Manuel Peter Kainz, BSc

Indexing of Grazing-Incidence X-Ray Diffraction Patterns

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme:

Technical Physics

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Roland Resel

Institute of Solid State Physics

Graz, October 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date, Signature

Acknowledgement

I would first like to thank my thesis supervisor Roland Resel from the Institute of Solid State Physics at the University of Technology in Graz. The door to the office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this thesis to be my own work, but steered me in the right the direction whenever he thought I needed it.

Special thanks to Josef Simbrunner from Medical University Graz. This thesis relies crucially on his previous work and many applications would not have been possible or at least much more time-consuming without his considerable expertise regarding the indexing of grazing-incidence X-ray diffraction patterns.

I also want to thank all my former and current colleagues at the institute and the work group. I am grateful for the team I was a member of and special thanks go to the people from k-Raum as well as k'-Raum. I had a great time being there.

Finally, I want to express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

Abstract

Grazing-incidence X-ray diffraction (GIXD) is a widely used technique for the crystallographic characterization of thin films. The identification of a specific phase or the discovery of an unknown polymorph requires indexing of the associated diffraction pattern. Indexing describes the assignment of Laue indices to the individual diffraction peaks with simultaneous determination of the lattice constants. However, despite the importance of this procedure, only few approaches have been developed so far. Recently, a mathematical framework for indexing of this specific diffraction pattern has been developed. This work introduces a successfully implemented indexing routine. The algorithm is based on the assumption of a triclinic unit cell with six lattice constants. Two approaches are chosen: i) using only diffraction peaks of the GIXD pattern and ii) combining the GIXD pattern with a specular diffraction peak. In the first approach the six unknown lattice parameters have to be determined by a single fitting procedure, while in the second approach two successive fitting procedures are used with three unknown parameters each. The latter case is elaborated in detail throughout this thesis. The computational toolkit is compiled in the form of a MATLAB application and presented within a user-friendly graphical user interface. The program is tested by application to four independent examples of organic thin films.

Kurzfassung

Röntgenbeugung unter streifendem Einfall (engl. GIXD) ist eine weit verbreitete Technik zur kristallographischen Charakterisierung dünner Filme. Die Identifizierung bestimmter Phasen sowie die Untersuchung von unbekannten polymorhpen Materialien erfordert die Indizierung des zugehörigen Beugungsmusters. Indizierung beschreibt die Zuordnung von Laue-Indizes zu den einzelnen Beugungspeaks bei gleichzeitiger Bestimmung der Gitterkonstanten. Trotz der Bedeutung dieses Verfahrens wurden bisher nur wenige Ansätze entwickelt. Kürzlich wurde ein mathematischer Rahmen für die Indizierung dieses spezifischen Beugungsmusters entwickelt. Die hier gezeigte Arbeit stellt eine erfolgreich implementierte Indizierungs-Routine vor. Der Algorithmus basiert auf der Annahme einer triklinen Einheitszelle mit sechs Gitterkonstanten. Es werden zwei Ansätze gewählt, wobei bei Ersterem nur die Beugungspeaks der GIXD-Messung verwendet werden. Bei der zweiten Methode wird zusätzlich ein spekularer Beugungspeak aufgenommen. Beim ersten Ansatz müssen die sechs unbekannten Gitterparameter durch ein einziges Verfahren bestimmt werden, während beim zweiten Ansatz zwei aufeinanderfolgende Routinen mit jeweils drei unbekannten Parametern verwendet werden. Der zweite Ansatz stellt das zentrale Thema dieser Arbeit dar. Das Toolkit wird in Form einer MATLAB-Anwendung kompiliert und in einer benutzerfreundlichen grafischen Benutzeroberfläche dargestellt. Das Programm wird durch Anwendung auf vier unabhängige Beispiele für organische Dünnfilme getestet und demonstriert.

Contents

Chapter 1

Introduction

1.1 Indexing: Definition and Use

The inner structure of a crystalline material is of central interest in solid state research. The structural order is used to understand and predict the properties of these materials. However, such crystal structures are not necessarily unique and one and the same material can crystallize in two or more forms with distinct structures. These *polymorphs* exhibit different unit cells and distinct arrangements of atoms within them. Knowledge about these polymorphs serves as a powerful tool to fabricate tailor-made applications in a variety of possible fields, such as pharmaceutical science [1]. Of great interest are also the changing characteristics upon tapering the size of organic semiconductor structures from bulk to thin films or even monolayers. Applied in microelectronics industry, promising results have been reported for example in organic thin-film devices [2, 3].

For characterization of the properties and especially for the search for potentially different polymorphs, the materials can be synthesized in a crystalline form. Besides composition, the inherent structure is responsible for the chemical and physical properties among these different crystal forms [4]. Crystallization of molecular materials on a solid surface and production of thin films proved particularly suitable for these structural investigations [5]. Because of the periodic arrangement of atoms or molecules within an ideal crystal, the region of interest can be reduced to the smallest repetitive building block, the unit cell. The unit cell describes the smallest portion of a crystal lattice, which is replicated in three dimensions to form the entire crystal. It is therefore sufficient to describe the properties of and within the unit cell, to describe the buildup of a crystalline solid. As an ideal crystal lattice can be seen as a grid with periodically arranged lattice points, diffraction experiments can help to learn about the inherent structure. X-rays with associated

wavelengths of 10^{-11} m to 10^{-10} m are particularly suitable to resolve the inter-atomic or net-plane distances in the range of 1 Å [6, 7, 8].

Depending on the appearance of the sample, different measurement methods are established. Powder and single-crystal X-ray diffraction are the two main techniques for solution of the crystal structures [9]. When it comes to surface-sensitive methods to investigate organic thin films, grazing-incidence X-ray diffraction (GIXD) experiments are used for structure determination [10, 11, 12].

What all of these methods have in common is that the information is expressed as a diffraction pattern. These diffraction patterns are the accumulation of the reflections due to constructively interfering scattered waves (Figure 1.1). Interfering waves lead to a diffraction peak if the scattering vector $\mathbf{q}$ matches a vector of a reciprocal lattice $\mathbf{g}_{hkl}$. This is expressed in the Laue condition of diffraction as

$$\mathbf{q} = \mathbf{g}_{hkl}, \tag{1.1}$$

where the reciprocal lattice vector is the linear combination of the reciprocal basis vectors of the form $\mathbf{g}_{hkl} = h\mathbf{a}^* + k\mathbf{b}^* + l\mathbf{c}^*$. The integer values h, k and l of this linear combination are the Laue indices. To extract structural information about the sample, the reflections need to be indexed. Indexing is the assignment of Laue indices to the diffraction peaks (or reflections) by simultaneous calculation of the crystal lattice parameters.

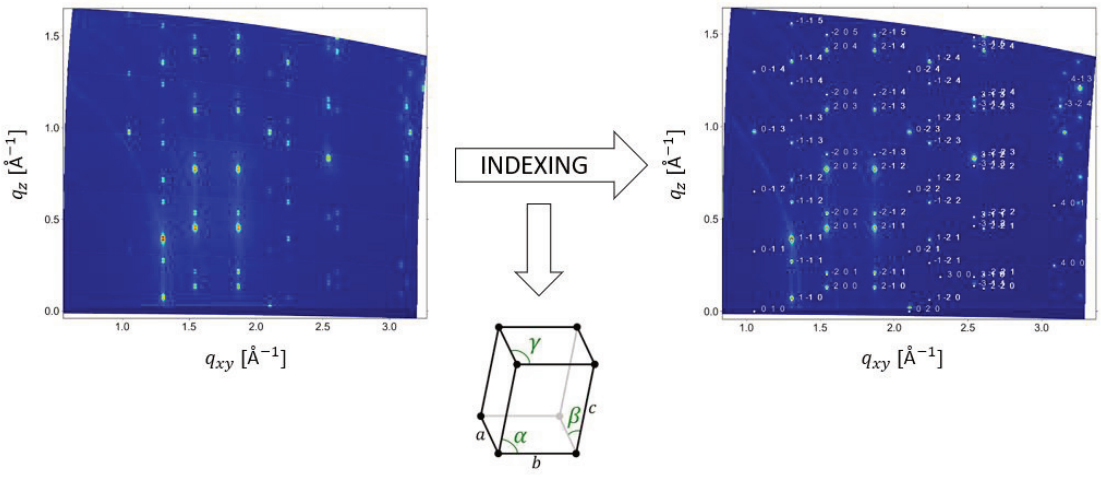


Figure 1.1: Principle of indexing: Assignment of Laue indices to diffraction peaks in reciprocal space map and unit cell determination [13, 14].

The results after indexing are the assigned triplets of Laue indices h, k and l for every reflection and an estimation of the unit cell dimensions (Figure 1.1). Not only the size of the unit cell is of interest, but also the intensity of every diffraction peak together with its

Laue triplets (h,k,l). We see a diffraction peak with the intensity I_{hkl}, where the scattering vector $\mathbf{q}$ images a reciprocal lattice vector $\mathbf{g}_{hkl}$:

$$I_{hkl} = |F_{hkl}|^2 \, \delta(\mathbf{g}_{hkl} - \mathbf{q}) \tag{1.2}$$

The amplitude of a reflection intensity is determined by the absolute square of the structure factor $|F_{hkl}|^2$. Analytically, the structure factor F_{hkl} itself is a complex quantity with amplitude and phase. Using the structure factors, the spatial electron distribution $\rho(\mathbf{r})$ within the unit cell volume V can be computed through Fourier transformation. This assertion is expressed in the electron density equation:

$$\rho(\mathbf{r}) = \frac{1}{V} \sum_{hkl} |F_{hkl}|^2 \, e^{i(\mathbf{g}_{hkl}\mathbf{r} + \Phi_{hkl})} \tag{1.3}$$

With this information obtained from indexing it is therefore possible to estimate the locations of atoms within the unit cell. This should show the importance and the central role of the indexing process. Clearly, the situation is more complex in reality as the absolute square of the structure factor $|F_{hkl}|^2$ reveals no information about the phase Φ_{hkl}. This *phase problem* remains to be solved and simulation of the packing of the molecules is an ambitious computational task. Techniques to solve the phase problem are so called *direct methods*, *Patterson analysis* for heavy atoms and approaches using *test structures* as well as applications of theoretical concepts such as density function theory and molecular dynamics simulations [15, 16].

All the information combined, the unit cell, the lattice type and the packing of the atoms in the space of the unit cell is then referred to as the *full crystal structure solution*. Figure 1.2 should summarize the above explained steps and point out once more the central role on the indexing process on the way to a crystal structure solution;
Starting with a sample which is suspected to be crystalline, the first step is to choose a proper measurement technique. Here, a schematic drawing of a GIXD setup is shown. In a further step, the detector data have to be converted from pixel space to reciprocal space to identify the peak positions. Indexing and determination of the unit cell follows and marks a central point of the process. If the packing of the atoms (or stacking of molecules) within the unit cell can be identified, a crystal structure solution is obtained.

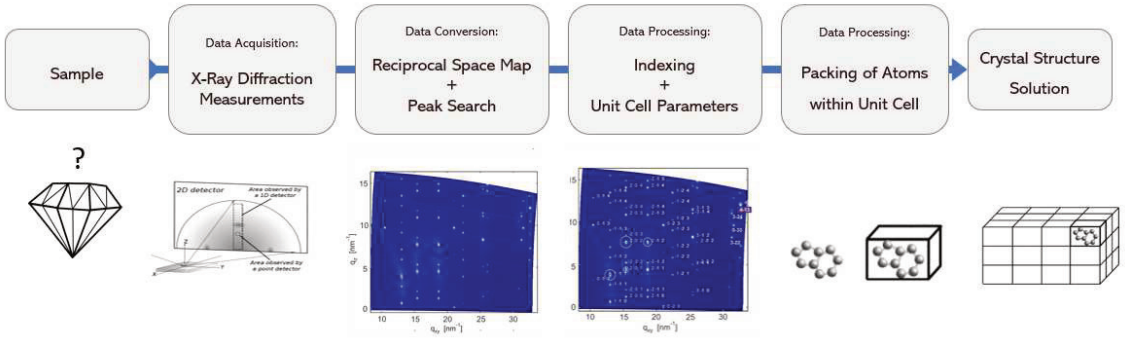


Figure 1.2: Workflow from left to right for determination of a crystal structure solution [13, 14, 17].

Depending on the measurement technique used, there are different approached for indexing established. On the following pages, a brief summary of the methods and the possible indexing processes are given for single-crystal X-ray diffraction, powder diffraction and GIXD.

Chapter 2

Review

2.1 Indexing of Single-Crystal Diffraction Data

Single-crystal analysis is the most commonly technique used for determining the electron density within the unit cell of crystals [9]. Monochromatic X-rays are collimated and directed onto the sample. The samples are usually mounted on thin glass fibers which are attached to glass pins and mounted onto goniometer heads. A schematic setup is shown in Figure 2.1 a) and b).
Constructive interference occurs when the geometry of the incident and the scattered X-rays satisfies the Bragg equation. The scattered X-ray signal is recorded by a detector which converts the signal into a count rate and provides the information about the individual intensity. While the sample is gradually rotated, different orientations of the crystal are probed and able to contribute to the diffraction pattern, if the conditions for constructive interference are met. Measuring single-crystal structures usually provides several hundred or thousand unique diffraction peaks (see Figure 2.1 c). This makes this technique very accurate when determining the structure upon indexing [18]. After the data is collected, corrections for instrumentation, polarization effects and X-ray absorption must be applied to the entire data set.
Different approaches for indexing and structure solving of single-crystal diffraction data are available [9]. One possibility to determine a crystal structure solution uses comparison and adjustment of structure factors. As mentioned earlier, the structure factor is a complex quantity and connected to the electron density through a Fourier relationship containing the phase information. During diffraction experiments, only intensities can be measured by the detector and information about the phase of the scattered wave is lost. For determination of the structure factors however, intensity *and* phase are needed.

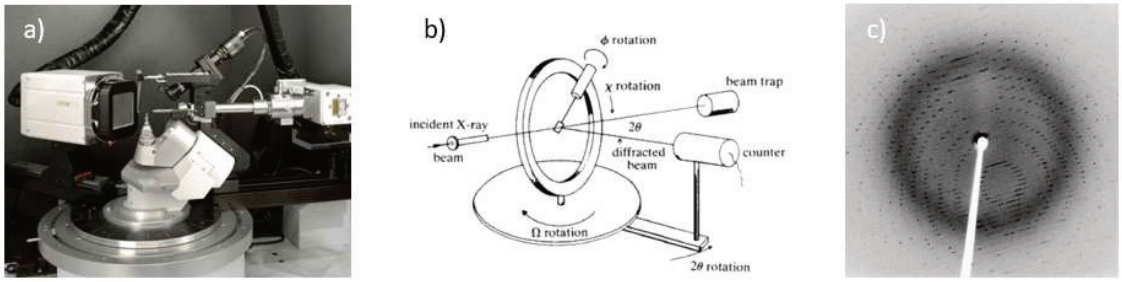


Figure 2.1: a) Example of a setup for single-crystal XRD with detector, sample holder and X-ray source (f.l.t.r). b) Schematic drawing of a setup to visualize the angles as degrees of freedom. The Φ-rotation of the sample is indicated on top side [19]. c) Example of a diffraction pattern of a single-crystal XRD measurement.

The drastically enhanced computational power allows solutions of the "phase problem" by the use of direct methods. This means, that the phase estimation is done statistically and only the intensities and a random starting set of phases are used. These methods are very successful for small molecule structures with up to 100 (non-H) atoms [9, 20, 21]. Patterson analysis is based on the squared structure factors and does therefore not need any phase information.

Indexing is one part of the crystal structure solution. In a possible indexing procedure, unit cell test volumes are built by the use of Laue index triplets. These test volumes consist of three linearly independent reciprocal lattice vectors. During this subsequent indexing, the triplets are assigned to the diffraction peaks in the form of (*hkl*) values, which have different intensities or scattering amplitudes. What follows is the conversion into structure factors due to their Fourier transform relationship. A preliminary structure solution is carried out. This solution serves to determine a preliminary electron density distribution in the unit cell. The electron density maxima are assigned which exhibit atoms with a suitable number of electrons. Bonds that lead to a molecular model are formed, when atoms are closer together than the sum of their Van der Waals radii [20].

The process of single-crystal refinement is a subsequent step. Slight changes of atomic positions within the cell are used to refine and optimize the crystal structure from the beginning [22, 23]. In that way, the models found are improved step by step with this structural refinement. The end of the refinement is reached when the difference of the observed intensities and the calculated structure factors is no longer reduced by changing the parameters (and their associated test volumes) and when the solution is chemically appropriate. In that way, a continuous adjustment of cell parameters and Laue indices lead to the best fitting solution through comparison of measured and calculated struc-

tures.

In principle, the software set *SHELX* with its derived packages works as described above. It brings packages for holistic crystal structure determination where the indexing routines are already included. It is the most widely used tool for solving single-crystal structures [21, 22, 23]. Although many information is given about the history and development of the powerful software package, a conclusive and detailed explanation about the indexing process itself is not available.

Another available software for indexing single-crystal diffraction data is an application called *IND_X* [24]. The working principle of this software is the generation of test volumes out of the unit cell in reciprocal lattice representation. An initial guess is either generated randomly by the software or provided by the user. However, this choice usually requires experience in the processing of single-crystal XRD data. The first derivation of a solution is based on a period-detection algorithm, which is a known algorithmic problem in computer science. The initial cell solution is used to derive a subset of vectors and compared with the observed vectors (diffraction data). The vectors are ranked, and the best fit constitute as input for calculation of the next test volume. In that way, the indexing problem is reduced to check all lattice bases of small-index super-lattices and their fit on the diffraction data. The output of this indexing algorithm are sets of proposed solutions for. They are provided in the form of direct lattice vectors and fulfill criteria for a Buerger-reduced cell [25]. We will see later, why this plays an important role when limiting possible solutions. The final choice for a unit cell solution is handed to the user [24].

2.2 Indexing of Powder Diffraction Data

X-ray powder diffraction (XRPD) is another commonly used technique using scattering of X-rays for structural investigation. As the name implies, the sample should consist of a powder containing randomly oriented crystalline particles. Usually, the ground material is packed in a cavity-type sample holder to avoid preferred orientations. Crystallites (small crystals) resolved in coatings, as well as sintered powders are another possible sample form to investigate.

XRPD is used for various investigations. In a straightforward manner, it is possible to distinguish quickly if a solid material is crystalline or amorphous. This basic capability can be applied to all kind of solid materials. Another general application is to test if the material or sample consists of one or more crystalline phases. The powder diffraction

pattern is characteristic for a crystalline structure and it is a fingerprint of the phase, when compared with experimental data from diffraction data bases [26]. Besides phase composition and quantitative phase analysis, the diffraction method provides data to determine the unit cell parameters of the present crystalline phase.
A widely used configuration in powder diffractometers (as well as in standard XRD experiments) is the Bragg-Brentano geometry. A schematic drawing of this setup and an example instrument are shown in Figure 2.2 b) and Figure 2.2 a), respectively. ω is here the incident angle and defined between the X-ray source and the sample. The diffraction angle 2θ is defined between the incident beam and the scattered beam. The incident angle ω is always one half of the detector angle 2θ.

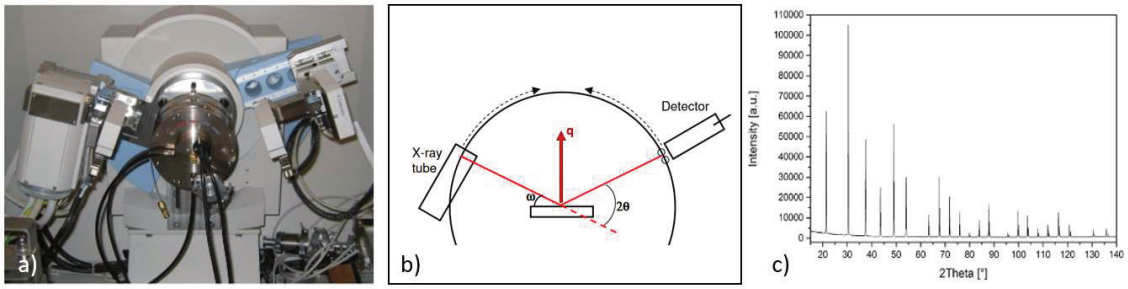


Figure 2.2: a) Powder X-ray diffraction instrumentation. b) Schematic image of Bragg-Brentano geometry with incident angle ω and diffraction angle 2θ. c) Diffraction pattern of a LaB_6 reference powder specimen with scattering intensity versus diffraction angle. [27, 28]

The scattering vector q is the vector that bisects the angle between the incident and scattered beam. In this specific geometry, it points always perpendicular to the sample surface (Figure 2.2 b). Ideally, the powder sample consists of thousands of randomly orientated crystallites. In that way, the Bragg condition can be fulfilled for various diffraction angles, depending on the orientation of the crystallites. For every set of planes within the powder, there will be a certain percentage of crystallites that are properly oriented to fulfill the condition for diffraction. The basic assumption for this behaviour is that there is a statistically relevant number of crystallites present in the powder. Only a fraction of the crystallites actually contribute to the measured diffraction pattern, making the technique vulnerable and somewhat inefficient. To enhance the efficiency, irradiation of larger volumes of the material and spinning can help ensure that a statistically relevant number of grains contribute to the diffraction pattern.

The data from X-ray powder diffraction experiments are detected intensities over the diffraction angle of 2θ (Figure 2.2 c). This information must be sufficient to determine unit cell parameters from the experimental data. Various approaches for indexing powder

patterns have been reported. A comprehensive list of software packages and tools can be found here [29]. Well-known powder indexing tools are the applications *DICVOL04* and *X-Cell*. As both share the same principle idea, this algorithm should serve as an example on indexing a powder pattern. The algorithms both use the length of the scattering vector **q** and its relation with the reciprocal lattice. The length can be derived using Bragg's law and the wavelength of the used X-rays:

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

Every scattering vector which leads to a diffraction peak in the powder pattern can be equivalently expressed through a lattice vector of the reciprocal space. Assuming a*, b* and c* are the cell edges of the reciprocal cell with their including angles α^*, β^* and γ^*, the reciprocal lattice vector for the i-th reflection reads as

$$q_i(hkl) = h_i^2 a^{*^2} + k_i^2 b^{*^2} + l_i^2 c^{*^2} + 2h_i k_i a^* b^* cos\gamma^* + 2h_i l_i a^* c^* cos\beta^* + 2k_i l_i b^* c^* cos\alpha^*. \tag{2.2}$$

This equation is not solvable by ordinary algebra since the number of equations is always smaller than the number of unknown parameters. This is even true for the most symmetric case, the cubic lattice. In this case, one equation still contains four unknown parameters with one cell edge, and a set of Laue indices (h_i, k_i, l_i).

A solution to equation 2.2 can be obtained by a successive dichotomy procedure [30, 31]. A set of possible indices is chosen in the beginning of every peak cycle. As the set of possible Laue index permutations is sufficiently small for the powder pattern indexing problem, low-index permutations can be scanned within a reasonable computing time [30]. In a next step a set of starting parameter is chosen. This is either done by the user (cell restrictions as input) or pre-defined by the algorithm. The cell edges and angles of this parameter set are equally divided into test volumes, so called volume domains. For a given set of (hkl), an upper bound $q_{i_+}(hkl)$ and a lower bound $q_{i_-}(hkl)$ serve as selection criteria. If the calculated pattern q_i with the current volume is in range of these boundaries, the volume is divided (bisected) in a volume sub-domain and evaluated again. If the pattern does not fit in the specified range, the domain is eliminated and a new domain is generated. At the implementation of *DICVOL04*, the bisection procedure is limited to a repetition number of six. After six repetitions, the domains define the cell parameters. In that way, the smallest cell volume is obtained and the reduced cell based on the shortest three non-coplanar lattice parameters is determined. In this tool, the successive dichotomy or bisection algorithm is applied successively to each crystal system. For reasons of efficiency, the algorithm starts with highest symmetry (cubic

system) and increases then the number of unknowns. A comprehensive elaboration of the procedure as well as the criteria for the boundaries are summarized in [31].
Although the algorithmic principle of using the dichotomy method is the same in *DICVOL04* and *X-Cell*, the programmatic implementation is different [32]. The number of possible sub-domain volumes as well as the maximum recursion number is regulated differently. This leads to different execution times and performances among the compared tools.

The point here is not to find a better or worse approach. Purpose of this and the previous chapter is, to show which possibilities are there for indexing X-ray diffraction patterns and in that way introduce the actual topic of this work, the indexing of GIXD patterns.

