2 = 0
                        if n == 1:
                                dTh[n] = R1*(1-Theta[n]) + (R1+deltaR)*(Theta[n]-xi_2)
                        else:
                                dTh[n] = (R1+deltaR)*(xi_1 - xi_2)
                if dTh[n] < 0:
                        dTh[n] = 0
                xi_1 = xi_2
        return dTh


def alphas(cov, pars):
        ''' returns dTh for a given set of Thetas '''
        R1 = pars[0].value
        deltaR = pars[1].value
        th_cr1 = pars[2].value
        th_cr2 = pars[3].value
        th_crn = pars[4].value
        N0 = pars[5].value

        (t_max,N) = shape(cov)
        xi = zeros((t_max,N))

        xi_1 = 1
        for t in range(1, t_max):
                for n in range(1, N):

                        if n == 1:
                                th_cr = th_cr1
                        elif n == 2:
                                th_cr = th_cr2
                        else:
                                th_cr = th_cr2 + (th_crn-th_cr2)*(1.0-exp(-(n-2)/N0));

                        if (n > 1) and (cov[t,n-1] <= 0):
                                xi_2 = 0
                        if cov[t,n] >= 1:
                                xi_2 = 1
                        else:
                                if cov[t,n] <= 0:
                                        xi_2 = 0
                                else:
                                        xi_2_temp = -math.log(1-cov[t,n])
                                        xi_2_temp_cr = -math.log(1-th_cr)
                                        if xi_2_temp > xi_2_temp_cr:
                                                xi_2 = 1 - math.exp(-math.pow((sqrt(xi_2_temp)-\
                                                        sqrt(xi_2_temp_cr)),2))
                                        else:
                                                xi_2 = 0
                        xi[t,n] = xi_2

        alpha = zeros((t_max, N))