2.3 Indexing of GIXD Data

Grazing-incidence X-ray diffraction has become an intensively elaborated technique for structural characterization of thin films. The formation of potentially new polymorphs due to transition from bulk to thin films is a well know phenomenon and reported in various works [33, 5]. The surface sensitivity in a GIXD setup is highly increased, what makes it particularly well-suited for investigations of such crystalline samples. A more detailed setup description is discussed in chapter *4*.
Thin films grown on isotropic substrates can exhibit fibre-texturing and thereby showing a defined crystalline plane oriented parallel to the substrates surface. For determination of a unit cell and subsequently a crystal structure solution, identification of this parallel plane (hereafter referred to as contact plane) is of central interest [34].
Just as in single-crystal XRD and powder diffraction, the crystallographic unit cell is obtained by indexing of the diffraction data. The consideration of the contact plane needs an innovative mathematical treatment for the index procedure of GIXD data. The influence results in an increased set of unknown parameters. A comprehensive formalism, which addresses exactly this problem, is proposed in recent works [10, 11, 12]. A few approaches for processing experimental GIXD data have yet been developed [35, 36, 37, 38, 39, 40]. Not all of them address the task of indexing with regard of unit cell determination and even less consider the contact plane as separate parameter necessary to determinate. All of the here reported approaches try to solve the same problem; An experimental GIXD pattern is compared with calculated reciprocal lattice vectors of the form

$$\mathbf{g}_{hkl} = h\mathbf{a}_r^* + k\mathbf{b}_r^* + l\mathbf{c}_r^*. \tag{2.3}$$

This can be done for both, the in-plane component and the out-of-plane component of the scattering vector and the lattice vector, respectively. Having regard to whether or not the contact plane is considered, the vectors $\mathbf{a}_r^*$, $\mathbf{b}_r^*$ and $\mathbf{c}_r^*$ are the known reciprocal basis vectors or the rotated ones where the rotation is recognised using $\mathbf{a}_r^* = \mathbf{R}\mathbf{a}^*$. The rotation matrix $\mathbf{R}$ acts on the lattice vectors as rotation around the zone axis of the (001) contact plane.

One of the earliest reported algorithms to address the problem of indexing GIXD data in that way was published in 2007 [40]. The reciprocal lattice vectors can be calculated in a straightforward manner, as knowledge of the unit cell parameters is a prerequisite here. The rotation around the zone axis is applied by using the *known* orientation of the planes parallel to the substrate surface. The indexing is based on calculation and comparison of the vectors and assignment of the best fitting diffraction indices. The unit cell parameters are not explicitly calculated, making it a tool for assignment of Laue indices only.
In 2008, a computer program for simulating GIXD experiments explicitly from thin films has been reported [35]. Particular attention has been given to the modelling of the peak intensities for fibre-textured films. Indexing is a subitem of this program called *SimDiffraction*. In the algorithm proposed, only a fibre axis perpendicular to the substrate is considered, what makes it a (001)-approach only [1]. Just as the former approach, knowledge of the direct lattice cell parameters is necessary for indexing of the experimental peak positions.

[1] In the context of this work, the term 'fibre axis' is used to describe the direction perpendicular to the contact plane. It is the same direction as the surface normal of the contact plane.

A significant improvement of this issue is the computational tool *Diffraction Pattern Calculator (DPC)*, published in 2014 [36]. It was the first toolkit that contained the determination of the unit cell parameters by simultaneous assignment of the Laue indices. A deviation from the special case with a (001) contact plane is also recognised and realised as additional input variable. The working principle is again based on checking the Laue condition by evaluating equation 2.3. The peak positions for numerical comparison are derived automatically from a GIXD pattern through a image processing routine. From this, the program extracts the q-data, separated in the in-plane components q_{xy} and the out-of-plane components q_z. Prior to the unit cell analysis, several operation parameter have to be defined by the user.
The analysis requires an initial guess for the contact plane, the range of the Laue indices h, k and l and the values of the direct lattice parameters. An initial guess of the unit cell is therefore needed. A further needed input is the choice of the space group. The routine then computes all the possible permutations of Laue indices in the specified range and removes forbidden reflections due to the reflection conditions. This is an efficient method to reduce computational effort. Additional boundary conditions in the automatic mode are here applied by the use of a specular scan ($q_{xy} = 0$). Although the toolkit integrates many features, the high amount of necessary input and previous knowledge for indexing could be a limitation of the *DPC*.
The software package *GIXSGUI* shares similarities with most of the former described programs. However, the focus is more on geometric corrections, two-dimensional intensity reshaping and visualization in general [37]. Another powerful GIXD processing tool is the MATLAB-based application *GIDVis* [39]. Some functionalities of *GIDVis* will be elaborated in detail in later chapters.

The latest published work addressing the indexing problem of GIXD patterns is the software package *GIWAXS-SIIRkit* [38][2]. The tool contains an indexing routine specifically designed for processing of small-molecule thin films (which corresponds to roughly 1-30 Å). The routine starts by processing diffraction patterns and returning data sets of q_{xy} and q_z. If the peak positions are available, it is convenient to simply upload the numerical values instead of using the image processing tool.This algorithm is of special interest, as the basic principle is also applied in the here presented thesis, namely dividing the indexing process in two parts. In the first part, the q_{xy}-data are preliminarily indexed by tuples of (hk). This is achieved by evaluation of the in-plane component of the scattering

[2]Even though the denotation is different, the physical description of grazing-incidence wide-angle X-ray scattering (GIWAXS) and GIXD is equivalent [41]

vector. It is convenient to use the form of a linear system of equation for three linearly independent peaks:

$$\begin{pmatrix} q_{xy1}^2 \\ q_{xy2}^2 \\ q_{xy3}^2 \end{pmatrix} = \begin{pmatrix} h_1^2 & k_1^2 & h_1k_1 \\ h_2^2 & k_2^2 & h_2k_2 \\ h_3^2 & k_3^2 & h_3k_3 \end{pmatrix} \begin{pmatrix} a_{||}^{*^2} \\ b_{||}^{*^2} \\ 2a_{||}^{*^2}b_{||}^{*^2}cos(\gamma_{||}^*) \end{pmatrix}. \tag{2.4}$$

The so obtained solution is used to assign indices to the remaining peaks. The program cycles through the index permutations within a range of -8 and 8, forms the difference for every case and thereby searches for a minimum in Δq_{xy}.
In the second indexing step, the equation which is valid for the out-of-plane components is solved:

$$q_z = ha_{\perp}^* + kb_{\perp}^* + lc_{\perp}^*. \tag{2.5}$$

The parameters $a_{\perp}^*$ and $b_{\perp}^*$ are the components of a^* and b^* perpendicular to the substrate. The out-of-plane reciprocal lattice vector is denoted by c^*. To solve the second problem, an initial guess of c^* as well as two linearly independent peaks in q_z *must* be provided by the user. The unit cell solution with the lowest total error expressed in Cartesian distances is the result of this algorithm. Clearly, this routine is limited to problems where the (001) plane is the plane parallel to the substrate surface. The use of a specular scan can help, but the initial guess required for c^* could limit the usability of this indexing routine.

This chapter should demonstrate the approaches which have yet been proposed and the according principle on which they work. It is noticeable that assumptions and estimates have to be made in various cases in order to determine solutions for unit cell parameters. In many numerical calculations and simulations of that kind, this is simply necessary to keep the computational effort in reasonable limits. Interestingly, none of the here reported programs or algorithms refer to crystallographic conventions such as the criteria originally imposed by Niggli or the test for a reduced cell defined by Buerger [25, 42].

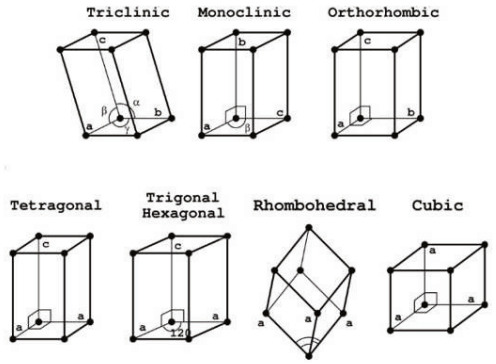
The program presented in this thesis is intended to be a useful addition and a further development to what has already been achieved in this field. In order to better understand the connections and, above all, the later explained formalism, some basic fundamentals will be repeated in the next chapter.

Chapter 3

Fundamentals

3.1 Crystal Lattice and the Unit Cell

A crystal is formed through repetitive translation of the unit cell content in three dimensions. This mathematical description is used to represent the structure as well as the translational symmetry of the crystal[1]. A lattice is a translationally periodic collection of discrete points. The atoms associated with each of these lattice points are introduced by the basis. For a crystal lattice, the allocation to one of the seven lattice types is required.



System	Unit-cell dimensions	Unit-cell angles (°)
Triclinic	$a \neq b \neq c$	$\alpha \neq \beta \neq \gamma \neq 90$
Monoclinic	$a \neq b \neq c$	$\alpha = \gamma = 90, \beta \neq 90$
Orthorhombic	$a \neq b \neq c$	$\alpha = \beta = \gamma = 90$
Tetragonal	$a = b \neq c$	$\alpha = \beta = \gamma = 90$
Trigonal	$a = b \neq c$	$\alpha = \beta = 90, \gamma = 120$
	$a = b = c$	$\alpha = \beta = \gamma \neq 90$
Hexagonal	$a = b \neq c$	$\alpha = \beta = 90, \gamma = 120$
Cubic	$a = b = c$	$\alpha = \beta = \gamma = 90$

Figure 3.1: The shape of the unit cells of the seven crystal systems (left) and their characteristic symmetry and restrictions on the geometry (right) [43].

[1]It should be noted that this is an ideal, mathematical and therefore simplified image of a real crystal. In reality, lattice mismatch, dislocations, crystal defects and impurities are crucially important for the description of crystals and their properties.

These seven types are called the seven crystal systems and can be classified in terms of their symmetry and divided in cubic, hexagonal, rhombohedral, tetrahedral, orthorhombic, monoclinic and trigonal systems. A summary of their characteristic shape and restrictions on the cell geometry is given in Figure 3.1. Once a representative lattice appropriate to the symmetry of the structure is chosen, any lattice node (or net point) can be described by a vector consisting of a linear combination of the lattice vectors. Such a real space vector can be written as

$$\mathbf{r} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c} \tag{3.1}$$

where **a**, **b** and **c** represent the lattice vectors spanning the parallelepiped of the unit cell (Figure 3.2) and u, v, w are any integer values. An alternative definition of the structure using the basis and one of the above introduced lattice types can be formulated as *crystal structure = basis + lattice*.

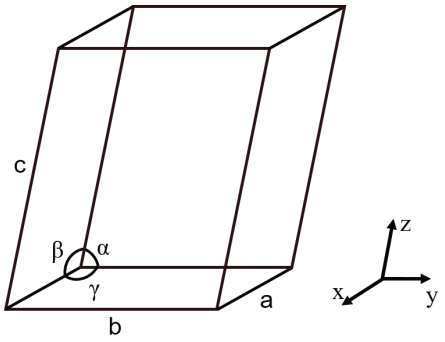


Figure 3.2: Schematic representation of a parallelepiped with edge lengths a, b and c. The angle γ is the enclosed angle between the edges a and b. β is defined as the angle between a and c and the third angle, α, is the enclosed one between b and c.

The direct lattice is convenient for describing the arrangement of atoms and molecules within the crystal. To explain the interaction with electromagnetic waves however, it is helpful to use another mathematical description for the crystalline structure. This arises from the fact that "*...if a structure is arranged on a given lattice, then its diffraction pattern is necessarily arranged on the lattice that is reciprocal to the first*" [6]. The representation in reciprocal space is a method which helps to explain the physical effects such as scattering and diffraction. This representation method describes the structure in terms of the *reciprocal lattice*. Direct and reciprocal lattice are connected by a Fourier relationship. The relation can be expressed in terms of the known expression of the electron density. With **R**, a vector from the origin indicating any point of the lattice, this

relationship reads as

$$\rho(\mathbf{R}) = \frac{1}{V}\sum_{hkl} F_{hkl}\, e^{i\mathbf{g}_{hkl}\mathbf{R}} \tag{3.2}$$

As the electron density must follow the condition of periodicity within the crystal lattice, a translation by any direct lattice vector $\mathbf{r}$ must return the same functional value

$$\rho(\mathbf{R}+\mathbf{r}) = \rho(\mathbf{R}). \tag{3.3}$$

Using $f_{hkl} = \frac{F_{hkl}}{V}$, equation 3.2 expressed in terms of a Fourier series reads as

$$\sum_{hkl} f_{hkl}\, e^{i\mathbf{g}_{hkl}\mathbf{R}} = \sum_{hkl} f_{hkl}\, e^{i\mathbf{g}_{hkl}(\mathbf{R}+\mathbf{r})} = e^{i\mathbf{g}_{hkl}\mathbf{r}} \sum_{hkl} f_{hkl}\, e^{i\mathbf{g}_{hkl}\mathbf{R}}, \tag{3.4}$$

what restricts the reciprocal lattice vector $\mathbf{g}_{hkl}$ to values which satisfy the relation

$$\mathbf{g}_{hkl}\mathbf{r} = 2\pi n \tag{3.5}$$

as with the integer numbers $n\ \varepsilon\ \mathbb{Z}$ the term

$$e^{i\mathbf{g}_{hkl}\mathbf{r}} = 1 \tag{3.6}$$

needs to be fulfilled. In that way, the reciprocal lattice is a mathematical description constructed on the direct lattice where equation 3.6 can be seen as a definition. In the case that the fundamental translations (the basis vectors) are all perpendicular to each other, the relation between the direct lattice and the reciprocal one is particularly simple. In this case the fundamental translations of the reciprocal lattice are parallel to those of the direct lattice. The edge lengths of the translations are then inversely proportional to the edge lengths of the associated translations of the direct crystal lattice. For the non-orthogonal case, the relations between the two representations are best shown in terms of the vector relations

$$\mathbf{a}^* = 2\pi\frac{\mathbf{b}\times\mathbf{c}}{\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})} \qquad \mathbf{b}^* = 2\pi\frac{\mathbf{c}\times\mathbf{a}}{\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})} \qquad \mathbf{c}^* = 2\pi\frac{\mathbf{a}\times\mathbf{b}}{\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})}, \tag{3.7}$$

what explains the orientation of the reciprocal lattice in terms of the direct one; Every fundamental translation of one lattice is oriented perpendicularly to the remaining two

fundamental translations of the second one. This is expressed particularly by

$$\mathbf{a}_i^* \cdot \mathbf{a}_j = \delta_{ij} \begin{cases} = 1, \text{for} i = j \\ = 0, \text{for} i \neq j \end{cases} \tag{3.8}$$

The volume V of the real space unit cell is given by the scalar triple product of the vectors with $V = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) \equiv \mathbf{a} \cdot \mathbf{b} \times \mathbf{c}$.
In the same way as shown in the equations 3.7, expressions for the direct lattice in terms of reciprocal vectors are defined. Using the volume of the reciprocal cell $V^* = \mathbf{a}^* \cdot (\mathbf{b}^* \times \mathbf{c}^*)$, the direct lattice vectors are

$$\mathbf{a} = 2\pi \frac{\mathbf{b}^* \times \mathbf{c}^*}{V^*} \qquad \mathbf{b} = 2\pi \frac{\mathbf{c}^* \times \mathbf{a}^*}{V^*} \qquad \mathbf{c} = 2\pi \frac{\mathbf{a}^* \times \mathbf{b}^*}{V^*} \tag{3.9}$$

In the same manner as a real space lattice vector is defined in equation 3.1, the reciprocal counterpart with its above defined vectors can be written as

$$\mathbf{g}_{hkl} = h\mathbf{a}^* + k\mathbf{b}^* + l\mathbf{c}^* \tag{3.10}$$

where the integer values h, k and l of the linear combination are the Laue indices of the crystal lattice plane. This is of particular interest when describing the scattering of X-rays from such a (hkl)-plane. There are two equivalent representations, which are discussed later on in *Chapter 3.3*.

3.2 The Reduced Cell

In the last chapter, a description of an ideal crystal structure has been introduced in terms of translational repetition of the unit cell in three dimensions. The vectors of the direct lattice are three-dimensional vectors and they can be explicitly written as

$$\mathbf{a} = \begin{pmatrix} a \\ 0 \\ 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b\cos\gamma \\ b\sin\gamma \\ 0 \end{pmatrix} \quad \text{and} \quad \mathbf{c} = \begin{pmatrix} c\cos\beta \\ \frac{c}{\sin\gamma}(\cos\alpha - \cos\beta\cos\gamma) \\ \frac{V}{ab\sin\gamma} \end{pmatrix}, \tag{3.11}$$

with the volume of the parallelepiped

$$V = abc\sqrt{1-\cos^2\alpha-\cos^2\beta-\cos^2\gamma+2\cos\alpha\cos\beta\cos\gamma}. \tag{3.12}$$

The representation of the vectors in equation 3.11 is only one possible representation. The defined vectors follow a right-handed coordinate system based on Figure 3.2. The choice of this unit cell is not unique. It is therefore possible to describe one and the same crystal with different sets of lattice vectors. Casually formulated, two conditions are usually advised when searching for a unit cell: i) if there is more than one choice, the lattice with highest symmetry should be preferred and ii) the unit cell based on the three shortest lattice vectors should be chosen. A unit cell which meets these conditions is called a *reduced cell*.

There are different conditions formulated in crystallography, which allow systematic determination of the reduced unit cell. The first is based on formulations published 1928 by P. Niggli [42]. A unit cell which fulfills the proposed conditions is called a *Niggli's reduced cell* or a unit cell which fulfills *Niggli's criteria for a reduced cell*. A so obtained cell provides an unambiguous choice of the unit cell and is defined independently of lattice symmetry. The procedure requires a classification of the cell and a distinction between two cases. The first case includes cells where all angles are acute (α, β and $\gamma < 90°$). These cells are referred to as type I cells. In the second case (type II cells), all angles are equal to or greater than 90° and therefore obtuse-angled. The type of the cell can be found by evaluating the sign of $\mathbf{T}$ where

$$\mathbf{T} = (\mathbf{a}\cdot\mathbf{b})(\mathbf{b}\cdot\mathbf{c})(\mathbf{c}\cdot\mathbf{a}) = \begin{cases} \text{type I, if } T > 0 \\ \text{type II, if } T \leqslant 0. \end{cases} \tag{3.13}$$

A general formulation of the criteria for a reduced cell can be found in the *International Tables for Crystallography* [44]. If the unit cell is present in the form of its (scalar) parameters, the use of the *scalar-product criteria* is convenient. These formulations of the criteria express the crystallographic restrictions in terms of the three lattice-edge lengths a, b and c with their enclosed angles α, β and γ. An extract of the general criteria for type-I and type-II cells is given in Table 3.1 below.

Table 3.1: General criteria for a reduced cell according to the *International Tables for Crystallography* [44]

Cell	Type I	Type II
Angles	α, β and $\gamma < 90°$	α, β and $\gamma \geqslant 90°$
T is	> 0	$\leqslant 0$
Condition (i)	$a^2 \leqslant b^2 \leqslant c^2$	$a^2 \leqslant b^2 \leqslant c^2$
Condition (ii)	$c \cos\alpha \leqslant \frac{b}{2}$	$c\,\lvert\cos\alpha\rvert \leqslant \frac{b}{2}$
Condition (iii)	$c \cos\beta \leqslant \frac{a}{2}$	$c\,\lvert\cos\beta\rvert \leqslant \frac{a}{2}$
Condition (iv)	$b \cos\gamma \leqslant \frac{a}{2}$	$b\,\lvert\cos\gamma\rvert \leqslant \frac{a}{2}$
Condition (v)	–	$(b\,c\,\lvert\cos\alpha\rvert + a\,c\,\lvert\cos\beta\rvert + a\,b\,\lvert\cos\gamma\rvert) \leqslant \frac{a^2+b^2}{2}$

One special characteristic resulting from the type II conditions should be emphasized here; The conditions not only uniquely defines the lengths a, b and c, but also limit the angles to the range $60° \leqslant \alpha, \beta, \gamma \leqslant 120°$ [44, 45].

Another form of reduced cells has been proposed by M. J. Buerger in 1957 [25]. The method transforms any unit cell into one based on the shortest three non-coplanar vectors. A cell which is reduced according to this is called a *Buerger cell* [46]. However, the cell is not necessarily unique if determined only by this algorithm.
A possible approach for this algorithm includes a set of three independent reciprocal lattice vectors (*cf.* equation 3.10) with the corresponding Laue indices expressed in matrix form. This can be written as

$$\mathbf{G} = \begin{pmatrix} g_{x_1} & g_{y_1} & g_{z_1} \\ g_{x_2} & g_{y_2} & g_{z_2} \\ g_{x_3} & g_{y_3} & g_{z_3} \end{pmatrix} \quad \text{and} \quad \mathbf{H} = \begin{pmatrix} h_1 & h_2 & h_3 \\ k_1 & k_2 & k_3 \\ l_1 & l_2 & l_3 \end{pmatrix}. \tag{3.14}$$

As the direct vectors must be a solution to all reciprocal lattice vectors, the relation

$$\mathbf{A}^T = 2\pi\mathbf{G}^{-1}\mathbf{H}^T \tag{3.15}$$

needs to be fulfilled. Here, $\mathbf{A}$ is the matrix which contains the real space vectors with $\mathbf{A} = (\mathbf{a},\mathbf{b},\mathbf{c})^T$. Equation 3.15 connects the direct and the reciprocal lattice and if the triplets formed for $\mathbf{G}$ belong to the same system, its inverse $\mathbf{G}^{-1}$ multiplied with the corresponding Laue indices will lead to the according real space vectors. The algorithm can therefore be performed by multiplying $\mathbf{G}^{-1}$ with vectors of the form

$$\mathbf{m} = \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix}. \tag{3.16}$$

The vector entries m_i are integer values imaging the Laue indices in a reasonable range. As an example, this range can be chosen in an interval of $m_i \; \varepsilon$ (-3,3) [10]. Vectors of the reduced cell can be obtained, if the $\mathbf{m}$ matches $(h_1,h_2,h_3)^T$, $(k_1,k_2,k_3)^T$ or $(l_1,l_2,l_3)^T$. The unit cell parameters can then be obtained by sorting the lengths of the products $2\pi\mathbf{G}^{-1}\cdot\mathbf{m}$ in ascending order. The three shortest entries with non-coplanar vectors are the new parameters of the unit cell. Furthermore, these vectors describe the reduced basis of this new unit cell.

3.2.1 Notation of Planes and Directions

In crystallography, there is a set of notations which is used to indicate different geometrical information such as points, directions and planes. To avoid any possibility of confusion, the nomenclature present in crystallography is briefly introduced.
As discussed earlier, any lattice node (point) of the direct lattice can be described by the linear combination of the lattice vectors by $\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$ where u, v and w are any integer values. Any other point of the lattice can be reached by adding the offset $\mathbf{r}_0$ relative to the origin of the basis with $\mathbf{p} = \mathbf{r}_0 + u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$.

Due to the symmetry properties of the crystal, the lattice can also be described in terms of infinitely extended, equidistant sets of planes. Such planes are usually described using Miller indices. The Miller indices are a set of three integer numbers, relatively prime and indicated by (hkl) [47]. Each such a triplet describes a specific plane of the lattice.

If the description relates to all symmetrically equivalent planes, the notation $\{hkl\}$ is used[2]. Vectors (or directions) are denoted differently within the crystal lattice. The notation $[uvw]$ is used to describe a specific direction. All symmetry equivalent directions and vectors are then denoted by $\langle uvw \rangle$.

A slightly different notation is used in the later described formalism. Instead of (hkl), the notation (uvw) is used to describe a plane. More precisely, to describe the crystallographic plane parallel to the substrate. This is simply to avoid any possible confusion with the Laue indices of diffraction hkl. In this way, the Laue indices are always used without brackets or parentheses, unless noted. The notations are summarized in the table below.

Table 3.2: Description of crystallographic notations

Notation	Describes
(hkl)	a crystallographic plane
$\{hkl\}$	symmetrically equivalent planes
$[uvw]$	a direction in the lattice
$\langle uvw \rangle$	symmetry equivalent directions in the lattice
hkl	the Laue indices of diffraction
(uvw)	the contact plane

[2]For example the plane $\{100\}$ of a cubic crystal system indicates at the same time the planes (100), ($\bar{1}$00), (010), (0$\bar{1}$0), (001) and (00$\bar{1}$). The bar is used for negative indices. This corresponds to the six surface areas of a cube.

3.3 X-ray Diffraction

Diffraction experiments are used to expose the inner structure of crystalline materials. A necessary (but not sufficient) requirement to resolve such a structure needs radiation with a wavelength comparable to, or smaller than the investigated dimensions of the repeated unit [6]. As distances of atomic and molecular structures are typically in the range of 1 Å, X-rays with a wavelength region between 10^{-12} m to 10^{-10} m are particularly suitable. Several conditions need to be fulfilled in order to observe diffraction. The basic conditions of diffraction were first formulated by Max von Laue[3] as well as by the father-and-son team Lawrence and William Henry Bragg[4]. In a diffraction experiment, the sample is irradiated by a monochromatic X-ray beam and the intensity of the scattered beam is recorded with a detector. A schematic drawing of the beam geometry is shown in Figure 3.3a. The incident beam impinges the sample surface under the angle α_i and is described in terms of the wave vector $\mathbf{k}_i$. The scattered beam is represented by its wave vector $\mathbf{k}_f$ and encloses the angle α_f with the lattice plane. The length of the incident wave vector $|\mathbf{k}_i|$ is $2\pi/\lambda$. Scattering of the X-rays from the electron cloud is an elastic scattering process (Thomson scattering), thus the length of the scattered wave vector is written as

$$|\mathbf{k}_f| = |\mathbf{k}_i| = k = \frac{2\pi}{\lambda}. \tag{3.17}$$

The momentum transfer due to the scattering process is the quantity which is probed during a diffraction experiments. It is represented by the scattering vector

$$\mathbf{q} = \mathbf{k}_f - \mathbf{k}_i. \tag{3.18}$$

In the previous chapter, the definition of reciprocal lattice vector $\mathbf{g}_{hkl}$ was given. According to the *Laue condition*, a diffraction peak is observed only if the length as well as the direction of the scattering vector is equal to a reciprocal lattice vector. This is expressed as

$$\mathbf{q} = \mathbf{g}_{hkl}. \tag{3.19}$$

[3] Awarded the Nobel Prize in Physics 1914 "for his discovery of the diffraction of X-rays by crystals."

[4] Jointly awarded the Nobel Prize in Physics 1915 "for their services in the analysis of crystal structure by means of X-rays."

The indices *h, k,* and *l* are the indices of the probed plane in the direct lattice, where the net plane distance between parallel planes is d_{hkl} (Figure 3.3). In terms of the reciprocal lattice, the indices describe reciprocal lattice points.

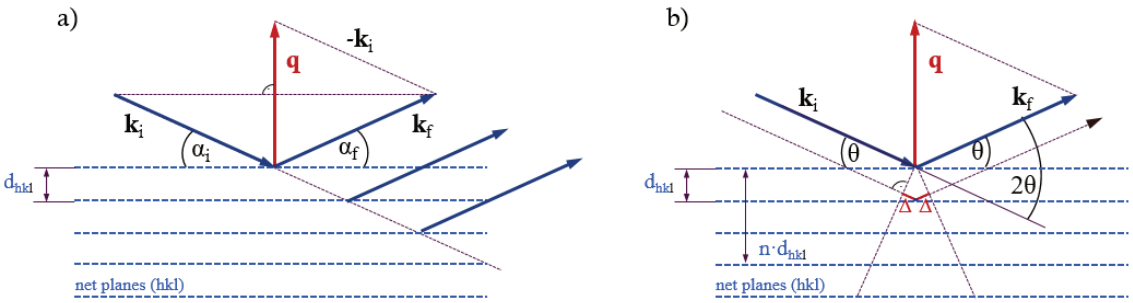


Figure 3.3: Schematic drawing of a diffraction geometry to a) show the scattering vector **q** formed through the difference in scattered and incident wave vectors and b) indicate the path difference Δ with the setup in specular condition.

Due to the geometric relation between direct and reciprocal lattice, an equivalent formulation for diffraction is possible: "Diffraction occurs only if the scattering vector **q** is perpendicular to a net plane indicated by (*hkl*). In this simplified image, the distance of the parallel direct lattice planes d_{hkl} can thus be calculated with

$$d_{hkl} = \frac{2\pi}{|\mathbf{g}_{hkl}|} = \frac{2\pi}{q}. \tag{3.20}$$

The wave vectors and the reciprocal space vectors are in general three-dimensional vectors. Depending on the experiment, different components of the scattering vector are accessible. As we will see later, the Laue condition needs to be fulfilled also for individual components of **q**. It is therefore possible to derive three-dimensional periodicities of a crystal such as the edge lengths of the unit cell and their enclosed angles. The Laue condition gives an expression for diffraction in terms of a reciprocal lattice vector. An analogous description for using real space components only is expressed by *Bragg's law*. It can be seen as a real space equivalent and is divided in two sub-conditions. Again, a monochromatic primary beam ($\mathbf{k}_i$) is considered, which is elastically scattered leading to a scattered beam $\mathbf{k}_f$. From geometric considerations using Figure 3.3 b) it can be derived that the path difference is

$$\Delta = d_{hkl} \sin\theta. \tag{3.21}$$

Ideally, scattered waves interfere completely constructively when they are in phase, and destructively when they are half a cycle out of phase. The path difference needs therefore to be an integer multiple of the used wavelength with $\Delta = n\lambda$.

The angle $\theta = 2\theta/2$ can be expressed as half of the scattering angle 2θ. This leads to the first condition, the known *Bragg equation*

$$n\lambda = 2d_{hkl} \sin\left(\frac{2\theta}{2}\right). \tag{3.22}$$

The second condition to be fulfilled restricts the angles of incident and scattered beam. The primary and the scattered beam have to enclose the same angle θ to the (*hkl*)-plane. With regard to Figure 3.3 this means $\alpha_i = \alpha_f = \theta$. This is called a specular condition and results in a scattering vector $\mathbf{q}$ which is always perpendicular to the net plane. The two consequences from these conditions are therefore that diffraction occurs only at discrete scattering angles of 2θ when the geometry fulfills the specular condition. To compare diffraction patterns independently from the used wavelength, an expression for the length of the scattering vector depending from the scattering angle can be derived. Combination of equation 3.20 and 3.22 allows to derive this expression where the length of the scattering vector

$$q = \frac{4\pi}{\lambda} \sin\left(\frac{2\theta}{2}\right) \tag{3.23}$$

is usually given in units of Å^{-1} or nm^{-1}. Equation 3.23 is especially interesting in this work as it allows a reciprocal space representation of the diffraction pattern measured in real space (2θ). If the sample is probed in specular condition, only the out-of-plane component $\mathbf{q}_z$ of the scattering vector is probed which is here referred to as *specular scan* or *specular peak*. The importance if this peak will we discussed in the following.

Chapter 4

Methods

4.1 Grazing Incidence X-Ray Diffraction (GIXD)

For crystallographic investigations, grazing-incidence X-ray diffraction is a widely used technique. The surface sensibility makes is attractive for probing (organic) thin films. The important characteristics of this method (as it is used for this work) are the small incidence angle of the primary X-ray beam combined with a larger two-dimensional detector to measure large areas of reciprocal space. Detailed information about the experimental method and corresponding data processing can be found, among others, in [39, 45, 48]. In GIXD, the incidence angle α_i is close to or below the critical angle α_c of the sample material. For any angle of incidence smaller than the critical angle, X-rays will undergo total external reflection. An evanescent wave-field is formed parallel to the surface which decays exponentially into the sample surface. Intensity enhancement of the transmitted wave and a penetration depth in the region of a few nm are the consequences of this phenomenon [48]. As the signal from the crystalline substrate is weakened, potential diffraction peaks from the sample are easier to detect. In an ideal GIXD experiment, the substrate signal is significantly lowered compared to experiments with higher incident angles. As the penetration depth is in the of range 5 nm, it is not possible to avoid the substrate signal.

The geometry of a GIXD experiment is shown in Figure 4.1a. The primary X-ray beam with wave vector $\mathbf{k}_i$ encloses the incident angle α_i with the sample surface. Just as it was used to introduce the fundamentals of Bragg diffraction, the scattered beam is described in terms of its wave vector $\mathbf{k}_f$. The geometry of the two wave vectors determine the scattering vector with $\mathbf{q} = \mathbf{k}_f - \mathbf{k}_i$. Diffraction peaks are obtained, where the Laue condition $\mathbf{q} = \mathbf{g}_{hkl}$ is fulfilled (*cf.* equation 3.18). The measured peaks on the detector

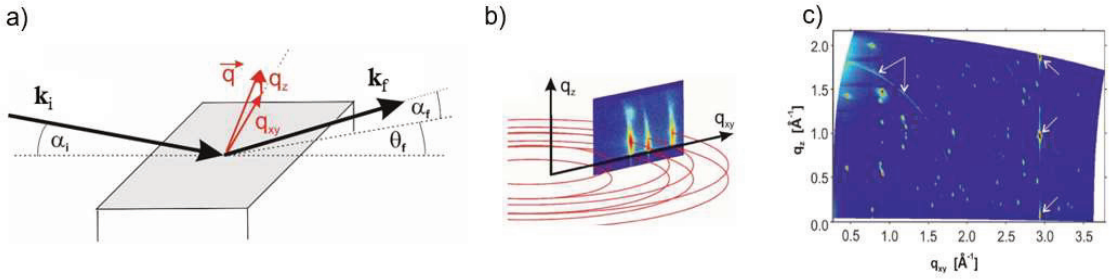


Figure 4.1: (a) Schematic drawing of the beam geometry of a grazing-incidence X-ray diffraction (GIXD) experiment [10]. (b) Concentric rings representing the lattice points of a fibre-textured sample in reciprocal space. A cut through the rings shows a reciprocal space map with a colour code for the measured intensities [10]. (c) Typical reciprocal space map of a GIXD experiment after data processing [13].

plane are defined intensities with associated two-dimensional coordinates. This is the actual diffraction pattern in real (or pixel) space. A so-called reciprocal space map is then a result of the data processing[1]. This map contains the intensities as a function of the in-plane components q_{xy} and the out-of-plane components q_z of the scattering vector (Figure 4.1c).

On the pattern shown, two features such as a Debye-Scherrer ring as well as the substrate peaks are indicated with white arrows [10]. Another "feature" is clearly visible in this image; Along the vertical axis of q_z, a wedge with no diffraction information appears. Due to the beam geometry, this region in reciprocal space is not accessible with GIXD experiments. As stated later, specular XRD measurements can help to provide additional data where $q_{xy} = 0$.

The in-plane component of the scattering vector is the projection of the two surface-parallel components q_x and q_y with $q_{xy} = \sqrt{q_x^2 + q_y^2}$ and a result from a static GIXD measurement. For samples, where the crystallites exhibit a preferred orientation along the out-of-plane direction (fibre-textured samples), the points in reciprocal space appear as concentric rings around that axis. The reciprocal space map is nothing else than a cut through these rings which reduce to points in a two-dimensional image (Figure 4.1b). If the crystalline thin film shows both, preferred in- and out-of-plane orientation, the reciprocal lattice points appear in the form of distinct points in reciprocal space. In this case, the sample has to be rotated around its surface normal. The integration over a full rotation provides then a reciprocal space map containing all diffraction information of the sample. The combination of several patterns (each measured at different rotational

[1] Which can be conveniently done using the comprehensive software tool *GIDVis* [39].

angles) allows three-dimensional representation of the the scattering vector [45].
For this work however, the two-dimensional diffraction data with their components q_{xy} and q_z are used for indexing. The underlying formalism for this procedure will be introduced in the following chapter.

4.2 The Role of the Specular Scan

In general, the determination of a unit cell solution through indexing of a GIXD pattern does not necessarily require the presence of a specular peak. The indexing formalism which is used in this work, however, considers the contact plane of the sample with respect to the substrate surface via a separate set of parameters. This makes the algebraic solution for the unit cell a more challenging task. A specular scan provides an additional peak position, which makes the computational determination of the unit cell solution significantly more efficient.

The fundamentals to measure such a specular peak were already discussed in the last chapter; The incident angle of the primary beam and the angle of the scattered beam are equal with the value of θ. The scattering vector $\mathbf{q}$ always points perpendicular to the plane parallel to the substrate surface. That means, with z direction defined as out-of-plane direction, only the z component q_z is probed with this method. In-plane periodicities cannot be measured this way.

As discussed previously, the scattering vector is split in two components for the indexing procedure. The in-plane components q_{xy} and the out-of-plane components q_z. The measurement in specular condition provides one additional peak where $q_{xy} = 0$ and $q_z = q_{specular}$. This contribution is of special interest as this peak (series) is located in an inaccessible region of reciprocal space map in GIXD measurements. This additional information is treated as an additional input parameter, which reduces complexity of the algorithm and the associated computational effort.
When a specular scan is performed with an X-ray diffractometer, out-of-plane periodicities are accessible in $\theta/2\theta$ operation mode. The so obtained diffraction pattern is expressed as intensity of the scattered beam over scattering angle 2θ. After identification of the substrate diffraction peak(s), the out-of-plane component of the scattering vector q_z can be deduced with equation 3.23.

Chapter 5

Indexing Formalism and Mathematical Preparation

The following chapter provides a summary of the mathematical toolkit necessary for this work. The theoretical treatment and the formulae are adopted from publications of J. Simbrunner et al., unless denoted [10, 11, 12]. For the sake of a better overview, the two cases with different contact plane descriptions are discussed separately. In each case, a numerical optimization algorithm is applied on the derived unit cell solutions. As the algorithm is the same for both, the special case and the general case, it is summarized in one sub-section of this chapter. The same mathematical notation is retained where the unit cell parameters of the direct cell are denoted as a, b, c, α, β and γ. The cell parameters in reciprocal space are a^*, b^*, c^*, α^*, β^* and γ^*. The equations for conversion from reciprocal space to real space and vice versa are summarized in the Table 5.1 at the end of this introduction.

The analytical derivation is based on a laboratory coordinate system where the xy plane is oriented parallel to the substrate surface. This plane parallel to the substrate surface (contact plane) is described by the means of the three Miller indices (uvw). Two cases can be distinguished for derivation of the unit cell constants: (i) The non-rotated case where the (001) lattice plane is parallel to the surface and (ii) the rotated case with a general contact plane denoted by (uvw). The terms 'special case', '(001)-case' and 'non-rotated case' are all synonyms for the configuration with the (001) net plane being parallel to the xy plane. The analytical description starts with the matrix $\mathbf{A}^*_{001}$ describing a general crystallographic cell in terms if its lattice constants. In a general case, this cell has to be rotated around the zone axis (*cf.* Figure 5.1). This axis is defined by the surface normal vector $\boldsymbol{\sigma}_1$ of the (001) plane and the new normal vector $\boldsymbol{\sigma}_2$ of the (uvw) plane.

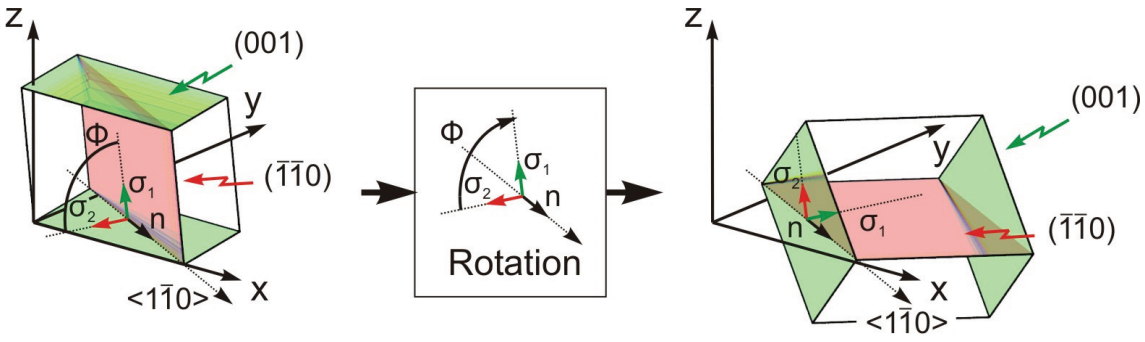


Figure 5.1: Schematic drawing of a triclinic cell to demonstrate the parameters of rotation. For description using a general contact plane, all vectors and planes have to be rotated around the zone axis defined by the vector **n**. It can be constructed using the surface normal vectors $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$, as indicated with the image in the center.

For example, for transformation from the (001) plane to the (-1-10) plane, all planes and vectors have to be rotated around the zone axis [1-10] by the rotation angle Φ. The transformation of the lattice planes and of the crystallographic directions is illustrated in Figure 5.1 from left to right.

Table 5.1: Summary of relations to convert real space parameters a, b, c, α, β and γ to reciprocal space parameters a^*, b^*, c^*, α^*, β^* and γ^* and vice versa. The volume is denoted by V.

$$a^* = \frac{2\pi bc\sin\alpha}{V} \qquad \cos\alpha^* = \frac{\cos\beta\cos\gamma-\cos\alpha}{\sin\beta\sin\gamma} \qquad \sin\alpha^* = \frac{V}{abc\sin\beta\sin\gamma}$$

$$b^* = \frac{2\pi ac\sin\beta}{V} \qquad \cos\beta^* = \frac{\cos\alpha\cos\gamma-\cos\beta}{\sin\alpha\sin\gamma} \qquad \sin\beta^* = \frac{V}{abc\sin\alpha\sin\gamma}$$

$$c^* = \frac{2\pi ab\sin\gamma}{V} \qquad \cos\gamma^* = \frac{\cos\alpha\cos\beta-\cos\gamma}{\sin\alpha\sin\beta} \qquad \sin\gamma^* = \frac{V}{abc\sin\alpha\sin\beta}$$

$$\cos\alpha = \frac{\cos\beta^*\cos\gamma^*-\cos\alpha^*}{\sin\beta^*\sin\gamma^*} \qquad \cos\beta = \frac{\cos\alpha^*\cos\gamma^*-\cos\beta^*}{\sin\alpha^*\sin\gamma^*} \qquad \cos\gamma = \frac{\cos\alpha^*\cos\beta^*-\cos\gamma^*}{\sin\alpha^*\sin\beta^*}$$

$$V = abc\sqrt{1-\cos^2\alpha-\cos^2\beta-\cos^2\gamma+2\cos\alpha\cos\beta\cos\gamma} = abc\sin\alpha^*\sin\beta\sin\gamma$$

5.1 Special Case: Contact Plane (001)

Throughout indexing of GIXD patterns, sets of Laue indices h, k and l are assigned to individual Bragg peaks. As emphasized, these peaks are visible only if a scattering vector $\mathbf{q}$ images a vector of the reciprocal lattice $\mathbf{g}$. This key relation allows to describe diffraction peaks in terms of reciprocal lattice vectors. For a contact plane with indices (001), such a vector can be expressed as

$$\mathbf{g} = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix} = \mathbf{A}^*_{001} \begin{pmatrix} h \\ k \\ l \end{pmatrix}. \tag{5.1}$$

The quadratic matrix $\mathbf{A}^*_{001}$ includes the crystallographic cell constants and can be explicitly written as

$$\mathbf{A}^*_{001} = \begin{pmatrix} a^* \sin\beta^* \sin\gamma & 0 & 0 \\ -a \sin\beta^* \cos\gamma & b \sin\alpha^* & 0 \\ a^* \cos\beta^* & b^* \cos\alpha^* & c^* \end{pmatrix}. \tag{5.2}$$

Using the inverse relationship

$$\mathbf{A}_{001} = 2\pi \mathbf{A}^{*-1}_{001}, \tag{5.3}$$

the matrix is convertible and can be stated through the real space lattice vectors $\mathbf{a}_0$, $\mathbf{b}_0$ and $\mathbf{c}_0$ by

$$\mathbf{A}_{001} = \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{b}_0 \\ \mathbf{c}_0 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ b\cos\gamma & b\sin\gamma & 0 \\ c\cos\beta & -c\sin\beta\cos\alpha^* & c\sin\beta\sin\alpha^* \end{pmatrix}. \tag{5.4}$$

The determinant of this matrix yields the volume V of the crystallographic unit cell by

$$V = \det(\mathbf{A}_{001}) = abc \sin\alpha^* \sin\beta \sin\gamma. \tag{5.5}$$

The reciprocal lattice vector stated in equation (6.14) can be summarized and expressed in two components where g_{xy} comprises the in-plane part and g_z the out-of-plane part of the vector. With the geometrical relations stated in Table 5.1 above, the components can be explicitly written as

$$g^2_{xy} = g^2_x + g^2_y = h^2 \left(\frac{2\pi}{a \sin\gamma} \right)^2 + k^2 \left(\frac{2\pi}{b \sin\gamma} \right)^2 - 2hk \frac{2\pi}{a\sin\gamma} \frac{2\pi}{b\sin\gamma} \cos\gamma \tag{5.6}$$

and

$$g_z = h a^* cos\beta^* + k b^* \cos\alpha^* + l c^*. \tag{5.7}$$

With this separation, step-wise determination of the unit cell parameters becomes feasible. The cell parameters a, b and γ are accessible by evaluation of the in-plane part (equation (5.6)) under systematic variation of the Laue indices h and k. The out-of-plane part stated in equation (5.7) can be used in a subsequent step to derive the remaining three parameters α, β and c. The implementation of the here stated formalism is summarized in Chapter 6. Prior to implementation, some mathematical adaption and preparation is necessary.

The unit cell constants can be derived by making use of the above-mentioned equations (5.6) and equation (5.7). Upon evaluation of the equation for the in-plane component, sets of partial solutions in the form of (a,b,γ) can be derived. In this form, the equation is not solvable as (i) the desired parameters appear in quadratic form and (ii) one equation is not analytically solvable for five unknown parameters. For the assumption of fixed values for the Laue indices h and k, the number of unknowns is reduced to three. To evaluate equation (5.6), it is linearized using the following substitutions:

$$\left(\frac{2\pi}{a\sin\gamma}\right)^2 := X, \qquad \left(\frac{2\pi}{b\sin\gamma}\right)^2 := Y \qquad \text{and} \qquad \frac{2\pi}{a\sin\gamma}\frac{2\pi}{b\sin\gamma}\cos\gamma := Z \tag{5.8}$$

If the index i is considered to describe the q_{xy} components of the i-th reflection, the Laue condition can be written as

$$g_{xy_i}^2 = h_i^2 X + k_i^2 Y - 2h_i k_i Z = q_{xy_i}^2 \tag{5.9}$$

For a set of three data points and their in-plane components, respectively, equation (5.9) can be expressed in terms of a matrix

$$\begin{pmatrix} h_1^2 & k_1^2 & -2h_1k_1 \\ h_2^2 & k_2^2 & -2h_2k_2 \\ h_3^2 & k_3^2 & -2h_3k_3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} q_{xy_1}^2 \\ q_{xy_2}^2 \\ q_{xy_3}^2 \end{pmatrix} \tag{5.10}$$

where solutions for the $(X,Y,Z)^{\mathrm{T}}$ vectors are derivable by inserting permutations of (h_i,k_i) tuples and evaluating the relation for linearly-independent sets of q_{xy_i}. One task of the Indexing Routine described in the following chapter is the generation of all the possible permutations with subsequent evaluation of each matrix. By use of the substitutions (5.8), solutions for (a,b,γ) are thereby derived. For the remaining unit cell constants, equation (5.7) is rewritten using the following substitutions:

$$a^* cos\beta^* := \kappa, \qquad b^*\cos\alpha^* := \lambda \qquad \text{and} \qquad c^* := \rho. \tag{5.11}$$

With the three Laue indices h, k and l, the Laue condition for the out-of-plane component is hereafter given as

$$g_{z_i} = h_i\kappa + k_i\lambda + l_i\rho = q_{z_i} \tag{5.12}$$

and the according linear system of equations is expressed as

$$\begin{pmatrix} h_1 & k_1 & l_1 \\ h_2 & k_2 & l_2 \\ h_3 & k_3 & l_3 \end{pmatrix} \begin{pmatrix} \kappa \\ \lambda \\ \rho \end{pmatrix} = \begin{pmatrix} q_{z_1} \\ q_{z_2} \\ q_{z_3} \end{pmatrix} \tag{5.13}$$

where the q_{z_i} are the out-of-plane components of an independent Bragg peak series. With the calculated terms of $(\kappa, \lambda, \rho)^{\mathrm{T}}$, the reciprocal lattice constants are determined (*cf.* substitutions (5.11)). From that, the unit cell parameters of the direct lattice can be derived with the relations given in Table 5.1.
The two systems, namely the one for q_{xy} and the one for q_z, are evaluated one after another and yield sets of solutions of the form $(a, b, c, \alpha, \beta, \gamma)$ for a (001) contact plane.

5.2 General Case: Contact Plane (uvw)

The above-stated special case is no longer sufficient for the indexing procedure, if the crystallographic plane parallel to the substrate surface can not be described by a $(00w)$-set of Miller indices. In the general case, the matrix $\mathbf{A}^*_{001}$ (*cf.* equation (5.2)) needs to be rotated around the zone axis as schematically shown in Figure 5.1. The rotation is achieved by applying a rotation matrix $\mathbf{R}$ on $\mathbf{A}^*_{001}$, which follows the condition $\mathbf{R}^{-1} = \mathbf{R}^T$. It can be shown that the application of a general rotation on the real space lattice vectors leads to identical rotation in reciprocal space. With equation (5.3), this can be formulated by

$$\mathbf{R}\begin{pmatrix} \mathbf{a}_0 \\ \mathbf{b}_0 \\ \mathbf{c}_0 \end{pmatrix}^{\mathrm{T}} = \mathbf{R}\mathbf{A}^{\mathrm{T}}_{001} = [\mathbf{A}_{001}\mathbf{R}^{\mathrm{T}}]^{\mathrm{T}} = 2\pi[\mathbf{A}^{*^{-1}}_{001}\mathbf{R}^{\mathrm{T}}]^{\mathrm{T}} = 2\pi[(\mathbf{R}\mathbf{A}^*_{001})^{-1}]^{\mathrm{T}}. \tag{5.14}$$

The rotation $\mathbf{R}$ applied on real space lattice vectors yields the rotated lattice vectors of the form $\mathbf{a} = \mathbf{R}\mathbf{a}_0$, $\mathbf{b} = \mathbf{R}\mathbf{b}_0$ and $\mathbf{c} = \mathbf{R}\mathbf{c}_0$. This allows to state a general expression of the inverse relation (5.3) where

$$\mathbf{A} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{pmatrix} = 2\pi\mathbf{A}^{*^{-1}} \tag{5.15}$$

and $\mathbf{A}^* = \mathbf{R}\mathbf{A}^*_{001}$.