        for t in range(0,t_max):
                for n in range (1,N):
                        alpha[t,n] = (cov[t,n]-xi[t,n])
        return alpha
```

References

[1] T. HOSOKAI, H. MACHIDA, A. GERLACH, S. KERA, F. SCHREIBER, AND N. UENO. **Impact of structural imperfections on the energy-level alignment in organic films**. *Phys. Rev. B*, **83**:195310, 2011. 1

[2] NORBERT KOCH. **Energy levels at interfaces between metals and conjugated organic molecules**. *Journal of Physics: Condensed Matter*, **20**(18):184008, 2008.

[3] R. RUIZ, A. PAPADIMITRATOS, A.C. MAYER, AND G.G. MALLIARAS. **Thickness Dependence of Mobility in Pentacene Thin-Film Transistors**. *Advanced Materials*, **17**:1795–1798, 2005. 1

[4] J.A. VENABLES, G.D.T. SPILLER, AND M. HANBÜCKEN. **Nucleation and growth of thin films**. *Rep. Prog. Phys.*, **47**:399–459, 1984. 1, 6

[5] S. K. SINHA, E. B. SIROTA, S. GAROFF, AND H. B. STANLEY. **X-ray and neutron scattering from rough surfaces**. *Phys. Rev. B*, **38**:2297, 1988. 4

[6] S. KOWARIK, A. GERLACH, AND F. SCHREIBER. **Organic molecular beam deposition: fundamentals, growth dynamics, and in situ studies**. *J. Phys.: Condens. Matter*, **20**:184005, 2008. 4, 5

[7] F. SCHREIBER. **Organic molecular beam deposition: Growth studies beyond the first monolayer**. *phys. stat. sol. (a)*, **201**:1037–1054, 2004. 4, 5

[8] W.J. HUANG, B.Q. LI, AND J.M. ZUO. **Diffusion-limited submonolayer pentacene thin film growth on hydrogen-passivated Si(111) substrates**. *Surface Science*, **595**:157–164, 2005. 5

[9] JENS WECKESSER, JOHANNES V. BARTH, AND KLAUS KERN. **Direct observation of surface diffusion of large organic molecules at metal surfaces: PVBA on Pd(110)**. *Journal of Chemical Physics*, **110**:5351–5354, 1999. 5

[10] J. P. RABE. **Surface anlysis IV. Moving single atoms and breaking chemical bonds**. *Advanced Materials*, **2**(9):428–429, 1990.

[11] J. IKONOMOV, P. BACH, R. MERKEL, AND M. SOKOLOWSKI. **Surface diffusion constants of large organic molecules determined from their residence times under a scanning tunneling microscope tip**. *Phys. Rev. B*, **81**:161412, Apr 2010. 5

[12] CHRISTOPHER WEBER. *Real-time and In-situ Growth Studies of n-Alkanes on Native Silicon Dioxide*. Master's thesis, Humboldt University Berlin, 2011. 6, 7, 23

[13] DK SCHWARTZ. **Mechanisms and kinetics of self-assembled monolayer formation**. *Annual Review of Physical Chemistry*, **52**:107–131, 2001. 6

[14] V.I. TROFIMOV AND H.S. PARK. **Growth Mode Transitions in Molecular-Beam Epitaxy: Theoretical Model and Experimental Verification**. *Journal of the Korean Physical Society*, **40**:344–348, 2002. 6, 9, 10

[15] S. KOWARIK, A. GERLACH, M.W.A. SKODA, S. SELLNER, AND F. SCHREIBER. **Real-time studies of thin film growth: Measurement and analysis of X-ray growth oscillations beyond the anti-Bragg point**. *Eur. Phys. J. Special Topics*, **167**:11–18, 2009. 7

[16] A.R. WOLL, T.V. DESAI, AND J.R. ENGSTROM. **Quantitative modeling of in situ x-ray reflectivity during organic molecule thin film growth**. *arXiv:1102.2676v2 [cond-mat.mtrl-sci]*, 2011. 7, 9, 13

[17] S. KOWARIK, A. GERLACH, S. SELLNER, F. SCHREIBER, L. CAVALCANTI, AND O. KONOVALOV. **Real-Time Observation of Structural and Orientational Transitions during Growth of Organic Thin Films**. *Phys. Rev. Lett.*, **96**:125504, 2006. 8

[18] V.I. TROFIMOV, V.G. MOKEROV, AND A.G. SHUMYANKOV. **Kinetic model for molecular beam epitaxial growth on a singular surface**. *Thin Solid Films*, **306**:105–111, 1997. 9, 10

[19] P.I. COHEN, G.S. PETRICH, P.R. PUKITE, G.J. WHALEY, AND A.S. ARROTT. **Birth-Death Models of Epitaxy**. *Surface Science*, **216**:222–248, 1989. 9, 11, 12

[20] W. BRAUN, B. JENICHEN, V.M. KAGANER, A.G. SHTUKENBERG, L. DÄWERITZ, AND K.H. PLOOG. **Layer-by-layer growth of GaAs(001) studied by in situ synchrotron X-ray diffraction**. *Surface Science*, **525**:126–136, 2003. 9

[21] V.G. MOKEROV V.I. TROFIMOV. **Rate equations model for layer epitaxial growth kinetics**. *Thin Solid Films*, **428**:66–71, 2003. 9

[22] V.G. MOKEROV V.I. TROFIMOV. **Epitaxial growth kinetics in the presence of an Ehrlich-Schwoebel barrier: comparative analysis of different models**. *Materials Science and Engineering B*, **89**:420–425, 2002.

[23] J. KIM V.I. TROFIMOV. **The effect of finite film thickness on the crystallization kinetics of amorphous film and microstructure of crystallized film**. *Thin Solid Films*, **495**:398–403, 2006.

[24] S. BAE V.I. TROFIMOV, J. KIM. **Influence of two different adatom mobilities on the initial heteroepitaxial growth kinetics**. *Surface Science*, **601**:4465–4469, 2007.

[25] V. I. TROFIMOV, J. KIM, AND S. BAE. **Effect of deposition conditions on the structure, phase composition and electrical properties of the chromium silicide films**. *Journal of Physics: Conference Series*, **100**:082005, 2008. 9, 10

[26] R.L. SCHWOEBEL AND E.J. SHIPSEY. **Step Motion on Crystal Surfaces**. *J. Appl. Phys.*, **37**:3682, 1966. 10, 32

[27] W. A. JOHNSON AND R. F. MEHL. **Reaction Kinetics in processes of nucleation and growth**. *Trans. Am. Inst. Min. Metall. Eng.*, **135**:416, 1939. 13

[28] M. FANFONI AND M. TOMELLINI. **The Johnson-Mehl- Avrami-Kohnogorov model: A brief review**. *Il Nuovo Cimento D*, **20**:1171–1182, 1998. 10.1007/BF03185527. 13, 28

[29] M. TOMELLINI AND M. FANFONI. **Mean field approach for describing thin film morphology**. *Journal of Physics: Condensed Matter*, **18**(17):4219, 2006. 13

[30] A. C. HINDMARSH. **Odepack, a systematized collection of ODE solvers**. *Scientific Computing*, **1**:55–64, 1983. 15

[31] L. N. VICENTE A. I. F. VAZ. **PSwarm: A Hybrid Solver for Linearly Constrained Global Derivative-Free Optimization**, December 2009. 40

[32] S. DIETRICH AND H. WAGNER. **Critical Surface Scattering of X Rays and Neutrons at Grazing Angles**. *Phys. Rev. Lett.*, **51**(16):1469–1471, Oct 1983.

[33] YU WU, TULLIO TOCCOLI, NORBERT KOCH, ERICA IACOB, ALESSIA PALLAORO, PETRA RUDOLF, AND SALVATORE IANNOTTA. **Controlling the Early Stages of Pentacene Growth by Supersonic Molecular Beam Deposition**. *Phys. Rev. Lett.*, **98**:076601, Feb 2007.

[34] N. KOCH. **Organic Electronic Devices and Their Functional Interfaces**. *ChemPhysChem*, **8**:1438–1455, 2007.

Acknowledgements

First of all I want to thank Stefan Kowarik for introducing me into the field of organic thin films and x-ray techniques, for giving me the possibility to do this research and for supporting me with intense discussions and most helpful collaboration and advice.
Furthermore, I would like to thank Christopher Weber and all my colleagues from the group for coherent optics with x-rays at Humboldt University for the rewarding dicussions and the pleasant collaboration.

I also wish to express my gratitude to Tushar Desai, Arthur Woll and James Engstrom at Cornell University for having me as a guest in summer 2010 and making this stay not only instructive, but a pleasure.

Last but not least I would like to thank Karina Bzheumikhova, Dorothea vom Bruch, Michael Borinsky and Martin Liesenberg for reviewing and discussing this thesis and for their occasional delightful distractions.

Declaration

I confirm that the work presented in this research report has been performed and interpreted solely by myself except where identified to the contrary.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Berlin, September 2011

Tammo Rukat