The axis of rotation is defined by the unit vector $\mathbf{n}$ which can be derived by the vector product

$$\mathbf{n} = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = \frac{\boldsymbol{\sigma}_1 \times \boldsymbol{\sigma}_2}{|\boldsymbol{\sigma}_1 \times \boldsymbol{\sigma}_2|}, \tag{5.16}$$

where the surface normal vector of the (001) plane $\boldsymbol{\sigma}_1$ and the surface normal vector of the (uvw) plane $\boldsymbol{\sigma}_2$ are defined by

$$\boldsymbol{\sigma}_1 = \mathbf{A}^*_{001} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\sigma}_2 = \mathbf{A}^*_{001} \begin{pmatrix} u \\ v \\ w \end{pmatrix}. \tag{5.17}$$

The angle of rotation Φ is the angle between the two normal vectors (*cf.* Figure (5.1), center). It can be derived by the scalar product

$$\cos\Phi = \frac{\boldsymbol{\sigma}_1 \cdot \boldsymbol{\sigma}_2}{|\boldsymbol{\sigma}_1|\,|\boldsymbol{\sigma}_2|}. \tag{5.18}$$

With this, the rotation matrix $\mathbf{R}$ can be explicitly written as

$$\mathbf{R} = \begin{pmatrix} n_1^2(1-\cos\Phi)+\cos\Phi & n_1n_2(1-\cos\Phi)+n_3\sin\Phi & n_1n_3(1-\cos\Phi)-n_2\sin\Phi \\ n_1n_2(1-\cos\Phi)-n_3\sin\Phi & n_2^2(1-\cos\Phi)+\cos\Phi & n_2n_3(1-\cos\Phi)-n_1\sin\Phi \\ n_1n_3(1-\cos\Phi)+n_2\sin\Phi & n_2n_3(1-\cos\Phi)-n_1\sin\Phi & n_3^2(1-\cos\Phi)+\cos\Phi \end{pmatrix} \tag{5.19}$$

Using this notation, $\mathbf{R}$ describes a rotation of the lattice vectors and planes by the angle of Φ around the zone axis $\mathbf{n}$. With the substitutions

$$z_a = \frac{2\pi}{a\sin\gamma} \quad \text{and} \quad z_b = \frac{2\pi}{b\sin\gamma}, \tag{5.20}$$

the components of the unit vector $\mathbf{n}$ are

$$n_1 = \frac{uz_a\cos\gamma - vz_b}{(u^2z_a^2 + v^2z_b^2 - 2uvz_az_b\cos\gamma)^{1/2}} \tag{5.21}$$

and

$$n_2 = \frac{uz_a\sin\gamma}{(u^2z_a^2 + v^2z_b^2 - 2uvz_az_b\cos\gamma)^{1/2}}. \tag{5.22}$$

The combination of equation (5.17) and equation (5.18) allows to state a different expression for the angle of rotation where

$$\cos\Phi = \frac{ua^*cos\beta^* + vb^*\cos\alpha^* + wc^*}{[u^2z_a^2 + v^2z_b^2 - 2uvz_az_b\cos\gamma + (ua^*cos\beta^* + vb^*\cos\alpha^* + wc^*)^2]^{1/2}}. \tag{5.23}$$

In conformity with the expression of a reciprocal lattice vector for a (001) plane (equation (6.14)), a general formulation can now be given by

$$\mathbf{g} = \mathbf{R}\mathbf{A}^*_{001}\begin{pmatrix} h \\ k \\ l \end{pmatrix} = \mathbf{A}^*_{uvw}\begin{pmatrix} h \\ k \\ l \end{pmatrix} \tag{5.24}$$

It is herewith possible to derive explicit expressions for the length $g_{xyz} = \sqrt{g_x^2 + g_y^2 + g_z^2}$ and for the out-of plane part g_z of the reciprocal space vector:

$$g_{xyz}^2 = h^2 z_a^2 + k^2 z_b^2 - 2hkz_a z_b \cos\gamma + (ha^* cos\beta^* + kb^* \cos\alpha^* + lc^*)^2, \tag{5.25}$$

$$\begin{aligned} g_z = [&huz_a^2 + kvz_b^2 - (hv+ku)z_a z_b \cos\gamma \\ &+ (ua^* cos\beta^* + vb^* \cos\alpha^* + wc^*) \\ &\times (ha^* cos\beta^* + kb^* \cos\alpha^* + lc^*)] \\ &/[u^2 z_a^2 + v^2 z_b^2 - 2uvz_a z_b \cos\gamma) \\ &+ (ua^* cos\beta^* + vb^* \cos\alpha^* + wc^*)^2]^{1/2}. \end{aligned} \tag{5.26}$$

In specular condition, the in-plane contribution is zero and there is only an out-of-plane part represented by q_z. This component of the reciprocal lattice vector can be explicitly written as

$$g_{\text{spec}} = [u^2 z_a^2 + v^2 z_b^2 - 2uvz_a z_b \cos\gamma + (ua^* cos\beta^* + vb^* \cos\alpha^* + wc^*)^2]^{1/2} \tag{5.27}$$

By the use of equation (5.25) and equation (5.27), the in-plane component g_{xy} can be described by

$$\begin{aligned} g_{xy}^2 g_{\text{spec}}^2 V^2 \frac{1}{(2\pi)^4} = &(kw-lv)^2 a^2 + (hw-lu)^2 b^2 + (hv-ku)^2 c^2 \\ &+ 2(ku-hv)(hw-lu)bc\cos\alpha \\ &+ 2(hv-ku)(kw-lv)ac\cos\beta \\ &+ 2(hw-lu)(lv-kw)ab\cos\gamma \end{aligned} \tag{5.28}$$

With this, the length of the reciprocal lattice vector can be rewritten as

$$\begin{aligned} g_{xyz}^2 = &h^2 z_a^2 + k^2 z_b^2 - 2hkz_a z_b \cos\gamma \\ &+ \frac{\{g_z g_{\text{spec}} - [huz_a^2 + kvz_b^2 - (hv+ku)z_a z_b \cos\gamma]\}^2}{g_{\text{spec}}^2 - (u^2 z_a^2 + v^2 z_b^2 - 2uvz_a z_b \cos\gamma)} \end{aligned} \tag{5.29}$$

and a general expression for the in-plane component g_{xy} can be achieved:

$$\begin{aligned} g_{xy}^2 g_{\mathrm{spec}}^2 = {} & z_a^2[u^2 g_{xy}^2 + (h g_{\mathrm{spec}} - u g_z)^2] + z_b^2[v^2 g_{xy}^2 + (k g_{\mathrm{spec}} - v g_z)^2] \\ & - 2 z_a z_b \cos\gamma [uv g_{xy}^2 + (h g_{\mathrm{spec}} - u g_z)(k g_{\mathrm{spec}} - v g_z)] \\ & - (hv - ku)^2 z_a^2 z_b^2 \sin^2\gamma. \end{aligned} \tag{5.30}$$

This general formula includes not only unit cell constants and Laue indices, but also two out of the three Miller indices as well as the contribution of the specular scan. For a set of Miller indices with $(00w)$, the equation reduces to the relation formulated for the special case (*cf.* equation (5.6)).

Relation (5.30) is used for the following Indexing Routine to derive the parameters a, b and γ while the Laue indices h and k as well as the two Miller indices u and v are systematically varied. By evaluation of

$$ha^* \cos\beta^* + kb^* \cos\alpha^* + lc^* = \frac{g_z g_{\mathrm{spec}} - huz_a^2 - kvz_b^2 - (hv + ku) z_a z_b \cos\gamma}{(g_{\mathrm{spec}}^2 - u^2 z_a^2 + v^2 z_b^2 + 2uv z_a z_b \cos\gamma)^{1/2}} \tag{5.31}$$

the three remaining parameters α, β and c are derivable upon variation of the third Miller index w. The indexing procedure here is again a gradual process where only sub-sets of unit cell solutions can be determined. In a first part, the set denoted as (u, v, a, b, γ) can be obtained. However, the appearance of the linearized system of equations is different as the information of the contact plane is included. The Miller indices u and v are treated as integer numbers and $u \neq v \neq 0$. To achieve again a solvable linear system of equations, equation (5.30) can be written in a slightly different way as

$$\begin{aligned} & X\left[\frac{1}{g_{xy}^2}\left(h - u\frac{g_z}{g_{\mathrm{spec}}}\right)^2 + \frac{u^2}{g_{\mathrm{spec}}^2}\right] + Y\left[\frac{1}{g_{xy}^2}\left(k - v\frac{g_z}{g_{\mathrm{spec}}}\right)^2 + \frac{v^2}{g_{\mathrm{spec}}^2}\right] \\ & -Z\left[\frac{1}{g_{xy}^2}\left(h - u\frac{g_z}{g_{\mathrm{spec}}}\right)^2 + \frac{u^2}{g_{\mathrm{spec}}^2}\right] = 1 \end{aligned} \tag{5.32}$$

where the following substitutions are used:

$$X = z_a^2\left(1 - v^2 z_b^2 \frac{\sin^2\gamma}{g_{\mathrm{spec}}^2}\right), \tag{5.33}$$

$$Y = z_b^2\left(1 - v^2 z_a^2 \frac{\sin^2\gamma}{g_{\mathrm{spec}}^2}\right) \tag{5.34}$$

and

$$Z = z_a z_b\left(\cos\gamma - uv z_a z_b \frac{\sin^2\gamma}{g_{\mathrm{spec}}^2}\right). \tag{5.35}$$

The matrix entries are now coupled terms using the specular peak(s) and both, the in-plane part and the out-of-plane part of a series of three linearly independent Bragg peaks. With

$$f_{i_1}(h_i) = \frac{1}{g_{xy_i}^2}\left(h_i - u\frac{g_{z_i}}{g_{\text{spec}}}\right)^2 + \frac{u^2}{g_{\text{spec}}^2}, \tag{5.36}$$

$$f_{i_2}(k_i) = \frac{1}{g_{xy_i}^2}\left(k_i - v\frac{g_{z_i}}{g_{\text{spec}}}\right)^2 + \frac{v^2}{g_{\text{spec}}^2} \tag{5.37}$$

and

$$f_{i_3}(h_i,k_i) = -\frac{2}{g_{xy_i}^2}\left(h_i - u\frac{g_{z_i}}{g_{\text{spec}}}\right)\left(k_i - v\frac{g_{z_i}}{g_{\text{spec}}}\right) - \frac{2uv}{g_{\text{spec}}^2} \tag{5.38}$$

the set of equations necessary for the first set of unit cell parameters is expressed in matrix form as

$$\begin{pmatrix} f_{11}(h_1) & f_{12}(k_1) & f_{13}(h_1,k_1) \\ f_{21}(h_2) & f_{22}(k_2) & f_{23}(h_2,k_2) \\ f_{31}(h_3) & f_{32}(k_3) & f_{33}(h_3,k_3) \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{5.39}$$

The partial solutions (u,v,a,b,γ) can subsequently be computed using the substitutions (5.20) as well as the relation

$$z_a^2 z_b^2 \frac{\sin^2\gamma}{g_{\text{spec}}^2} = \frac{XY - Z^2}{g_{\text{spec}}^2 - (u^2X + v^2Y - 2uvZ)}. \tag{5.40}$$

In a subsequent part, the sets referred to as partial solutions (w,α,β,c) can be determined by evaluation of equation (5.31). The calculation is based on the same matrix system as defined in equation (5.13) for the special case. However, an overdetermined LSE (incorporating the specular scan by $q_{\text{spec},T}$ from equation (5.27)) is used to derive w, α, β and c. The same substitutions as stated in (5.11) are applied for κ, λ and ρ. With the additional abbreviation for the right-hand side terms of equation (5.31)

$$\frac{g_{z_i}g_{\text{spec}} - h_i u z_a^2 - k_i v z_b^2 - (h_i v + k_i u) z_a z_b \cos\gamma}{(g_{\text{spec}}^2 - u^2 z_a^2 + v^2 z_b^2 + 2uv z_a z_b \cos\gamma)^{1/2}} := q_{z,T_i} \tag{5.41}$$

the set of equations can be explicitly written as

$$\begin{pmatrix} u & v & w \\ h_1 & k_1 & l_1 \\ h_2 & k_2 & l_2 \\ h_3 & k_3 & l_3 \end{pmatrix} \begin{pmatrix} \kappa \\ \lambda \\ \rho \end{pmatrix} = \begin{pmatrix} q_{\text{spec},T} \\ q_{z,T_1} \\ q_{z,T_2} \\ q_{z,T_3} \end{pmatrix}. \tag{5.42}$$

With this, the mathematical description of the indexing procedure is complete and ready for programmatic implementation. It should be noted that this order and use of the equations is only one possibility out of symmetry-equivalent sets of equations. By using the symmetry relations defined in Table 5.2, the algorithm is applicable also in different orders and step-wise derivation of (u,v,a,b,γ)+(w,α,β,c), (u,w,a,c,β)+(v,α,γ,b) as well as (v,w,b,c,α)+(u,β,γ,a) is possible.

Table 5.2: Summary of relations for the length g_{xyz}, the in-plane part g_{xy} and the out-of-plane part g_z of reciprocal space vectors.

$$g_{xyz}^2 = g_x^2 + g_y^2 + g_z^2 = h^2a^{*2} + k^2b^{*2} + l^2c^{*2} + 2hka^*b^*\cos\gamma^* + 2hla^*c^*\cos\beta^* + 2klb^*c^*\cos\alpha^*$$

$$\begin{aligned} g_z g_{\text{spec}} = {} & hua^{*2} + kvb^{*2} + lwc^{*2} + (hv+ku)a^*b^*\cos\gamma^* \\ & + (hw+lu)a^*c^*\cos\beta^* + (kw+lv)b^*c^*\cos\beta^* \end{aligned}$$

$$g_{\text{spec}} = (u^2a^{*2} + v^2b^{*2} + w^2c^{*2} + 2uva^*b^*\cos\gamma + 2uwa^*c^*\cos\beta^* + 2vwb^*c^*\cos\alpha^*)^{1/2}$$

$$\begin{aligned} g_{xy}^2 = {} & \left(\frac{2\pi}{a\sin\gamma}\right)^2\left[\left(h-u\frac{g_z}{g_{\text{spec}}}\right)^2 + \left(u\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] + \left(\frac{2\pi}{b\sin\gamma}\right)^2\left[\left(k-v\frac{g_z}{g_{\text{spec}}}\right)^2 + \left(v\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] \\ & -2\frac{2\pi}{a\sin\gamma}\frac{2\pi}{b\sin\gamma}\cos\gamma\left[\left(h-u\frac{g_z}{g_{\text{spec}}}\right) + \left(k-v\frac{g_z}{g_{\text{spec}}}\right) + \right)uv\frac{g_{xy}}{g_{\text{spec}}}\Bigg)^2\Bigg] \\ & -\frac{(hv-ku)^2}{q_{\text{spec}}^2}\left(\frac{2\pi}{a\sin\gamma}\right)^2\left(\frac{2\pi}{a\sin\gamma}\right)^2\sin^2\gamma \end{aligned}$$

$$\begin{aligned} g_{xy}^2 = {} & \left(\frac{2\pi}{a\sin\beta}\right)^2\left[\left(h-u\frac{g_z}{g_{\text{spec}}}\right)^2 + \left(u\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] + \left(\frac{2\pi}{c\sin\beta}\right)^2\left[\left(l-w\frac{g_z}{g_{\text{spec}}}\right)^2 + \left(w\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] \\ & -2\frac{2\pi}{a\sin\beta}\frac{2\pi}{c\sin\beta}\cos\beta\left[\left(h-u\frac{g_z}{g_{\text{spec}}}\right) + \left(l-w\frac{g_z}{g_{\text{spec}}}\right) + \left(uw\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] \\ & -\frac{(hw-lu)^2}{q_{\text{spec}}^2}\left(\frac{2\pi}{a\sin\beta}\right)^2\left(\frac{2\pi}{c\sin\beta}\right)^2\sin^2\beta \end{aligned}$$

$$\begin{aligned} g_{xy}^2 = {} & \left(\frac{2\pi}{b\sin\alpha}\right)^2\left[\left(k-v\frac{g_z}{g_{\text{spec}}}\right)^2 + \left(v\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] + \left(\frac{2\pi}{c\sin\alpha}\right)^2\left[\left(l-w\frac{g_z}{g_{\text{spec}}}\right)^2 + \left(w\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] \\ & -2\frac{2\pi}{b\sin\alpha}\frac{2\pi}{c\sin\alpha}\cos\alpha\left[\left(k-v\frac{g_z}{g_{\text{spec}}}\right) + \left(l-w\frac{g_z}{g_{\text{spec}}}\right) + \left(vw\frac{g_{xy}}{g_{\text{spec}}}\right)^2\right] \\ & -\frac{(kw-lv)^2}{q_{\text{spec}}^2}\left(\frac{2\pi}{b\sin\alpha}\right)^2\left(\frac{2\pi}{c\sin\alpha}\right)^2\sin^2\alpha \end{aligned}$$

5.3 Numerical Optimization

One part of the postprocessing sub-routine is the numerical optimization of the unit cell parameters. The numerical error with respect to the total length q_{xyz} and the out-of-plane component q_z can be reduced if the Laue indices as well as the (reciprocal) lattice constants are known. The algorithm is valid for small deviations and based on first-order correction [12]. The numerical optimization can be achieved by minimizing the quadratic error function

$$E^n(a^*,b^*,c^*,\alpha^*,\beta^*,\gamma^*) = \left[\left(E^n_{xyz}(a^*,b^*,c^*,\alpha^*,\beta^*,\gamma^*)\right)^2 + \left(E^n_z(a^*,b^*,c^*,\alpha^*,\beta^*,\gamma^*)\right)^2\right]^{1/2}. \tag{5.43}$$

For a total number of n diffraction peaks, the individual terms can be explicitly written as

$$E^n_{xyz} = \left[\frac{1}{n}\sum_{i=1}^{n}\left(g_{xyz_i} + \frac{\delta g_{xyz_i}}{\delta a^*}\varepsilon_{a^*} + \frac{\delta g_{xyz_i}}{\delta b^*}\varepsilon_{b^*} + \frac{\delta g_{xyz_i}}{\delta c^*}\varepsilon_{c^*} + \frac{\delta g_{xyz_i}}{\delta \alpha^*}\varepsilon_{\alpha^*} + \frac{\delta g_{xyz_i}}{\delta \beta^*}\varepsilon_{\beta^*} + \frac{\delta g_{xyz_i}}{\delta \gamma^*}\varepsilon_{\gamma^*} - q_{xyz_i}\right)^2\right]^{1/2} \tag{5.44}$$

and

$$E^n_z = \left[\frac{1}{n}\sum_{i=1}^{n}\left(g_{z_i} + \frac{\delta g_{z_i}}{\delta a^*}\varepsilon_{a^*} + \frac{\delta g_{z_i}}{\delta b^*}\varepsilon_{b^*} + \frac{\delta g_{z_i}}{\delta c^*}\varepsilon_{c^*} + \frac{\delta g_{z_i}}{\delta \alpha^*}\varepsilon_{\alpha^*} + \frac{\delta g_{z_i}}{\delta \beta^*}\varepsilon_{\beta^*} + \frac{\delta g_{z_i}}{\delta \gamma^*}\varepsilon_{\gamma^*} - q_{z_i}\right)^2\right]^{1/2}. \tag{5.45}$$

The i-th experimental peak position is considered by (q_{xyz_i}, q_{z_i}) together with the according calculated peak position (g_{xyz_i}, g_{z_i}). The correction terms with respect to the reciprocal lattice constants are denoted by ε_{a^*}, ε_{b^*}, ε_{c^*}, ε_{α^*}, ε_{β^*} and ε_{γ^*}. The quadratic error function $E^n(a^*,b^*,c^*,\alpha^*,\beta^*,\gamma^*)$ is minimized if the partial derivative vanishes for every component:

$$\frac{\delta E^n}{\delta a^*} = \frac{\delta E^n}{\delta b^*} = \frac{\delta E^n}{\delta c^*} = \frac{\delta E^n}{\delta \alpha^*} = \frac{\delta E^n}{\delta \beta^*} = \frac{\delta E^n}{\delta \gamma^*} = 0. \tag{5.46}$$

This condition leads to the following relation:

$$\sum_{i=1}^{N} \begin{pmatrix} \mathbf{f}_{a^*,i}^2 & \mathbf{f}_{a^*,i}\mathbf{f}_{b^*,i} & \mathbf{f}_{a^*,i}f_{c^*,i} & \mathbf{f}_{a^*,i}\mathbf{f}_{\alpha^*,i} & \mathbf{f}_{a^*,i}\mathbf{f}_{\beta^*,i} & \mathbf{f}_{a^*,i}\mathbf{f}_{\gamma^*,i} \\ \mathbf{f}_{a^*,i}\mathbf{f}_{b^*,i} & \mathbf{f}_{b^*,i}^2 & \mathbf{f}_{b^*,i}\mathbf{f}_{c^*,i} & \mathbf{f}_{b^*,i}\mathbf{f}_{\alpha^*,i} & \mathbf{f}_{b^*,i}\mathbf{f}_{\beta^*,i} & \mathbf{f}_{a^*,i}\mathbf{f}_{\gamma^*,i} \\ \mathbf{f}_{a^*,i}\mathbf{f}_{c^*,i} & \mathbf{f}_{b^*,i}\mathbf{f}_{c^*,i} & \mathbf{f}_{c^*,i}^2 & \mathbf{f}_{\alpha^*,i}\mathbf{f}_{\gamma^*,i} & \mathbf{f}_{c^*,i}\mathbf{f}_{\beta^*,i} & \mathbf{f}_{c^*,i}\mathbf{f}_{\gamma^*,i} \\ \mathbf{f}_{a^*,i}\mathbf{f}_{\alpha^*,i} & \mathbf{f}_{b^*,i}\mathbf{f}_{\alpha^*,i} & \mathbf{f}_{c^*,i}\mathbf{f}_{\alpha^*,i} & \mathbf{f}_{\alpha^*,i}^2 & \mathbf{f}_{\alpha^*,i}\mathbf{f}_{\beta^*,i} & \mathbf{f}_{\alpha^*,i}f_{\gamma^*,i} \\ \mathbf{f}_{a^*,i}\mathbf{f}_{\beta^*,i} & \mathbf{f}_{b^*,i}\mathbf{f}_{\beta^*,i} & \mathbf{f}_{c^*,i}\mathbf{f}_{\beta^*,i}^2 & \mathbf{f}_{\alpha^*,i}\mathbf{f}_{\beta^*,i} & \mathbf{f}_{\beta^*,i}^2 & \mathbf{f}_{\beta^*,i}\mathbf{f}_{\gamma^*,i} \\ \mathbf{f}_{a^*,i}\mathbf{f}_{\gamma^*,i} & \mathbf{f}_{b^*,i}\mathbf{f}_{\gamma^*,i} & \mathbf{f}_{c^*,i}\mathbf{f}_{\gamma^*,i} & \mathbf{f}_{\alpha^*,i}\mathbf{f}_{\gamma^*,i} & \mathbf{f}_{\beta^*,i}\mathbf{f}_{\gamma^*,i} & \mathbf{f}_{\gamma^*,i}^2 \end{pmatrix} \begin{pmatrix} \varepsilon_a^* \\ \varepsilon_b^* \\ \varepsilon_c^* \\ \varepsilon_\alpha^* \\ \varepsilon_\beta^* \\ \varepsilon_\gamma^* \end{pmatrix} \tag{5.47}$$

$$= \sum_{i=1}^{N} \begin{pmatrix} \mathbf{f}_{a^*,i} \\ \mathbf{f}_{b^*,i} \\ \mathbf{f}_{c^*,i} \\ \mathbf{f}_{\alpha^*,i} \\ \mathbf{f}_{\beta^*,i} \\ \mathbf{f}_{\gamma^*,i} \end{pmatrix} \begin{pmatrix} q_{xyz_i} - g_{xyz_i} \\ q_{z_i} - g_{z_i} \end{pmatrix}$$

where the individual matrix entries are the inner products of the vectors with partial derivatives with respect to the edge lengths written as

$$\mathbf{f}_{a^*,i} = \begin{pmatrix} \frac{\delta g_{xyz,i}}{\delta a^*} \\ \frac{\delta g_{z,i}}{\delta a^*} \end{pmatrix}^{\mathrm{T}}, \qquad \mathbf{f}_{b^*,i} = \begin{pmatrix} \frac{\delta g_{xyz,i}}{\delta b^*} \\ \frac{\delta g_{z,i}}{\delta b^*} \end{pmatrix}^{\mathrm{T}}, \qquad \mathbf{f}_{c^*,i} = \begin{pmatrix} \frac{\delta g_{xyz,i}}{\delta c^*} \\ \frac{\delta g_{z,i}}{\delta c^*} \end{pmatrix}^{\mathrm{T}} \tag{5.48}$$

and with respect to the three angles described by

$$\mathbf{f}_{\alpha^*,i} = \begin{pmatrix} \frac{\delta g_{xyz,i}}{\delta \alpha^*} \\ \frac{\delta g_{z,i}}{\delta \alpha^*} \end{pmatrix}^{\mathrm{T}}, \qquad \mathbf{f}_{\beta^*,i} = \begin{pmatrix} \frac{\delta g_{xyz,i}}{\delta \beta^*} \\ \frac{\delta g_{z,i}}{\delta \beta^*} \end{pmatrix}^{\mathrm{T}}, \qquad \mathbf{f}_{\gamma^*,i} = \begin{pmatrix} \frac{\delta g_{xyz,i}}{\delta \gamma^*} \\ \frac{\delta g_{z,i}}{\delta \gamma^*} \end{pmatrix}^{\mathrm{T}}. \tag{5.49}$$

After successful assignment of the Laue indices and determination of the unit cell constants, the matrix stated in (5.47) can be built with the computed derivatives. It is possible to derive analytical expressions for the partial derivatives stated in (5.48) and (5.49). For this work however, a numerical solution is implemented. The evaluation of the matrix yields the numerical corrections in $(a^*, b^*, c^*, \alpha^*, \beta^*, \gamma^*)$ and the real space lattice parameters can subsequently be derived by the relations stated in Table 5.1.

Chapter 6

Indexing Routine

The formalism for indexing two-dimensional GIXD data is given in the previous chapter. Here, the programmatic implementation of this formalism is discussed. The Indexing Routine is visualized in terms of flow charts (or flow diagrams). Subsequently, the individual building blocks of the charts are explained in more detail using the accompanying balloons. The routine is written in MATLAB, but for the sake of clarity no code is explicitly printed is this chapter. A copy of the total program source code as well as a collection of the functions written for this routine can be found in the appendices. If necessary, used build-in functions are discussed separately on the basis of the MathWorks ® MATLAB Documentation [49, 50].

The icons and shapes used for the flow diagrams follow certain guidelines. Box titles and statements within the icons used in the flow charts are written in italic font. Different shapes and colours are used to describe distinct tasks. Pale blue shading is used to mark *Start* and *End* of the overall program. The same colour in combination with rhombus-shaped icons is used to define decisions. Data input is necessary only once in the whole routine and it is marked with a grey parallelogram. In Figure 6.1, the data input icon is located at position (1). From a programmatic point of view, it is a part of the sub-process *Initialization*, marked with (2). Processes or sub-programs (sub-routines) as this one are indicated by green boxes where vertical lines on the outer edges refer to a parallel process (*cf.* sub-chapter 6.1). If the input data as well as the settings are checked and released, the indexing sub-routines are accessed. The test is indicated by the decision box at location (3).

Every of the four sub-routines is described within an individual flow chart. Results obtained from the program parts *Routine Part 1* at position (4) and from *Routine Part 2* at position (7) are marked with frames shaped like reels. The first set is obtained by running

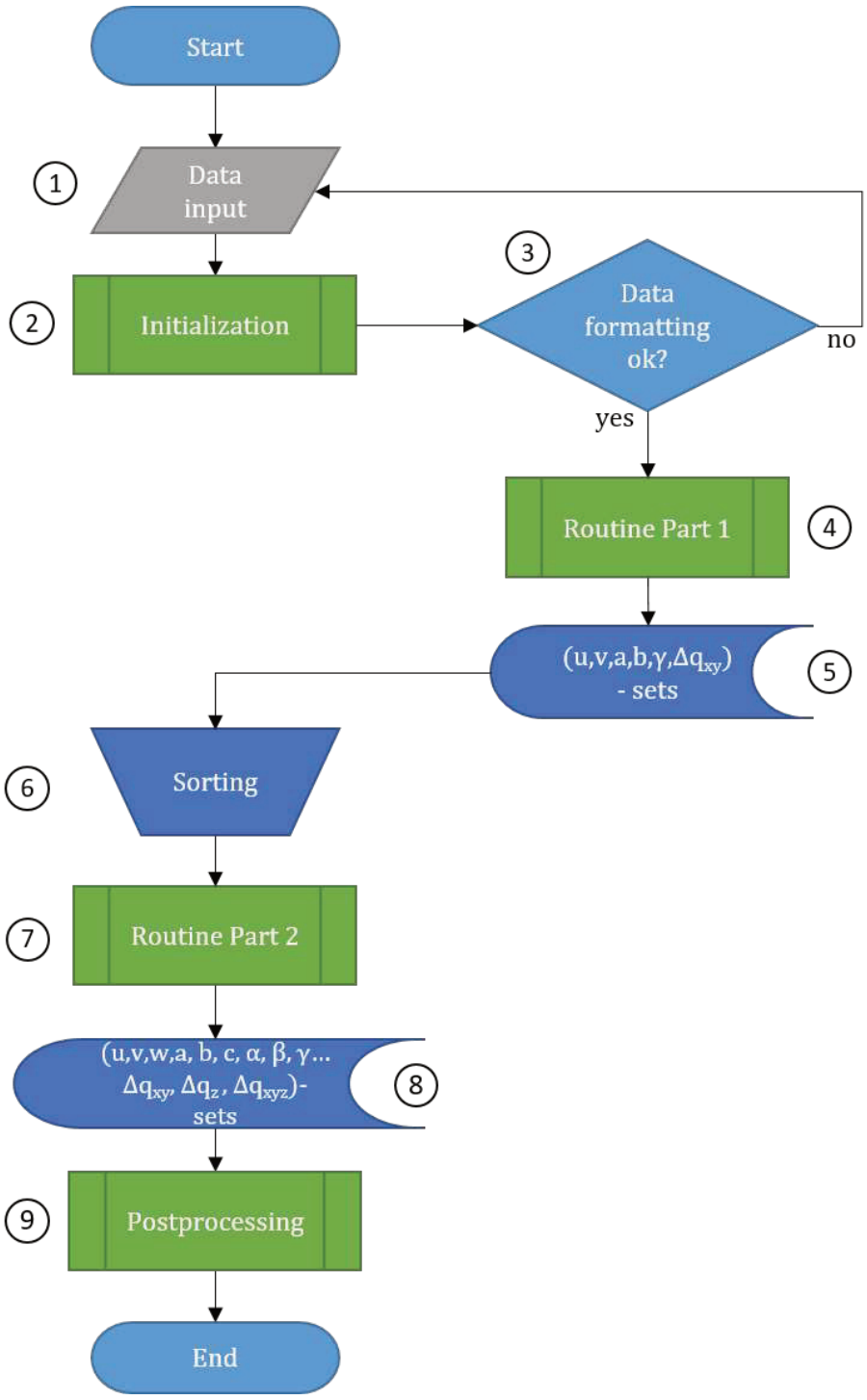


Figure 6.1: Flow diagram describing the overall workflow of the Indexing Routine.

through *Routine Part 1* with the results stated at position (5). In a next step, the partial unit cell results are sorted. The graphical user interface (which is introduced in Chapter 7) allows to apply two ways of sorting before continuing to the *Routine Part 2* in (7). One button applies sorting of the preliminary sets of $(u, v, a, b, \gamma, \Delta q_{xy})$ by the total error

in the components q_{xy}. The other button allows sorting by the area of the associated parallelogram. This corresponds to sorting by the smallest cell lengths. The area is computed with

$$A_{||} = a\,b\,\sin\gamma \tag{6.1}$$

and the parameters are subsequently arranged in ascending order beginning with the smallest. The process *Sorting* is indicated by a funnel-like icon and marked with (6). Within the parallel computation of *Routine Part 2*, sets of
$(u, v, w, a, b, c, \alpha, \beta, \gamma, \Delta q_{xy}, \Delta q_z, \Delta q_{xyz})$ are determined. The resulting sets are collected in (8) before being evaluated with the last sub-routine marked as *Postprocessing* at position (9). The output of the routine after the postprocessing are sets of unit cell solutions and associated indexed diffraction peaks. Figure 6.1 shows an overview of the workflow containing the four sub-processes. These processes are implemented in a parallel way and described separately.

6.1 Parallel Computing

Processing time and memory management are two important characteristics of an implemented routine. Data-intensive problems can slow down the overall process and unnecessarily occupy the random access memory. To some extent, this problem can be avoided by implementing parallel programming methods or *parallel computing*. The methods allow to divide tasks by outsourcing calculations to so-called *workers*. In the easiest case, these workers are the cores of a multi-core central processing unit (CPU). Depending on the system features, the numerical problems are separated and allocated to the individual cores. This is demonstrated in Figure 6.2. For mathematically unsophisticated tasks in combination with a low number of loop iterations (e.g. adding numbers from 1 to 1000), the advantages of the parallel computation are insignificant. However, for more elaborated processes such as the determination of matrix eigenvalues, parallel loops can considerably enhance the performance in terms of computation time. Another example for this is the here used application of the symbolic matrix left division for solving linear systems of equations. Many different sub-programs are necessary for such a calculation and parallel structures can be superior to a serial procedure, especially if the number of loop iterations increases.

A common application in this work is the usage of a so-called parfor-loop. The loop executes general for-loop iterations in parallel on workers specified in a parallel pool. The parallel pool is the maximum number of workers to outsource the tasks. A pool is

started automatically if a parallel language feature such as `parfor` is used in the code. Manual startup of a parallel pool can be done by entering `parpool(m)` into the command window. The input `m` specifies the number of workers. Such a pool has a certain lifetime. After this time, a new one has to be initiated either manually or automatically by starting the program. By default, the cluster of the pool is the local computer and the lifetime is 30 min. The number of workers can be specified explicitly, if needed. In the code here, the maximum number of workers is always adapted to the number of physical cores and therefore not further specified in the code. Note that the startup of a parallel pool takes some time as the code has to be shifted to the different workers. The second and every further run of the loop (and therefore the overall routine) should then show reduced runtime.

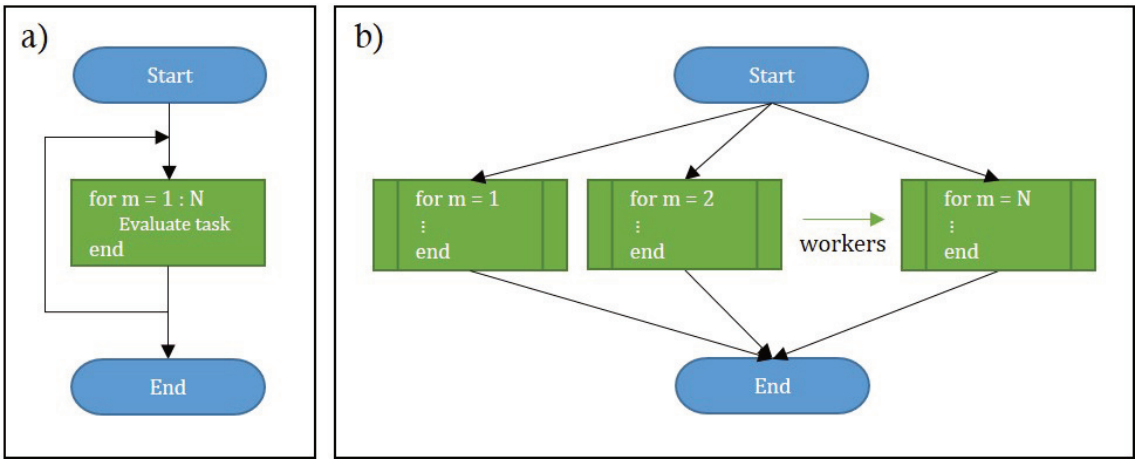


Figure 6.2: Working principle of (a) serial computation and (b) parallel computation of a random for-loop with loop variable m. The loop is finished as soon as $n = N$.

MATLAB provides application of parallel methods with the *Parallel Computing Toolbox*™. The here proposed algorithm relies on such methods and the toolbox is therefore a prerequisite to run the Indexing Routine. Especially the parallel execution of for-loops reduces the computation time of the sub-programs and with this, the overall processing time. As mentioned above, green boxes with parallel lines at the outer edges are implemented in a parallel way. The first sub-routine is the *Initialization*.

6.2 Initialization

The aim of the *Initialization* building block is to verify the uploaded data for further processing and to create general program variables and arrays. An overview of the tasks and the according workflow is shown in Figure 6.3.

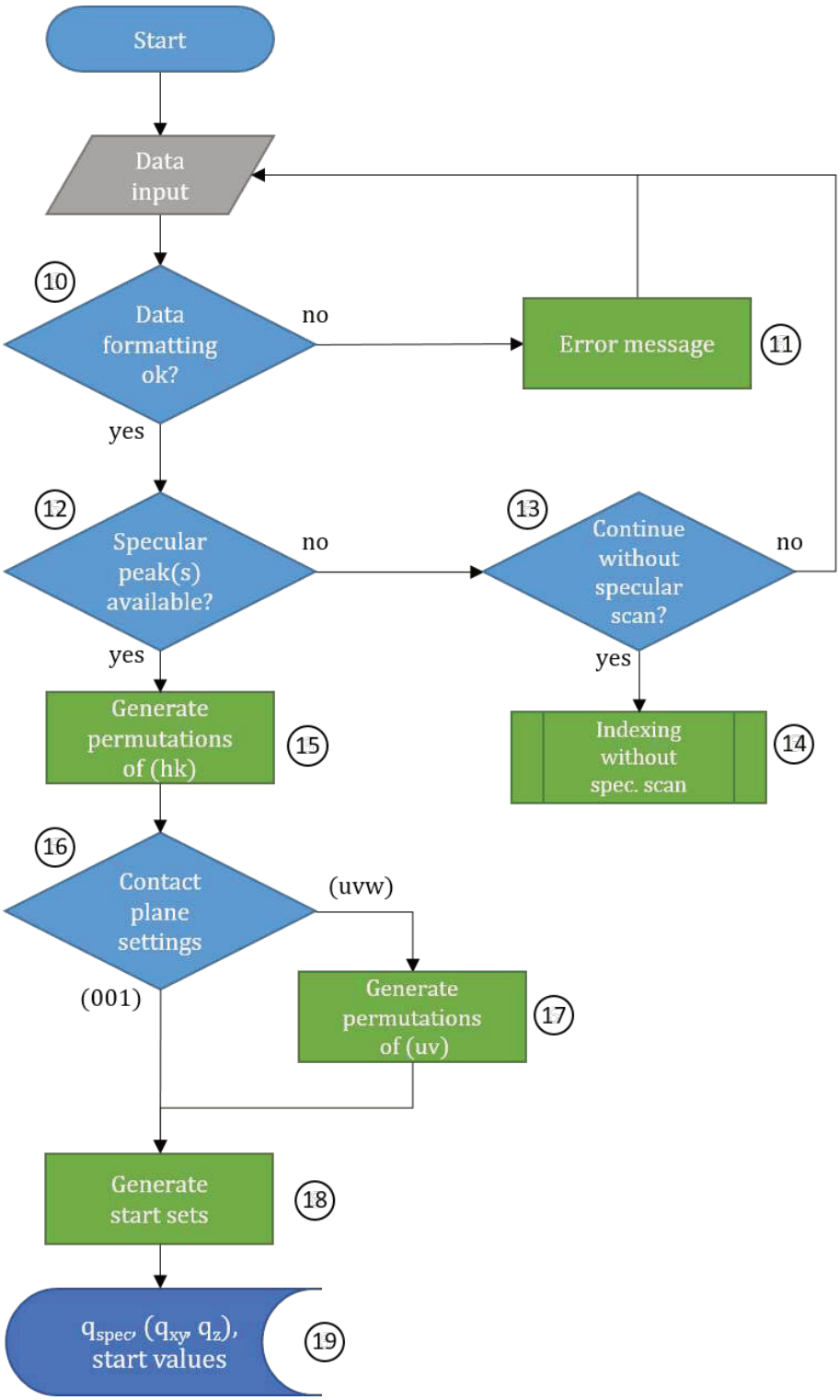


Figure 6.3: Flow diagram to visualize the building block *Initialization*. Data formatting and specular peaks are verified and general variables such as the numeric matrices containing the Laue index permutations are generated in this block.

The data have to be provided in a formatted .xls (or .xlsx) file. The first column contains the numerical values of q_{xy} and the second column the corresponding values for q_z. Only the first and the second column are considered from the routine. The specular scan has to be included in the uploaded file. This is the entry with $q_{xy_i} = 0$ in the first column. The uploaded file should only contain numeric floating-point numbers with a dot as decimal separation. Any other data formatting will lead to an internal error in the routine, as indicated by (11) in 6.3. The data are sorted and counted to create corresponding numeric arrays with appropriate sizes for further processing. If the formatting is accepted and at least one specular peak is contained, the first variables are generated. This is indicated by (10) and (12). The balloon (13) marks a decision point. If no specular peak is available, the program can be restarted with an altered data file. The second possibility, marked by (14), includes an indexing algorithm without the need of a specular peak. The external MATLAB code can be called with a separated button at the graphical user interface. The determination of the unit cell parameters upon indexing with q_{spec} requires less iterative steps and is therefore less time-consuming. It is advised to include the specular information when working with this app. In a next step, indicated by (15), the numeric arrays containing the permutations of the Laue indices h and k are generated. This is realized with the function `LPermutation(hkvalues,n)`. The source code of the main program as well as for all written MATLAB-functions is included in Appendix B. The function to create the matrices $\mathbf{K}$ takes two input parameters. The first, `hkvalues`, is a vector defining the v elements available for permutation. The digits are specified by the input variable `n`. For tuples, `n` is set to the value of 2 and for triplets it is set to 3. The generation of these permutation matrices is done by iterative filling of empty default matrices of the size $(v^n \times n)$, what corresponds to application of the known combinatorial problem 'permutations with repetition'. For example, all permutations for h and k in a range of -1 and 1 can be explicitly read out with

$$\mathbf{K} = \begin{pmatrix} -1 & -1 \\ 0 & -1 \\ 1 & -1 \\ -1 & 0 \\ 0 & 0 \\ 1 & 0 \\ -1 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \tag{6.2}$$

where every row of the permutation matrix represents one tuple (hk). In general, v^n combinations have to be considered. The regularly mentioned term 'systematic variation of Laue indices' corresponds effectively to scanning through the v^n rows of the permutation matrices with subsequent evaluation and comparison of the associated $g_{xy}(hk)$. The further process requires the same permutation for the Miller indices (uv) if a general contact plane is chosen at the node (16). If only the special case is considered, the permutation matrix reduces to a vector of the form $uv = (0,0)$.

To generate potential solutions for (u,v,a,b,γ), linearly independent combinations of Bragg peaks have to be considered for the LSEs stated in the expressions (5.10) and (5.39), respectively. A possible approach to check this (in)dependency could be implemented as follows: The quotient of two different points q_{xy_1} and q_{xy_2} is computed where $q_{xy_1} > q_{xy_2}$. If the absolute difference between the quotient and its rounded integer neighbour is below a certain numerical limit (e.g. $\varepsilon_c = 0.1$ Å^{-1}), the two values are considered as linearly dependent. A pair of q_{xy_1} and q_{xy_2} where

$$\sqrt{\left(\frac{q_{xy_1}}{q_{xy_2}} - \left|\frac{q_{xy_1}}{q_{xy_2}}\right|\right)^2} > \varepsilon_c \tag{6.3}$$

is considered to be numerically independent and a triplet of such independent values is here referred to as start set. This has to be tested for every possible combination out of a pick of $(q_{xy_1}, q_{xy_2}, q_{xy_3})$.

Such an algorithm was implemented in an early stage for the special case with a (001) contact plane and should be considered if memory occupation is a problem and/or parallel computation is not possible at the used machine. In reality, the routine switches through a specified number of q_{xy_i}-values (with the variable `lines_to_permute`), permutes the lines in every possible way and thus creates a series of different start sets. These sets are chosen independently from any user-defined numerical limit. If there is at least one possible combination of three q_{xy_i}-values within the lowest and the `lines_to_permute`-th entry, it can be found with this algorithm. This is especially true, as numerically similar values are traced and combined with the MATLAB build-in function `uniquetol()`. This function allows to summarize numerical values within a specified range. The introduced algorithm has been tested by creation of ideal diffraction patterns and successive parameter control and adjustment. The lowest values of q_{xy_i} should be considered first, as the range of possible Laue indices is narrowed. The according MATLAB function can be found in the Appendix B under the name of `function_INITIALIZE_GUI.m`. The outputs of the *Initialization* building block are the generated permutation matrices and sets of start values. If the formatting is correct, the input data as well as the specular scan

are submitted to the next program block.

6.3 Indexing Routine Part I

In this part of the program, potential solutions of (u,v,a,b,γ) are generated. Subsequently, the diffraction data are indexed via assignment of two Laue indices to each in-plane component q_{xy_i} of the scattering vectors and evaluated by their mean deviation. The outcome reduces to (a,b,γ) for the (001) case. To obtain the partial unit cell solutions, linear systems of equations have to be formed and solved. These staps are marked with (20) to (22) in Figure 6.4 as part of the overall work flow of *Routine Part 1*. The systems have the form of equation (5.10) for the special case. The right-hand side containing the input data is derived within the *Initialization* process. For the special case, the matrices can be formed by using again the permutation function `LPermutation(hkvalues,n)`. To generate every possible solution in $(X,Y,Z)^{\mathrm{T}}$ and (a,b,γ), respectively, every possible form of the coefficient matrix has to be generated and solved. This can be achieved by creating every possible permutation for six digits $(h_1,k_1,...h_3,k_3)$ out of a set with a lower and upper limit for the Laue indices. This limit is specified by the program input *Max. hk for LSE* on the GUI. As the start sets are usually generated from the lowest q_{xy}-values, an upper limit for the Laue indices of 3 to 5 has shown to be sufficient. Upon running the routine, lower values (e.g. *Max. hk for LSE* = 3) should be considered first, as the number of matrix equations to be solved grows again with the power of the digits.
Every homogeneous system of linear equations with a determinant unequal to zero has a unique solution. The system contains at least two equations that are not linearly independent if the determinant is zero. For every matrix with a non-zero determinant, the solution is determined by the MATLAB build-in function `mldivide(A,b)` which solves the system $\mathbf{Ax} = \mathbf{b}$ for x. By using the substitutions defined in (5.8), the sets of $(X,Y,Z)^{\mathrm{T}}$ can be converted to sets of (a,b,γ). The situation becomes more complex for the general case, as the matrix entries do not only depend on the Laue indices h and k. Thus, mere permutation of integer numbers is no more sufficient. The matrix stated in equation (5.39) has to be constructed in this case whereby the individual entries are defined by the equations (5.36), (5.37) and (5.38). The generation of the matrices is realized by the function `function_NEWMATRIXFILLER.m`. The calculation of $(X,Y,Z)^{\mathrm{T}}$ and subsequently the lattice constants is done using the function `function_XYZ.m` within the script of `function_ABG_RESTRICTED.m`. These steps are indicated by (22) and (23).

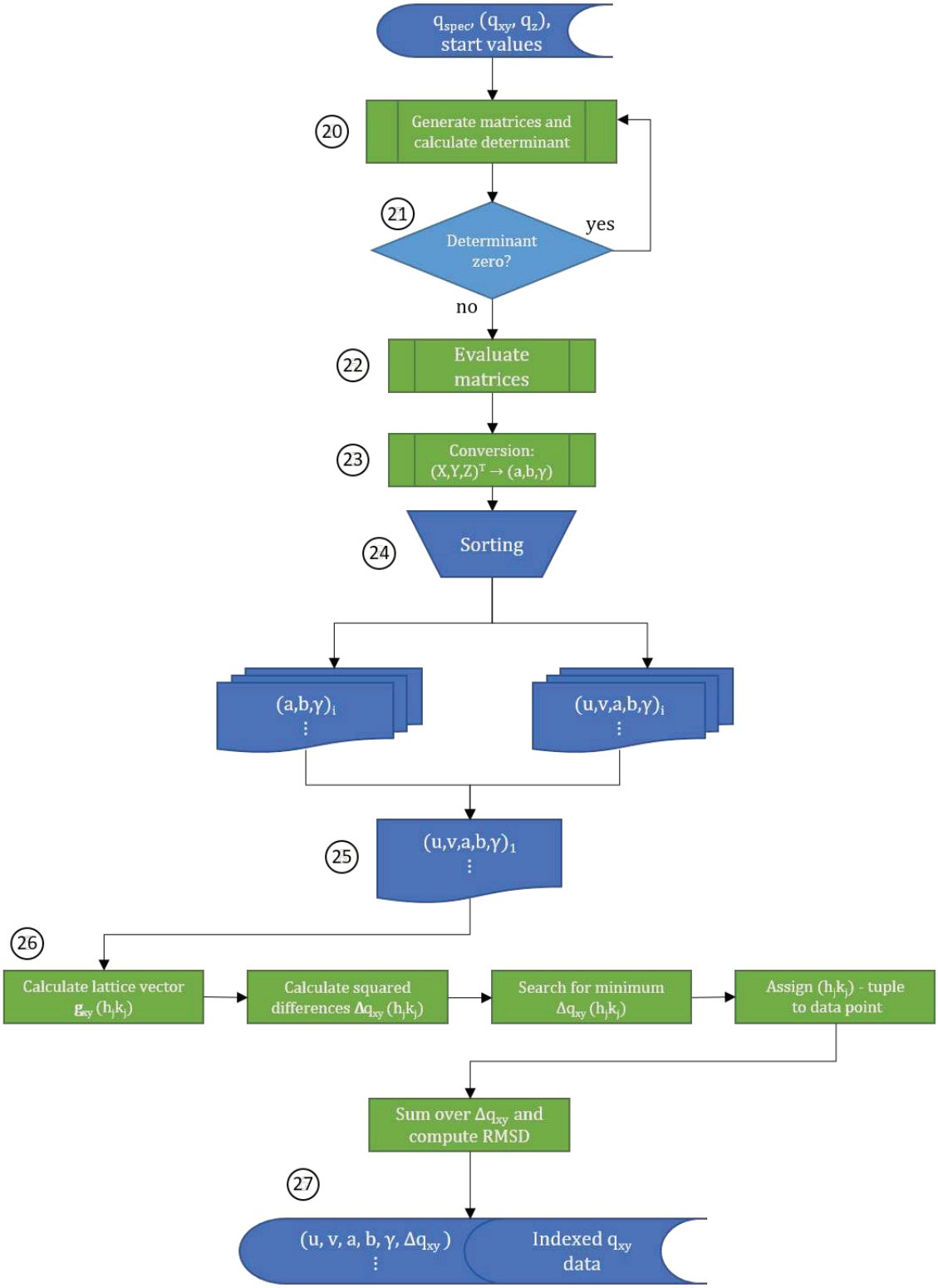


Figure 6.4: Flow chart demonstrating the programmatic approach for the *Indexing Routine 1*.

The start sets (q_{xy_i}, q_{z_i}) are taken from the *Initialization* part, whereby the lowest values in $q_{xyz} = (q_{xy}^2 + q_z^2)^{1/2}$ are considered at first. The overall *Routine Part 1* is contained in

the attached MATLAB function with the name
`function_PART_ONE_PARALLEL_GUI_RESTRICTIONS.m`. The ensuing sorting step of the sets is indicated by (24) in the flow chart 6.4 above. Non-real or negative numerical values and values which do not lie within the restrictions are not further considered. As demonstrated later in Chapter 7, restrictions can be applied to the lattice constants. Such boundaries can help thereby to limit the number of possible results. Niggli's criteria for a reduced cell (*cf.*, Chapter 3, Table 3.1) can be applied at this stage as a further filtering tool. Condition (iv) and the general consequence for the angles where $60° \leqslant \alpha, \beta, \gamma \leqslant 120°$ restrict the numerical output and only the partial solutions according to recognised crystallographic criteria remain. The *Sorting* block is realized in the functions `function_CALC_ABGfromXYZ_RESTRICTED_v7` and `function_ABGNIGGLI.m`. After that sequence, the potential sets of (u, v, a, b, γ) are ready to be indexed. The procedure is applied iteratively to the sets, as indicated with (25) and (26). Using the graphical user interface, the maximal number for the Laue index permutation can be specified with the program parameter *Max. hk for indexing qxy*. This value should be distinguished from the maximum number for solving the LSEs, as not only the lowest values in q_{xy} are considered now, but all the diffraction peaks. For the higher values q_{xy}, the maximum number should increase. Testing the routine has shown that *Max. hk for indexing qxy* = 6 is a good starting point.

For every row ν out of the permutation matrix, the value of the component $g_{xy}(h_\nu, k_\nu)$ is calculated according to equation (5.9). Subsequently, the squared differences

$$\Delta_{q_{xy,n}}(h_\nu, k_\nu) = [g_{xy}(h_\nu, k_\nu) - q_{xy,n}]^2 \tag{6.4}$$

are calculated and the Laue index tuple (h_ν, k_ν) is assigned to the q_{xy}-value where the difference $\Delta_{q_{xy,n}}(h_\nu, k_\nu)$ is lowest. In reality, not only the tuple with the lowest deviation is stored, but a set of four possibilities is shifted to further processing in *Routine Part 2*. This is because different combinations can lead to the same numerical values in g_{xy}. Only the determination of all three Laue indices allows to derive an unambiguous value. After all the minimal differences $(\Delta q_{xy_1}, \Delta q_{xy_2}, ...\Delta q_{xy_N})$ are found for each of the N data points, the root mean square deviation for one set of (a, b, γ) can be computed by

$$RMSD_{q_{xy}}(a, b, \gamma) = \sqrt{\frac{1}{N}\sum_{n=1}^{N} \Delta q_{xy_n}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (g_{xy_n} - q_{xy_n})^2}. \tag{6.5}$$

This is the quantity used to rate the 'quality' of the solution. The assignment of the Laue indices takes place in `function_NEWINDEXING_26.m` and the calculation of the root mean square deviations is done with `function_NEWRMSD_GXY.m`.

The indexing approach for the general case is the same as for a (001) contact plane. For calculation of the scattering vector components, however, equation (5.30) as well as the one from Table 5.2 have to be considered. An additional loop is also required to take the variation of the Miller indices u and v into account. The routine is applied to all possible sets of (u,v,a,b,γ) and results in two outputs: (i) a list of diffraction peaks with assigned sets of Laue tuples and (ii) an associated list of partial unit cell solutions. This output, indicated by (27) in Figure 6.4, is the input for the *Routine Part 2*.

6.4 Indexing Routine Part II

The determination of the three remaining lattice constants α, β and c takes plane within the block *Routine Part 2*. The individual sub-processes are demonstrated in Figure 6.5. The programmatic approach is similar to the first part and the special case is treated in the same way and in the same code as the general way in `function_PART_TWO_v12.m`. Solutions to linear systems of equations are generated what results in potential sets of $(u,v,w,a,b,c,\alpha,\beta,\gamma)$. The sets are rated numerically as well as in accordance with Niggli's criteria for a reduced cell. Only sets which pass these sorting steps are considered for the subsequent indexing procedure. The mean deviations in q_z and, in particular, in q_{xyz} are evaluated and used to sort and rate the output.
In this sub-routine, the overdetermined systems stated in equation (5.42) are evaluated. For the general case, the systems can be explicitly written as

$$\begin{pmatrix} u & v & w \\ h_1 & k_1 & l_1 \\ h_2 & k_2 & l_2 \\ h_3 & k_3 & l_3 \end{pmatrix} \begin{pmatrix} \kappa \\ \lambda \\ \rho \end{pmatrix} = \begin{pmatrix} q_{\mathrm{spec},T} \\ q_{z,T_1} \\ q_{z,T_2} \\ q_{z,T_3} \end{pmatrix}. \tag{6.6}$$

where the vector $(\kappa,\ \lambda,\ \rho)^{\mathrm{T}}$ contains the remaining unit cell parameters. In a first step, labelled as (28), start values for q_z have to be determined just as for *Routine Part 1*.The numerical values are summarized using the build-in function `uniquetol()`. In the general case, the vectors on the right hand side of equation (6.6) contain not only the input q_z, but more complex terms. The information from the first indexing part for every of the determined (u,v,a,b,γ)-sets as well as the specular peak is needed to form the q_{z,T_i} terms stated in equation (5.41). Same applies for the expressions $q_{\mathrm{spec},T}$ from equation (5.27). This is done prior to the generation of the matrices using the vectorized operations in `function_CALCSUBSIS.m` and `function_NEWPACKING.m`. The terms and substitutions for every partial solution are stored in matrices as part of three-dimensional numeric

arrays. This is based on the idea of real register cards, where a single page contains only information for one specific solution of (u, v, a, b, γ). The information corresponding to every set can be accessed by calling the index of the according page within a for-loop.

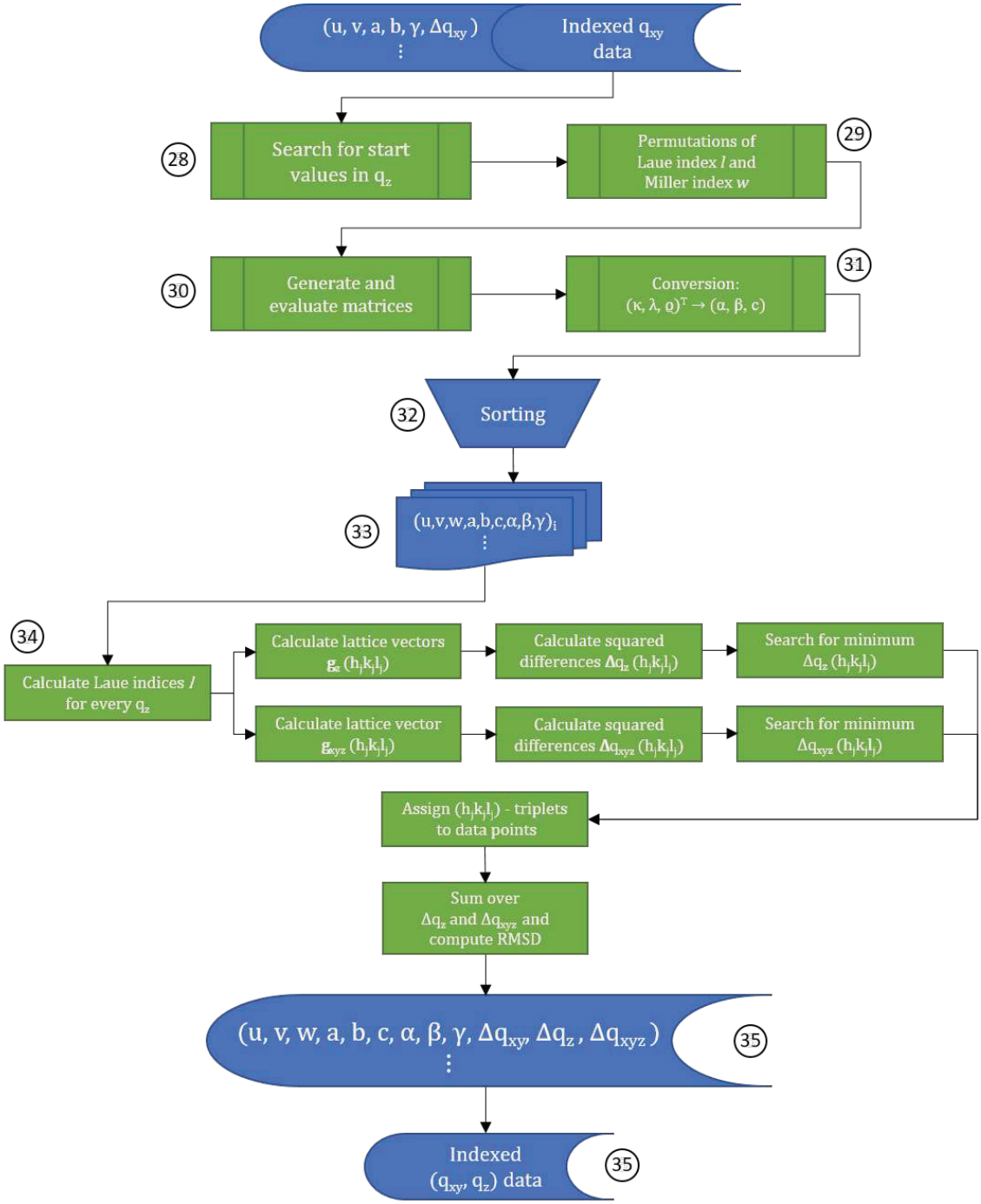


Figure 6.5: Work flow diagram for the indexing *Routine Part 2* with the input from the preceding blocks. The output contains whole unit cell solutions and indexed data points.

The following procedure is done for every loop iteration. In the first part of the routine in sub-chapter 6.3, every value of q_{xy} got tuples of Laue indices assigned. For every value q_{xy}, the corresponding q_{z,T_i} and $q_{\mathrm{spec},T}$ terms are retrieved and the matrices are filled with the pairs (h_1k_1), (h_2k_2) and (h_3k_3) together with the two Miller indices u and v of one set. This step is marked by (30). The third Laue index, l, is varied in a range specified by the program parameter *limit_fuer_l*. The according parameter on the GUI is *Max. Laue index l* and the default limit is set to $l = \pm 6$. All the permutations are generated in this specified range with the known function `LPermutation()`, where the number of digits is 3 as the positions l_1, l_2 and l_3 have to be filled in the expression (6.6) above. The third Miller index, w, is varied in the range between -*Max. Miller index*-1 and *Max. Miller index*+1.

The generation in the special case simplifies to matrices of the form

$$\begin{pmatrix} 0 & 0 & w \\ h_1 & k_1 & l_1 \\ h_2 & k_2 & l_2 \\ h_3 & k_3 & l_3 \end{pmatrix} \begin{pmatrix} \kappa \\ \lambda \\ \rho \end{pmatrix} = \begin{pmatrix} q_{\mathrm{spec}} \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}. \tag{6.7}$$

The values of q_z as well as q_{spec} can be used directly from the sorted data and no further computation of substitutions is necessary. The variation of indices is identical for both cases and indicated in Figure 6.5 with number (29).
The systems are solved using again the backslash-operator for linear systems of equations. This is the same functions as `mldivide(A,b)` for a matrix equation $\mathbf{Ax} = \mathbf{b}$. As the linear systems are overdetermined (i.e. more rows than columns), the function returns no exact solutions, but the least-squares solutions in $(\kappa, \lambda, \rho)^{\mathrm{T}}$. These are found by internally minimizing `norm(A*x - b,2)`. At this stage of the algorithm, the least-squares solutions are sufficient, as the parameters are optimized in a later sub-routine. The norm of the residual `(b-A*x)` can be evaluated and used as a sorting quantity. The norm of the residual is zero if an exact solution is obtained.
With the solutions obtained from the linear systems, the sets of $(u, v, w, a, b, c, \alpha, \beta, \gamma)$ can be computed using the earlier stated relations in (5.11). This conversion is marked with (31) in the flow chart. The sets are sorted and filtered according to Niggli's criteria in (32). A summary of the scalar-product criteria which include all six lattice constants can be found in Table 3.1. The numerical restrictions are also applied at this point. Again, only sets of solutions which pass the applied rules will be considered for the last indexing step. The sets are indicated by the symbol shaped like register cards in (33).

The general relation stated in (5.31) is considered for the second indexing procedure. Note that the relation reduces to equation (5.12) if $u = v = 0$. As all variables are derived at this point, the numerical values of the third Laue index l' can be computed by rearranging the equation to

$$l' = \frac{1}{c^*}\left(q_{z,T_i} - ha^*\cos\beta^* - kb^*\cos\alpha^*\right). \tag{6.8}$$

The apostrophe should emphasize that these are real floating-point numbers. Four possible tuples (hk) where assigned to every data point is the first part. To obtain the four according Laue indices l for each diffraction peak tuple (q_{xy}, q_z), the numbers are rounded to the next integer numbers. With that, the according values for g_z (equation 5.26) as well as the values of g_{xyz} (equation 5.25) can be calculated with the substitutions and compared with the sorted experimental data by evaluating the errors of the form

$$\Delta_{q_z}(h,k,l) = \left[g_z(h,k,l) - q_z(h,k,l)\right]^2 \tag{6.9}$$

and

$$\Delta_{q_{xyz}}(h,k,l) = \left[g_{xyz}(h,k,l) - q_{xyz}(h,k,l)\right]^2. \tag{6.10}$$

The Laue triplet hkl with the lowest value $\Delta_{q_{xyz}}$ is assigned to the peak position (q_{xy}, q_z). In this way, one pick out of four possibilities is made by searching the minimum error individually for every of the N peak positions. This procedure is applied within loops to every peak position what allows again the comparison by the means of an overall error. The comparison is done by deriving the root mean square deviations

$$RMSD_{q_z}(a,b,c,\alpha,\beta,\gamma) = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\Delta q_{z_n}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(g_{z_n} - q_{z_n})^2} \tag{6.11}$$

and

$$RMSD_{q_{xyz}}(a,b,c,\alpha,\beta,\gamma) = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\Delta q_{xyz_n}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(g_{xyz_n} - q_{xyz_n})^2}. \tag{6.12}$$

These errors are stored together with the associated unit cell solutions and provide a quantity to sort and rate the obtained sets. Beside the numerical solutions for $(u, v, w, a, b, c, \alpha, \beta, \gamma, \Delta_{q_{xy}}, \Delta_{q_z}, \Delta_{q_{xyz}})$, three-dimensional arrays containg the indexed data are submitted to the last block.

6.5 Postprocessing

Within the last block, the unit cell solutions containing all six lattice constants as well as the deviations regarding the scattering vectors are obtained. Only those solutions are released and indexed, which follow the user-defined numerical restrictions as well as the crystallographic conventions according to Niggli (*cf.* [42]). The formalism stated earlier does not allow to determine a unique mathematical unit cell solution. For this reason, an additional sub-routine is implemented to check if the obtained solution corresponds to a reduced unit cell. The reduced cell based on the three shortest lattice vectors is referred to as Buerger cell. This solution of the cell is used for the next and final sub-routine, the numerical optimization.

The derivation of the Buerger cell is based in the algorithm stated in Chapter 3.2. The work flow is a part of the function `function_PART_TWO_v12.m` and outlined in Figure 6.6. The program is set up to generate and compute vectors of the form

$$\mathbf{v} = 2\pi\mathbf{G}^{-1}\mathbf{m} \tag{6.13}$$

where $\mathbf{G}^{-1}$ contains a triple of linearly independent reciprocal lattice vectors (*cf.* equation 3.14) and $\mathbf{m}$ is a vector containing integer numbers imitating the Laue indices. In a first step, indicated by (36), the unit cell parameters are converted to reciprocal constants to build the matrices $\mathbf{A}^*_{001}$ and subsequently the lattice vectors $\mathbf{g}$. The Laue indices obtained from indexing are iteratively stacked to matrices as stated in the right equation in (3.14). The determinant of every possible matrix is computed and sorted in ascending order. The vectors $(h_1,k_1,l_1)^{\mathrm{T}}$,$(h_2,k_2,l_2)^{\mathrm{T}}$ and $(h_3,k_3,l_3)^{\mathrm{T}}$ of the matrices where the value of the determinant equals to 1 are used to compute three reciprocal lattice vectors of the form

$$\mathbf{g} = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix} = \mathbf{A}^*_{001} \begin{pmatrix} h \\ k \\ l \end{pmatrix} \tag{6.14}$$

for the case of a contact plane with Miller indices $(00w)$ and

$$\mathbf{g} = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix} = \mathbf{R}\mathbf{A}^*_{001} \begin{pmatrix} h \\ k \\ l \end{pmatrix} \tag{6.15}$$

for the case of a general contact plane (uvw).

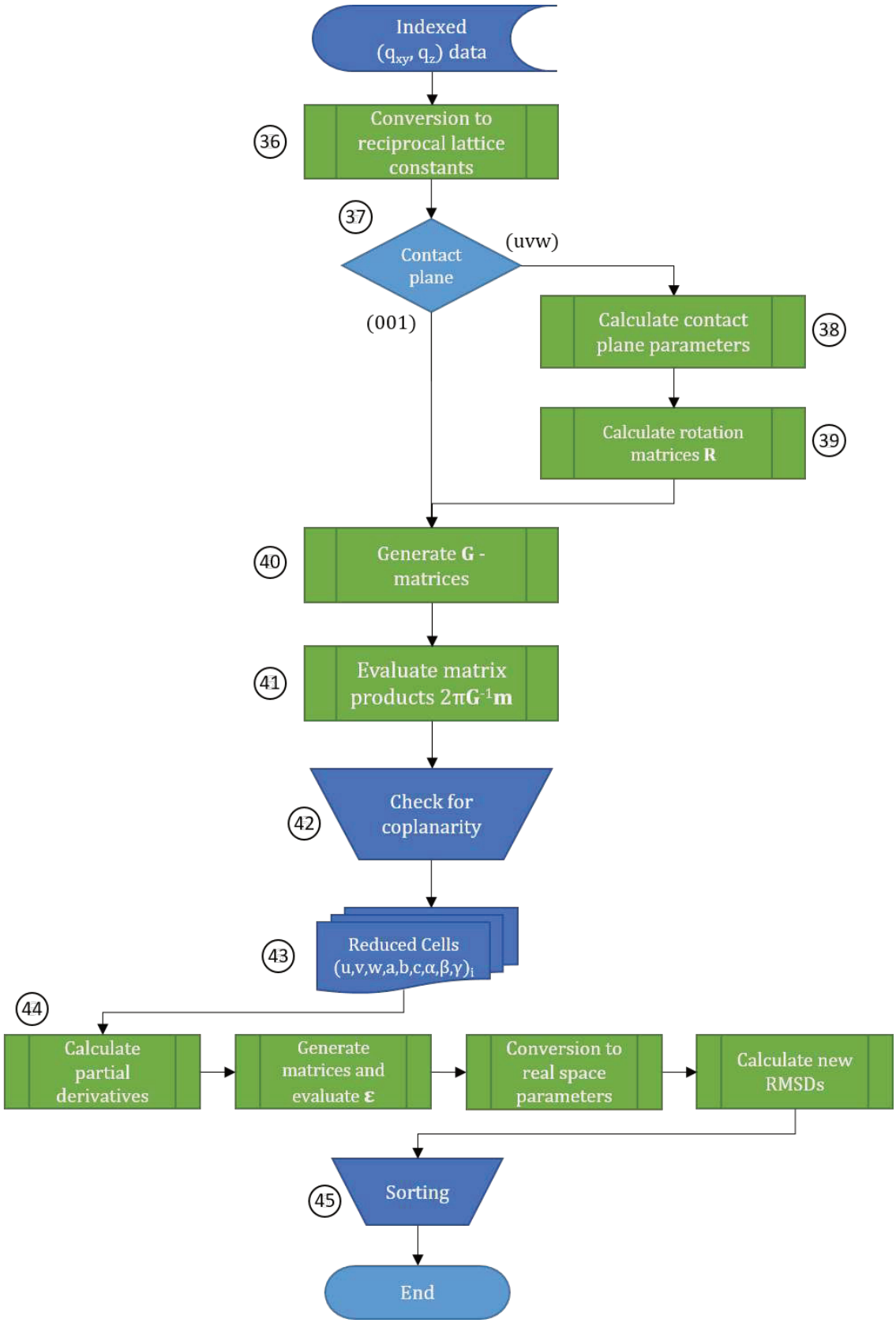


Figure 6.6: Flow chart containing the determination of the reduced cell as well as the numerical optimization with respect to the deviation in q_z and q_{xyz}.

The distinction between these two cases is indicated by the decision box (37) and the corresponding functions within the attached MATLAB code are `function_REDUCED_CELL_MY_001.m` and `function_REDUCED_CELL_MY_UVW.m` for the special case and for the general case, respectively. For the general case, the rotation around the zone axis has to be considered by including the rotation matrix **R**. The determination of the contact plane parameters (σ_1, σ_2, $\boldsymbol{n}$, Φ) and the subsequent rotation matrix is outlined by (38) and (39) in the flow diagram 6.6. The equations to determine the contact plane parameters are stated in Chapter 3. For the generation of potential vectors of the reduced cell **v** (equation 6.13), integer numbers in the range of $(-6,6)$ are varied to build vectors of the form $\boldsymbol{m} = (m_1, m_2, m_3)^{\mathrm{T}}$. This is done with the known function `LPermutation(([-6:6],3))` for the three numbers m_1, m_2 and m_3. In (40), the matrices **G** are formed by using the three reciprocal lattice vectors from above. For every combination of m_1, m_2 and m_3, the vector **v** is evaluated in a loop with equation (6.13), as marked by (40). The norm of each of these resulting vectors is sorted in an ascending order and the three shortest and non-coplanar vectors are stored as new unit cell vectors. Three vectors are considered coplanar if their scalar triple product is zero (i.e. a vanishing volume). In this routine indicated by (42), the coplanarity is tested by evaluating the determinant of $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Any combination which yields a non-zero determinant is considered to be non-coplanar and the resulting vectors of the reduced cell are obtained.

In this way, every unit cell solution is tested if it follows the algorithm of the reduced cell. The resulting sets of unit cell solutions are numerically optimized with respect to the deviation in q_z and q_{xyz}. The corresponding sub-routine in indicated by (44) and is a part of the two codes for the reduced cell. The algorithm is the same for both, the general case and the special case as the general formulae reduce to the one stated in special case for $(uvw) = (001)$. Using the reciprocal lattice constants, the partial derivatives according to the relations (5.48) and (5.49) have to be derived. These numerical differentiations are computed by subtracting a backward expansion from a forward expansion and dividing the interval by the total interval size $2\delta = 2 \cdot 10^{-5}$. This is known as central difference method. For general function $f(\chi)$, this can be written as

$$\frac{df(\chi)}{d\chi} = \frac{f(\chi+\delta) - f(\chi-\delta)}{2\delta}. \tag{6.16}$$

The computation of the derivatives and the construction of the corresponding matrix stated in relation (5.47) is done using a for-loop. This, and the sum over the N data points, is embedded in the function `function_EPSILON_UVW.m`. The evaluated matrix yields the so-called ε-vector. Every entry of this vector represents a correction term for a

reciprocal lattice constant. The correction terms are added in a vectorized form to the lattice constants and the real space parameters are computed subsequently by the use of the relations stated in the Table 5.1. With the new unit cell parameters, new numerical values for g_{xy}, g_z and g_{xyz} can be determined. This allows to evaluate the new numerical errors in the form of the root mean square deviations defined earlier. This algorithm, containing the generation of the derivatives, generation of the matrices, conversion of the parameters and the conclusive determination of the deviations in summarized in (44). In a last step, the solutions are sorted with regard to the RMSD in the total length q_{xyz} of the scattering vectors. This parameter has shown to be most reliable when estimating the 'quality' of an obtained result. This result is released in the form of output files containing the unit cell solution with corresponding errors as well as a list of the diffraction peaks with assigned Laue indices.

With this last building block, the description of the *Indexing Routine* is complete. Starting with formatting in the *Initialization* block, the experimental diffraction data are partially indexed in q_{xy} upon derivation of sets of (u,v,a,b,γ) in the *Routine Part 1*. The sets and the indexed data serve as input for the *Routine Part 2*. Within this sub-routine, the remaining cell constants are determined and the diffraction data are assigned by three Laue indices *hkl*. All determined solutions are evaluated with regard to crystallographic conventions stated by Niggli and the reduces Buerger cells are derived. The output contains the numerically optimized unit cell parameters and the the indexed peak positions in formatted text files. The whole routine is embedded within a graphical user interface (*cf.* Chapter 7) and its functionality is demonstrated on four independent examples in Chapter 8.

Chapter 7

Instruction Manual and Tutorial

With the following chapter, the graphical user interface (GUI) of the indexing routine is introduced. The main window consists of five different sub-windows, hereinafter referred to as *panels*. Figure 7.1 shows a screenshot of the main window with the five panels marked in different colours.

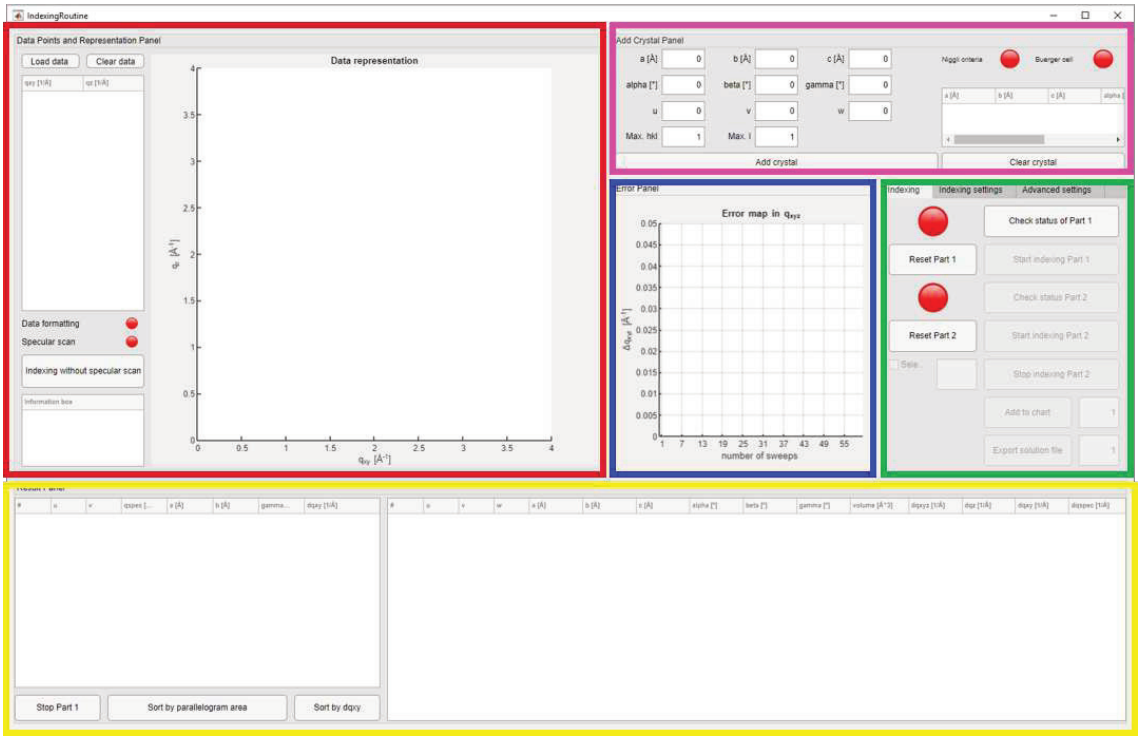


Figure 7.1: Screenshot of the indexing routine's main window right after start and prior to any input.

Below, the possible settings and features of every panel are explained individually. The panels are indicated and described in the following order:

(i) *Add Crystal Panel* in magenta, (ii) *Data Points and Representation Panel* in red, (iii) *Indexing Panel* in green, (iv) *Error Panel* in blue and (v) the *Result Panel* indicated in yellow. For demonstration purposes, an organic thin film sample with the name "Cu INA"-MOF is used. The peak positions are derived and provided by *Lukas Legenstein* using the software tool GIDVis [39].

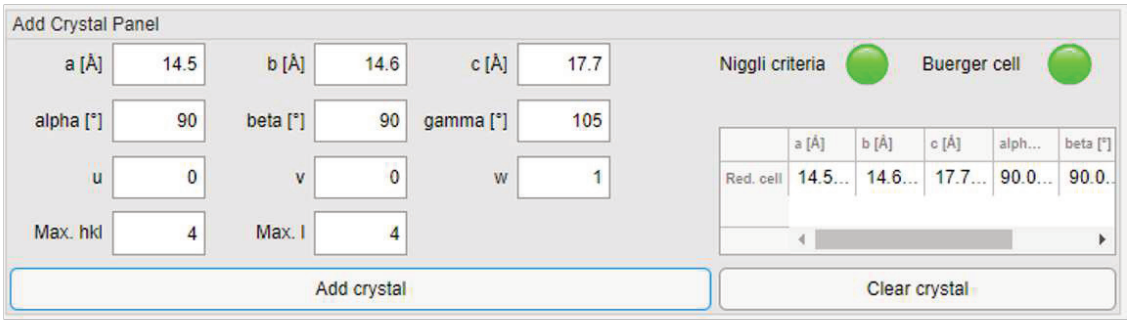
The program is provided in the form of a MATLAB application (.mlapp-file), written and tested in MATLAB version R2019b, Update 5. One important prerequisite to operate the indexing routing is the MATLAB *Parallel Computing Toolbox*™. As previously mentioned, the toolbox helps to reduce computation time through dividing program blocks in smaller units. After successful download, the app should appear in the tab 'APPS' in your MATLAB interface. Beside the toolbox for parallel computing, no further installation is required and the app can be used right away. To access the program, please contact *manuel.kainz@student.tugraz.at*.

7.1 Add Crystal Panel

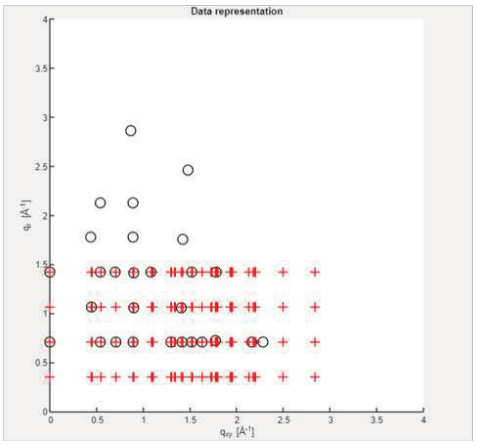
The *Add Crystal Panel* is based on the formulas stated in the Chapter *Indexing Formalism*. A screenshot is shown in Figure 7.2a. The panel serves two purposes. The first is to check whether or not a cell fulfills Niggli's criteria and the criteria for a reduced cell according to Buerger. The second purpose is to compare generated (ideal) diffraction peaks with data from GIXD experiments. It allows to quick-check unit cell configurations and contact plane indices. Every unit cell input is tested for Niggli's criteria and the reduced cell according to Buerger is determined. If the criteria are met, the icon on the left-hand side ('Niggli's criteria') is switched to green. The type of the cell is indicated in the 'Information box' (*cf.* Figure 7.2d). If a cell does not meet the criteria, a warning message pops up: 'Warning: Entered cell does not meet Niggli criteria for a reduced cell.'. When confirmed by pressing 'OK', the peak positions are calculated with the provided unit cell data and added to the graph in the *Data Points and Representation Panel* (*cf.* Figure 7.2b). The icon remains red in this case.
The lattice constants of the reduced cell are printed in the table on the right-hand side (*cf.* Figure 7.2c). If the input leads to a unit cell which can be reduced by the earlier introduced Buerger-algorithm, the newly derived parameters are printed. Otherwise, the input parameters are displayed at this location. As soon as the button 'Add crystal' is pressed and the input is reviewed, the icon with the label 'Buerger cell' turns green. The

button with the label 'Clear crystal' can be used to remove the input from the panel. All data points from the *Data representation* graph will be removed in this case.

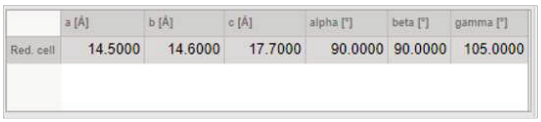
a)



b)



c)



d)

Figure 7.2: (a) Screenshot of the indexing routine's *Add Crystal Panel*. (b) Image of the *Data representation* graph with loaded diffraction peaks (black circles) and generated peak positions (red crosses). (c) Table with parameters of the reduced cell. The table is cut out from (a) and scaled up to show all six lattice constants. (d) Icon with information box indicating the type of the cell according to Niggli's criteria.

The *Add Crystal Panel* is an additional feature and does not include any indexing of data points. A diffraction peak can be calculated in a straightforward manner, if all six lattice constants as well as the Miller and the Laue indices are known. With these input parameters, the ideal diffraction patterns are generated. The generated peak positions are added to the *Data Points and Representation Panel*. If data points (diffraction peaks) are loaded, they can be easily compared with each other (*cf.* Figure 7.2b). The input 'Max. hk' limits the maximal number of the two Laue indices h and k. It is separated from the input variable 'Max. l' to allow more variability when testing generated patterns. Peak positions are calculated for permutations of Laue indices in the range from $(-Max.hk)$

to $(+Max.hk)$. The same applies for the maximal value of l. Plotting the generated diffraction points can help to confine the range of potential unit cell solutions. This *trial and error* approach is sometimes useful to figure out constraints in lengths and angles. These constraints can be applied in the subsequent indexing procedure, as the panel introduced in sub-chapter 7.3 allows to set restrictions to unit cell parameters and contact plane indices.

7.2 Data Points and Representation Panel

The first part of the indexing workflow includes upload of the diffraction data. The buttons to upload (and remove) peak positions are located within the *Data Points and Representation Panel* and labelled ”Load data’ and ”Clear data’, respectively. If the properties of the input data are fulfilled, peaks can be uploaded using the button ’Load data’. The data are printed in the table on the left-hand side (see Figure 7.3). Here, ’data point’, ’peak’ or ’peak position’ always refers to a (q_{xy}, q_z)–tuple. Successfully uploaded peak positions are plotted directly in the *Data representation* graph. The requirements for the input data are discussed in detail in Chapter 6. Most important, however, is that the file with the diffraction data should not contain any other data type than *numeric* floating-point numbers. A dot is used for decimal separation. When pressing the button to load data, a *Windows dialog box* appears and allows to select files of the type .xls or .xlsx. The data have to be provided as row vectors. The first column includes the q_{xy}–data, the second one the q_z–data. As this routine is designed to make use of the specular information, at least one entry in the first column has to be zero. An initialization program in the background searches for this data point. If at least one specular peak is detected, the icon labelled ’Specular scan’ toggles to green. If no such peak is contained in the uploaded file, a message pops up printing ’No specular peak detected. Please reload data!’. The user should add the specular information to the file or consider to use the button ’Indexing without specular scan’. In this case, a sub-routine is started which does the indexing of the diffraction peaks without any specular information. Hitherto, this is a rather time-consuming procedure. The inclusion of a specular scan is highly recommended.

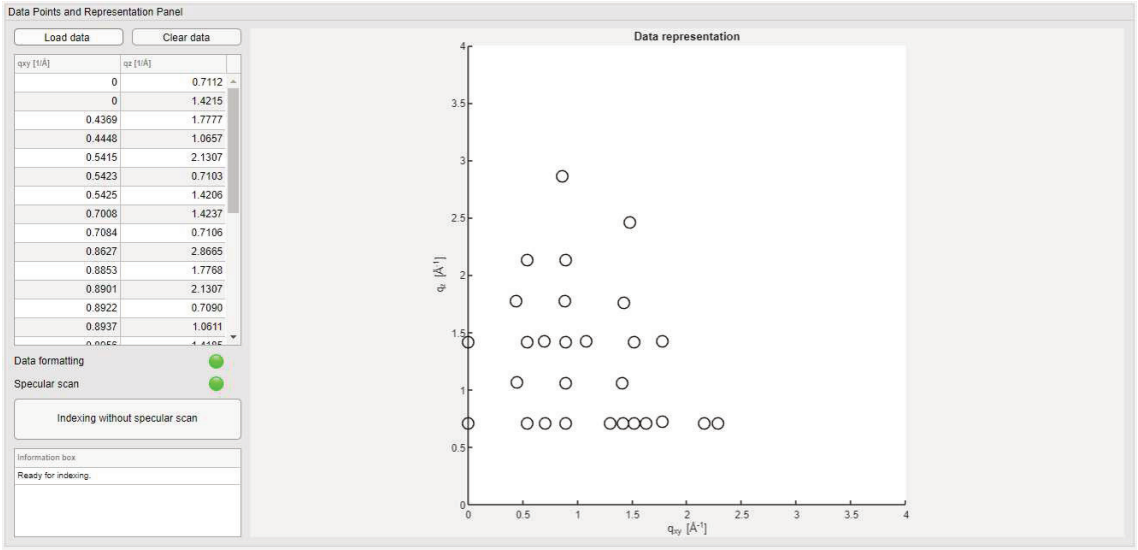


Figure 7.3: Screenshot of the indexing routine's *Data Points and Representation Panel*. An arbitrary set of peaks (black circles) is used to demonstrate the functionality.

The data set is not only checked for specular information, but also for the formatting. This includes the above mentioned representation as row vectors and the size of the data set (i.e. the number of data points). The icon labelled 'Data formatting' is switched to green if at least four (linearly independent) peaks are provided. This is the minimum number required to start this indexing routine. For data files with less than four peaks, the following message appears: 'Not enough peak positions for indexing. Please reload data!'. If the number of data points exceeds the value of 50, another message appears: 'The number of peak positions is above 50. Consider reducing the input'. To keep the computational effort in a reasonable range, this limit should not be exceeded in the beginning. If, however, enough restrictions (*cf.* Chapter 7.3) can be applied to the variables, the number of data points can be expanded without any constraint. After successful upload of the data points, the settings for the indexing algorithm have to be checked or adjusted.

7.3 Indexing Panel

The routine for indexing the uploaded GIXD data is controlled via the *Indexing Panel*. The region is marked with a green frame in Figure 7.1. Three tabs are used to control the routine and to provide maximal manual adjustability. The first tab, labelled *Indexing*, is used to review the input and to start, stop and reset the calculations. A screenshot is shown in 7.4. The button 'Check status of Part 1' triggers a subprogram to request certain parameters necessary for the routine. Such parameters are for instance the number of

data points, contact plane settings (Miller indices), restrictions of lattice constants and presence of a specular peak. The panel interface right after start is shown in Figure 7.4a. Red icons indicate that no checks are requested. The buttons to control the routine are disabled in this case. If all conditions of the subprogram are fulfilled and 'answered', the upper icon turns green and further buttons are enabled, as shown in Figure 7.4b. Incomplete requests are indicated by popup messages of the kind 'Declined. Please load data and check formatting and specular scan!'.

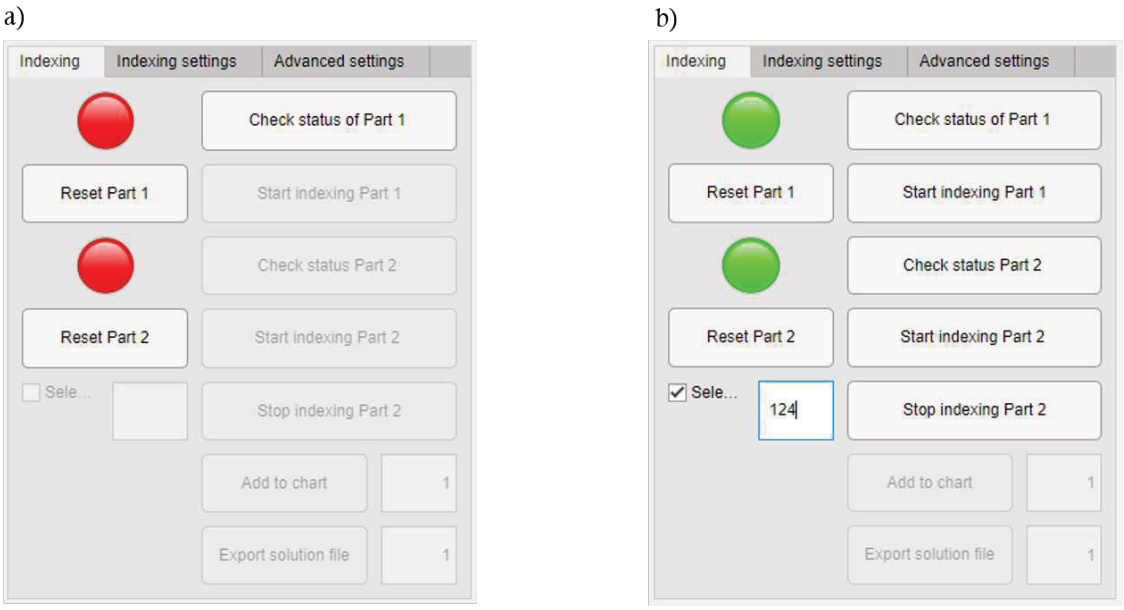


Figure 7.4: Screenshot of the routine's *Indexing Panel*. The image shows the tab 'Indexing' for two different states: (a) Right after start and (b) while the indexing routine is running

The indexing routine can be started by pressing the button 'Start indexing Part 1'. After indexing the data in q_{xy}, the results for the (a, b, γ)-sets are shown in the *Result Panel*. The status for the second part has to be checked by pressing 'Check status Part 2'. If both icons are set to green, the second part of the indexing routine can be started by pressing the button 'Start indexing Part 2'. The buttons 'Reset Part 1' and 'Reset' part 2' allow to set the routine to initial state (*cf.* Figure 7.4a).

Before starting the indexing procedure, certain adjustments can be done using the tabs *Indexing settings* and *Advanced setting*. In any case, the settings should be reviewed after successful upload of the peak positions. As mentioned earlier, applying constraints to the unit cell parameters can help to reduce computational effort. This can be done in the tab labelled as *Indexing settings*. The range for every of the six lattice constants as well as the corresponding cell volume can be limited by tick off the according checkbox placed on the left-hand side of the panel (*cf.* Figure 7.5). The 'Contact plane settings'

are located at the bottom of the panel. Three modes are provided. If the first checkbox is enabled, the Miller indices are set to the values $u = 0$, $v = 0$. The indexing program for the special case is applied in this case. The third index, w, is varied in the range from $(-Max.Millerindex)$ to $(+Max.Millerindex)$. This setting can be changed in the *Advanced settings* tab, row 5. Only one checkbox can be chosen at a time. The second checkbox allows to define one specific set of indices, without any permutations. The indices have to be separated by using a comma in the form u,v,w. The input has to be confirmed by pressing the ENTER key and is included to the algorithm by pressing 'Apply changed settings'. Changes are only accepted after this button is pressed. The messages in the 'Information box' inform about the current state of the settings. All input can be set to default settings by pressing the button 'Reset settings'.

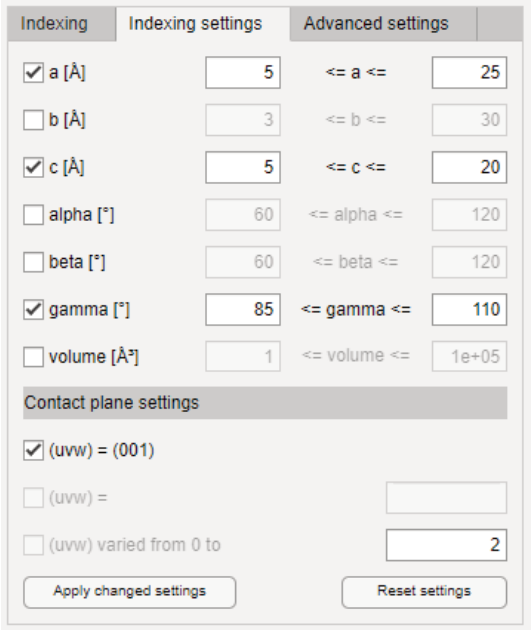


Figure 7.5: Screenshot of the routine's *Indexing Panel*. The image shows the tab labelled as *Indexing settings*. For demonstration purposes, three unit cell parameters are restricted and the 'Contact plane settings' are adjusted for the special case.

The last checkbox allows to define a permutation range for the three Miller indices of the contact plane. In Figure 7.5, the input is set to a value of 2. In this case, every possible permutation of (u,v,w) in the range from -2 to $+2$ is generated and tested by the indexing program. The limit for the permutations can be set in the tab labelled as *Advanced settings* via the variable 'Max. Miller index'. After the variables are reviewed and/or adjusted in the *Indexing settings* tab, the routine is ready to use. Further tuning of certain variables is possible in the tab labelled as *Advanced settings*, however, not

necessarily required. This tab requires some understanding of the algorithm proposed in Chapter 6. It may occur that no unit cell solutions can be obtained by use of the default settings or that only solutions with unsatisfying errors in the components of **q** are derived. In this case, re-adjusting of the program parameters in the *Advanced settings* tab can be useful. Even if solutions are obtained, the sets can be refined by the use of these settings (i.e. by change of maximum order of the Laue indices). The tab contains eight parameters (*cf.* Figure 7.6). As explained in Chapter 6, determination of potential unit cell solutions requires solutions to linear systems of equations (LSEs).

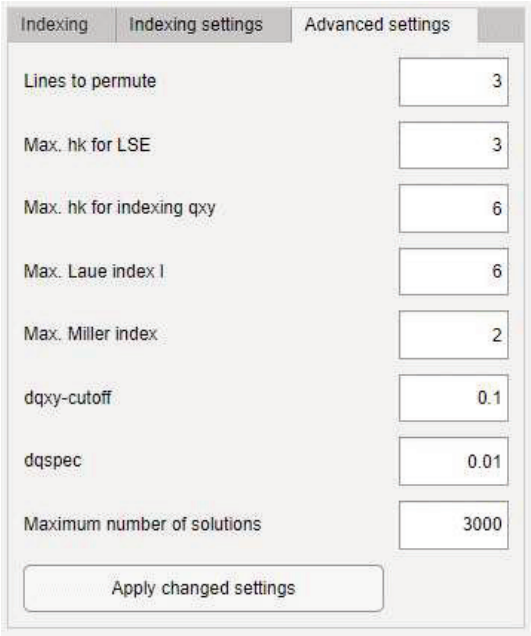


Figure 7.6: Screenshot of the routine's *Indexing Panel*. The image shows the *Advanced settings* tab. The settings should not be changed, unless enough restrictions to the unit cell parameters can be applied.

The start set needs three linearly independent pairs of (q_{xy}, q_z). If the parameter 'Lines to permute' is set to the value of 3, a subprogram checks for linear dependence and sets up LSEs for these three values with every permutation in the Laue indices between $(-Max.hkforLSE)$ to $(+Max.hkforLSE)$. With increase of the parameter 'Lines to permute' more combinations are considered what makes it more likely to determine solutions. After possible solutions are provided through solving the LSEs, the in-plane components of **q** are indexed. This occurs by assignment of the Laue indices and subsequent evaluation of the RMSD. The program parameter 'Max. hk for indexing qxy' allows to specify the upper limit for this indexing. In a refinement run (i.e. after the range of the lattice constants can be restricted), this value can be increased to a value of 8

or more. It is advised not to change this parameter in the beginning. The parameter 'Max. Laue index l' defines the limit for the second set of the LSEs. As discussed earlier for the *Indexing settings*, the maximal range for the Miller index permutations is defined by the parameter 'Max. Miller index'. The last three program parameters of the *Advanced settings* tab relate to the output of the determined solutions. The table where the results are shown (marked in yellow in Figure 7.1) contains only solutions where the error in q_{xy} is lower or equal to 'dqxy-cutoff'. Furthermore, no solutions are indexed if the error in the specular peak is greater than 'dqspec'. If the routine is started and not stopped manually by the user, the algorithm will stop automatically as soon as the value 'Maximum number of solutions' is reached. All changes have to be confirmed by pressing 'Apply changed settings'. The status is then indicated in the 'Information box'.

7.4 Error Panel

During the indexing procedure, the root mean square deviation (RMSD) is calculated for the components q_{xy} and q_z as well as for the length q_{xyz} of the scattering vector. The length can be obtained from the experimental data by $q_{xyz} = \sqrt{q_{xy}^2 + q_z^2}$. The error in the length turned out to be a promising indicator for the choice of a possible unit cell solution and it is therefore used as a sorting quantity. For this reason, the error is displayed throughout the indexing procedure over the number of sweeps (*cf.* Figure 7.7).

a) b)

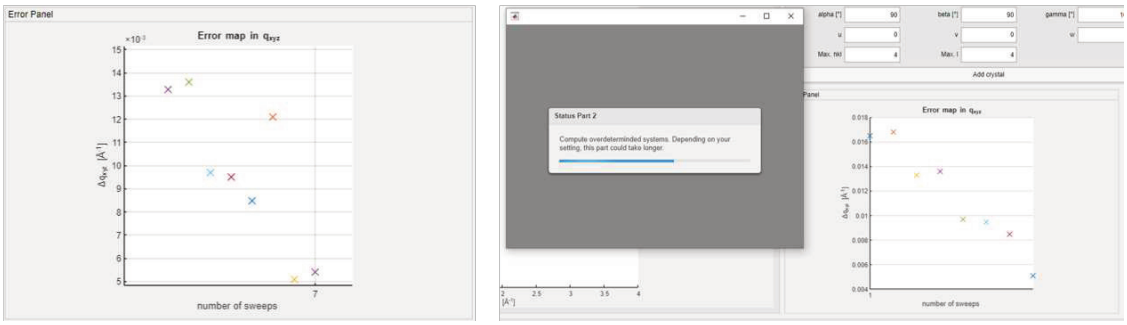


Figure 7.7: Screenshot of the indexing routine's *Error Panel*. In (a) the image shows the panel as embedded in the graphical user interface. The image in (b) shows a cutout from the main window while the routine is running. The error decreases with every sweep and the routine is stopped after the error is below 0.6 %.

One sweep contains the obtained solutions from one set of (a, b, γ). The minimal error of these solutions is printed in the graph labelled as 'Error map in q_{xyz}'. The indication can be used to monitor the error and to stop the algorithm if a proper value is met.

7.5 Result Panel

The unit cell solutions obtained from the indexing routine are presented in the form of two tables. These tables are located in the *Result Panel* and marked with a yellow frame in Figure 7.1. To give a better overview, the panel is divided in two and shown separately in Figure 7.8. Every row corresponds to one possible solution. The solutions obtained from the first part are stored in the left table (*cf.* Figure 7.8a) and labelled with a number (#). The output is sortable by every quantity shown in the table by pressing the ↑↓-icon, right beside to the variable names. In addition, the sets can be sorted by smallest area by pressing 'Sort by parallelogram area'. The default sorting can be restored with the button 'Sort by dqxy'. As the first part of the algorithm is based on parallel-executed loops, it

can be stopped only during the initialisation phase. Usually this takes up to 20 seconds after start up. To this point, the program can be stopped by pressing 'Stop Part 1'.

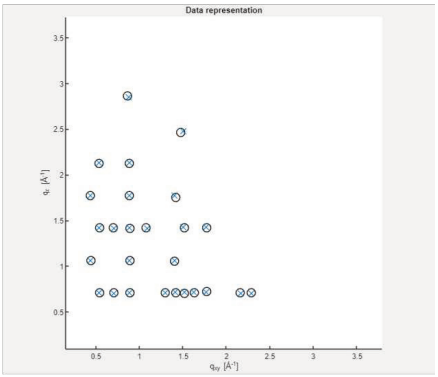
a)

Result Panel

#	u	v	qspec [...	a [Å]	b [Å]	gamma...	dqxy [1/Å]
74.0000	0	-1.0000	0.7112	14.0866	14.5608	90.7164	0.0196
75.0000	0	1.0000	0.7112	14.0866	14.5608	90.7164	0.0196
76.0000	0	-1.0000	0.7112	14.0873	14.5843	89.1428	0.0193
77.0000	0	1.0000	0.7112	14.0873	14.5843	89.1428	0.0193
78.0000	0	0	0.7112	11.5870	17.7383	89.9973	0.0251
79.0000	0	0	0.7112	8.8691	23.1740	89.9973	0.0288
80.0000	0	0	0.7112	14.4845	14.7017	74.8620	0.0077
81.0000	0	0	0.7112	14.4845	14.7017	105.1380	0.0077
82.0000	1.0000	-1.0000	0.7112	12.5144	16.9678	90.1806	0.0133
83.0000	-1.0000	1.0000	0.7112	12.5144	16.9678	90.1806	0.0133
84.0000	-1.0000	-1.0000	0.7112	12.5142	17.0289	89.9206	0.0134

Stop Part 1 | Sort by parallelogram area | Sort by dqxy

c)



b)

#	u	v	w	a [Å]	b [Å]	c [Å]	alpha [°]	beta [°]	gamma [°]	volume [Å^3]	dqxyz [1/Å]	dqz [1/Å]	dqxy [...	dqspec [1/Å]
54	0	0	2	14.5372	14.6549	17.7390	90.3691	90.1334	104.8954	3.6520e+03	0.0062	0.0067	0.0084	0.0043
55	0	0	2	14.5155	14.6711	17.7750	90.4185	89.8276	75.0913	3.6578e+03	0.0062	0.0071	0.0086	0.0066
56	0	0	2	14.5188	14.6937	17.6448	90.4240	89.9383	75.3712	3.6421e+03	0.0062	0.0076	0.0088	0.0024
57	0	0	2	14.5123	14.6747	17.7785	90.4386	90.1571	104.8541	3.6595e+03	0.0062	0.0071	0.0085	0.0066
58	0	0	2	14.5197	14.7057	17.6680	89.6800	89.9962	74.8821	3.6419e+03	0.0062	0.0073	0.0087	3.5834e-04
59	0	0	2	14.5286	14.7004	17.6711	89.7291	90.0530	74.8658	3.6432e+03	0.0062	0.0071	0.0091	2.8270e-04
60	0	0	2	14.5140	14.7083	17.6708	89.6858	90.0260	105.0585	3.6427e+03	0.0062	0.0072	0.0085	3.5834e-04
61	0	0	2	14.5120	14.7066	17.6543	90.2178	89.7406	74.8974	3.6376e+03	0.0062	0.0071	0.0087	0.0015
62	0	0	2	14.5200	14.6678	17.7142	90.3615	90.1514	104.6062	3.6507e+03	0.0063	0.0061	0.0086	0.0023
63	0	0	2	14.5443	14.6825	17.6522	89.6743	90.0496	75.2833	3.6458e+03	0.0063	0.0073	0.0087	0.0020
64	0	0	2	14.5550	14.6975	17.6953	89.7491	90.1075	74.9384	3.6553e+03	0.0063	0.0070	0.0082	8.4961e-04
65	0	0	2	14.5308	14.6602	17.7431	89.6127	90.1213	75.1293	3.6530e+03	0.0063	0.0066	0.0082	0.0049
66	0	0	2	14.5060	14.7299	17.6318	90.3538	90.2923	105.0608	3.6378e+03	0.0063	0.0076	0.0194	0.0028

Figure 7.8: Screenshots of the indexing routine's *Result Panel*. (a) shows the table containing the results from the first indexing part. The complete unit cell solutions obtained from the indexing routine are presented in the table shown in (b). The image in (c) shows a cutout from the *Data Points ans Representation Panel*. The peak positions calculated from one solution (blue) are printed together with the experimental data points (black).

The final results after the second indexing part are presented in the table located at the right-hand side. It is shown as cutout in Figure 7.8b. The solutions are labelled with numbers and sorted by the RMSD in q_{xyz}. Again, all solutions can be sorted with the ↑↓-icon. For demonstration purposes, the solution with row number 58 is selected and added to the *Data representation* graph (*cf.* Figure 7.8c).

7.6 Tutorial

The application of the program is presented in terms of a short tutorial on a organic thin film sample ("Cu INA"- MOF). A set of 26 peak positions is extracted from GIXD

patterns using the software tool GIDVis [39]. Two additional data points are obtained from XRD experiments measured in specular condition. This leads to a total of 28 peaks for the indexing procedure and for subsequent determination of a unit cell solution.

Having MATLAB installed, the app can be started with a double-click on the app-icon in the tab 'APPS'. The graphical user interface starts, expands to full-screen size and is then compressed to a default size. The icons may occur compressed or distorted, depending on the screen size. The GIXD data are uploaded by pressing the 'Load data' button and selecting the according file. The file has to be of the format .xls or .xlsx. It is possible that the main window is minimized during the selection. The window expands again as soon as a file is selected. The uploaded data points are added directly to the graph in the form of black circles. As all formatting requirements are fulfilled and two specular peaks are contained in the data, both icons are set to green. Next, the indexing settings are checked. As there's no prior information, the routine is operated in general mode to estimate the range of potential unit cell constants. The program parameter 'Max. Miller index' is set to 1. The Miller indices are therefore of the kind $(11w)$, $(10w)$, $(00w)$ and so on. By pressing the button labelled as 'Apply changed settings', the changes are accepted. The indexing procedure is initiated by checking the settings one more time by pressing 'Check status of Part 1'. A green lamp and enabled buttons indicate that the program is ready. The routine is started by pressing 'Start indexing Part 1' and 'Proceed' in the subsequently opened dialog box. After start up, a window containing status messages opens. This window informs the user about the current state of the algorithm. The very first run of the routine may takes more time due to startup of the MATLAB parallel pool.

The first part of the indexing routine is finished as soon as results appear in the left table of the *Result Panel*. To continue indexing, the settings have to be checked by pressing 'Check status of Part 2'. Two options are enabled for the further process: (i) start indexing the sets of (a, b, γ) in ascending order or (ii) ticking the checkbox and define specific lines to consider for the second indexing part. The checkbox is located below the button 'Reset Part 2'. The rows have to be specified using MATLAB syntax. The screenshot in Figure 7.9 shows the indexing routine in use. The checkbox is enabled and two rows are specified for the further indexing procedure. These rows are chosen as the corresponding sets in the *Results Panel* exhibit comparably low errors (highlighted in gray).

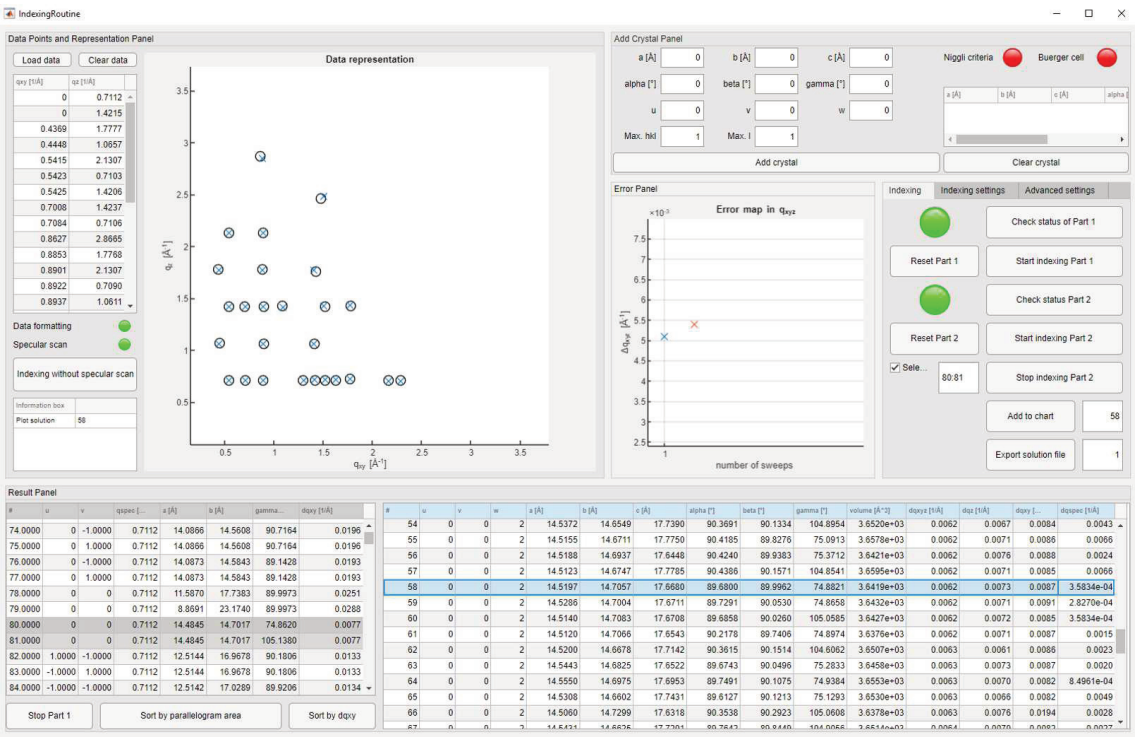


Figure 7.9: Screenshot of the indexing routine's *Main Window* in use. The indexing procedure is finished and the peak positions obtained from the solution highlighted in blue is added to the data representation graph.

By pressing the button 'Start indexing Part 2', the routine assigns the remaining indices and calculates the lattice constants α, β and c. In this example the routine is stopped after two sweeps as the error in q_{xyz} is in a range of below 1 %. With an error in this range, the solution can be 'tested' by adding the calculated diffraction pattern to the graph. The line number has to be entered in the field beside the 'Add to chart' button. After confirming by pressing the ENTER key and pressing the button 'Add to chart', the generated peak positions are added to the graph. The peak positions generated with this unit cell solution

show good agreement with the experimental data. For further comparison, the solution can be used as input for GIDVis. By use of the according Laue indices, the diffraction peaks can be plotted directly on top of a reciprocal space map (*cf.* Figure 7.10). The Laue indices are provided by the indexing routine in the form of a .xls-file. The file can be generated by input of the row number in the corresponding field and pressing the button 'Export solution file'. An example of such a file is shown in Figure 7.11. It is advised to use the software GIDVis together with the graphical comparison of the indexing routine to achieve the best results.

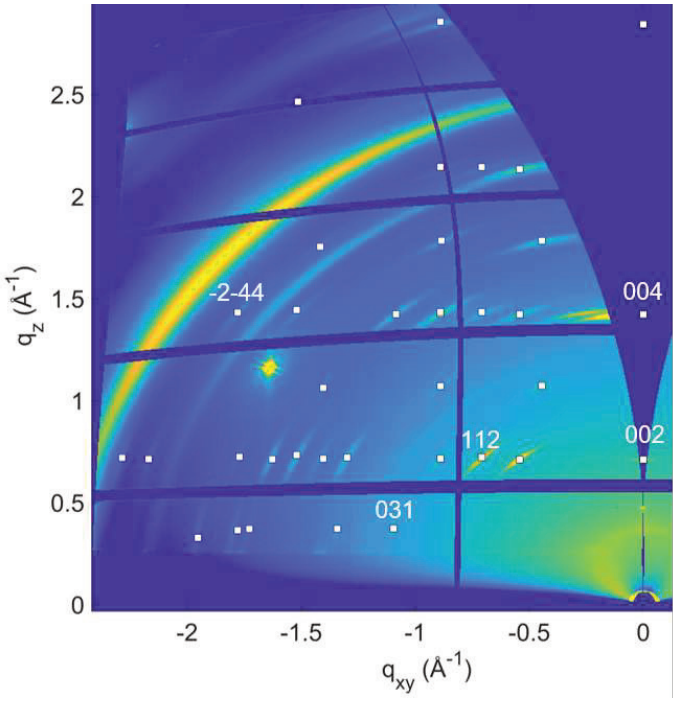


Figure 7.10: Screenshot of a reciprocal space map generated in GIDVis. The white squares represent the calculated peak positions using the derived unit cell solution and the Laue indices provided by the indexing routine. Five data points are presented with the according Laue indices.

The errors in length q_{xyz} and in the two components of the scattering vector are helpful criteria for the search of a unit cell solution. However, the final choice of such a solution depends also on other factors (e.g. highest symmetry of the cell and shortest lattice edge-lengths). The here shown unit cell solution is not the only possible solution. For the sake of a compact tutorial however, the discussion of other possibilities is postponed to Chapter 8.

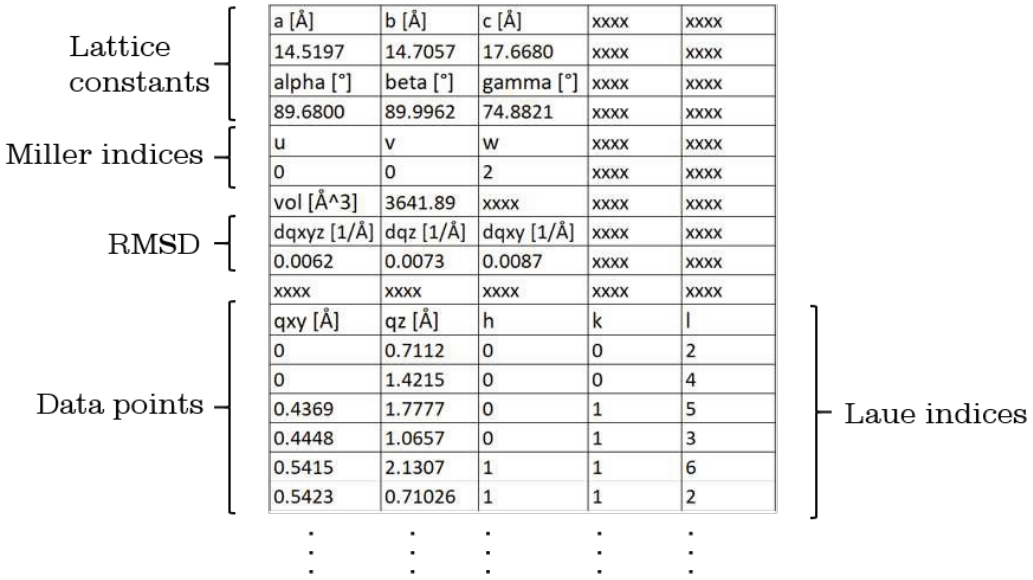


Figure 7.11: Screenshot of the output file generated by the routine. For the sake of brevity, the file is cut after six peak positions. The locations of the different output information are marked separately with braces.

The mathematically optimized solution chosen for this short tutorial is a cell of type I with the following unit cell parameters:
$u = 0$, $v = 0$, $w = 2$, $a = 14.52$ Å, $b = 14.71$ Å, $c = 17.67$ Å, $\alpha = 89.9°$, $\beta = 89.9°$ and $\gamma = 74.9°$. The volume of the reduced Buerger cell is $V = 3642.1$ Å^3 and the mean deviation in q_{xyz} is 0.6 %. An additional check using the *Add Crystal Panel* of the indexing routine confirms that the solution obeys the scalar-product criteria according to Niggli.

Chapter 8

Indexed Samples

The Indexing Routine is used for indexing and determination of the unit cell solution of four different thin film samples. One example, namely the Cu INA - MOF fIna-04, was shown earlier in sub-chapter 7.6. In this final chapter, the program is demonstrated on three further samples. The input data were derived using GIDVis and provided as .xls-files. The used data are explicitly printed in the Appendix A. Results are compared with published values from literature if available.

8.1 Diindenoperylene (DIP)

Grazing incidence X-ray diffraction data of a DIP thin film on highly oriented pyrolytic graphite (HOPG) are evaluated in this sub-chapter. Diindenoperylene is an organic semiconductor molecule with the chemical formula $C_{32}H_{16}$. A specular peak is provided at $q_z = q_{\mathrm{spec}} = 1.776$ Å^{-1} and used together with 11 other diffraction peaks for the subsequent indexing process. The routine is started with default contact plane settings where the Miller indices are varied in a range between -2 and $+2$. A total number of $5^2 = 25$ different combinations result which are tested in parallel manner. The partial solutions from the first indexing part are sorted by the RMSD in q_{xy} is ascending order. The second indexing part is cancelled after 30 sweeps as the error in q_{xyz} does not decrease with increasing number of possible sets. The routine is re-started with readjusted settings where the number of start sets is increased by setting 'Lines to permute = 5' and 'Max. hk for LSE = 4'. The contact plane settings are unchanged and the program runs again through 25 different pairs of Miller indices. The second indexing block yields first promising results after approximately 20 sweeps and 10 minutes. The errors are in the range of 1 % and below, from this point on. A screenshot from this state of indexing is shown in

Figure 8.1. The solution with unit cell parameters $a = 8.49$ Å, $b = 14.27$ Å, $c = 16.64$ Å, $\alpha = 87.1°$, $\beta = 89.2°$ and $\gamma = 89.9°$ is plotted in the *Data representation* graph. The corresponding derived contact plane is indicated by (uvw) =(-2-21). The errors can be explicitly given with $\Delta_{q_{xy}} = 0.006$ Å^{-1}, $\Delta_{q_z} = 0.001$ Å^{-1} and $\Delta_{q_{xyz}} = 0.004$ Å^{-1}. The numerical deviations are in a low region and as the calculated pattern does practically not depart from the experimental data, this solution is chosen for a further investigation. In a next step, the program is used to test specific contact plane variations based on the numbers derived above. Moreover, numerical restriction are made to concentrate the calculation on solutions close to the above determined solution range.

Figure 8.1: Screenshot of the main window of the Indexing Routine upon indexing of the provided DIP data. The solution from the *Result Panel*, marked in blue, is used to generate a simulated diffraction pattern. The pattern is added to the *Data Points and Representation Panel* for graphical comparison with the experimental data points.

After cycling through different permutations of the Miller indices consisting of combinations like (221), (2-12) or (121), a concluding mathematical solution can be recommended. The Miller indices can be stated as (uvw) =(-121) and the corresponding lattice constants are $a = 7.13$ Å, $b = 8.48$ Å, $c = 16.67$ Å, $\alpha = 89.4°$, $\beta = 87.8°$ and $\gamma = 89.7°$. The cell volume is derived with $V = 1070.1$ Å^3 and the mean deviations for this solution are $\Delta_{q_{xy}} = 0.002$ Å^{-1}, $\Delta_{q_z} = 0.005$ Å^{-1} and $\Delta_{q_{xyz}} = 0.002$ Å^{-1}. Note that a very similar

solution is contained two rows below the highlighted solution in Figure 8.1 above. Indexing and the resulting solutions of the DIP sample can be used as a vivid example for the interchangeability of the Miller indices and the determined unit cell parameters. From a mathematical point of view, every symmetric solution should be contained and provide an equivalent description of the crystallographic unit cell. This effect was observed for the defined Miller and Laue indices at the here shown example and is reported in literature [10]. The indexing procedure of this particular sample shows a different approach, compared to the example of Naproxen. Here, a promising solution is derived and investigated further by gradually limiting the cell parameters under testing of different Miller indices. Upon gradual application of the routine and its features, a reduced form of the preliminarily derived cell can be determined. A similar unit cell solution was reported earlier in literature [11, 51].

8.2 Pentacenequinone (PQ)

A total number of 30 diffraction peaks is used for indexing the GIXD pattern of a PQ thin film sample. 6,13-pentacenequinone is another example of an organic semiconductor material with the chemical formula $C_{22}H_{12}O_2$. The specular peak is derived from XRD and located at $q_z = 1.946$ Å^{-1}. For indexing of this sample, no changes at all are made in the first approach. The data set is uploaded and the indexing procedure is started right away with the default settings. The obtained sets of (u, v, a, b, γ) are processed without any further sorting. The routine is stopped manually after sweep 31. A screenshot of this state is shown in Figure 8.2 and the best ranked result (#1) can be explicitly stated as $u = 1$, $v = 0$, $w = 2$, $a = 5.06$ Å, $b = 8.08$ Å, $c = 8.87$ Å, $\alpha = 91.5°$, $\beta = 93.2°$ and $\gamma = 94.2°$. This is the reduced solution of the unit cell. The volume of this cell is $V = 360.8$ Å^3 and the mean deviation in q_{xyz} is 0.003 Å^{-1}. The RMSDs with regard to the components of the scattering vector are $\Delta_{q_{xy}} = 0.003$ Å^{-1} and $\Delta_{q_z} = 0.01$ Å^{-1}. As mentioned already for the DIP sample, mathematically symmetric solutions appear for every of the stated sets.

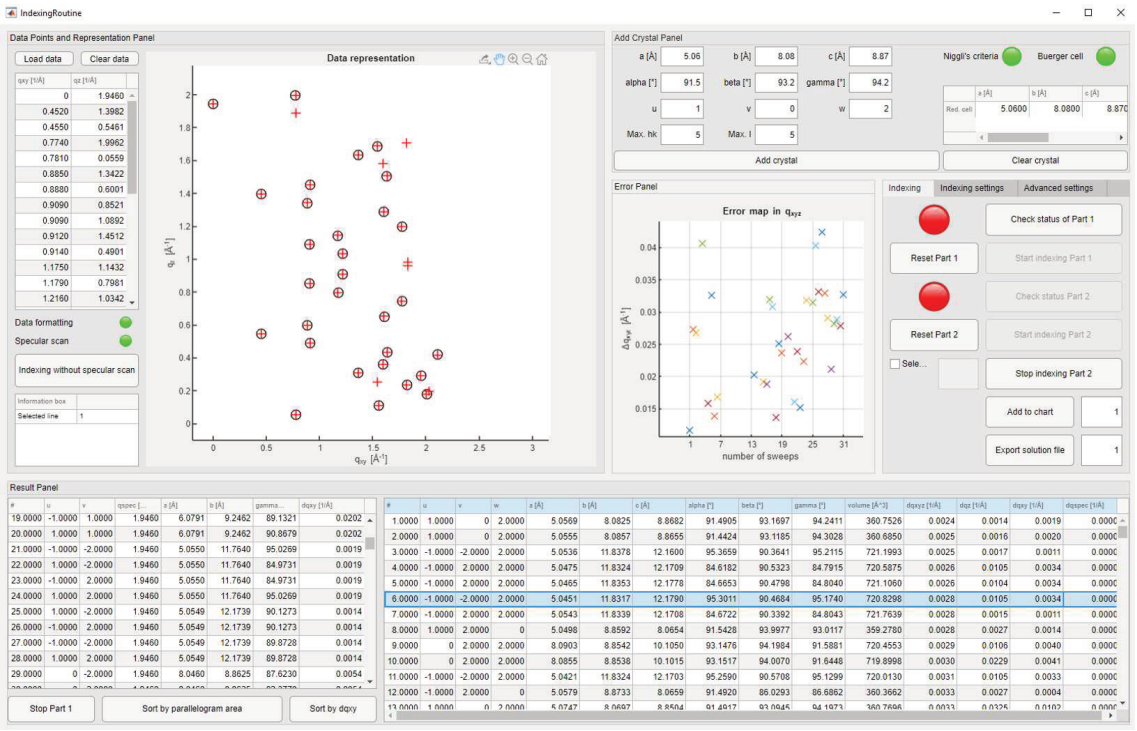


Figure 8.2: Screenshot of the Indexing Routine upon indexing of the provided PQ data. The two obtained solutions are shown within the *Result Panel*: The first one is ranked nr. 1 and the other solution is highlighted in blue. The experimental peak positions are printed as black circles and the the red crosses represent the generated diffraction pattern by applying the *Add Crystal Panel* for the first solution. Note that there are more data points generated as provided for indexing. For reasons of overview, generated peaks beyond $q_{xy} = 2.5$ Å^{-1} are deleted.

The second unit cell solution is explicitly given by the edge lengths $a = 5.05$ Å, $b = 11.83$ Å, $c = 12.18$ Å and the three angles are Å , $\alpha = 95.3°$, $\beta = 90.5°$ and $\gamma = 95.2°$. With a volume of $V = 720.8$ Å^3, the cell is approximately twice as large as the reduced cell stated above. The contact plane is derived with (uvw) =(-1-22) The deviations are in the same range with $\Delta_{q_{xy}} = 0.003$ Å^{-1}, $\Delta_{q_z} = 0.01$ Å^{-1} and $\Delta_{q_{xyz}} = 0.003$ Å^{-1}. Both unit cell solutions were found in literature [10]. Anyhow, a final comparison using the original GIXD data and GIDVis is recommended.

8.3 Naproxen

Diffraction data from a thin film sample of the organic molecule Naproxen are provided for indexing to investigate the crystalline phase. Naproxen is a pharmaceutical active ingredient and used for nonsteroidal anti-inflammatory drugs. A total of 30 peak positions are used for determination of a unit cell solution. The specular peak is obtained

from XRD measurements and is already included in the data set. In the first approach, the Indexing Routine is operated with contact plane settings $(uvw) = (001)$. No changes are done in the 'Advanced settings' and for the time being, no restrictions are applied to the unit cell constants. The first indexing part takes less than five minutes and approximately 130 partial solutions can be determined with an error smaller than 1 % in q_{xy}. With sorting the results according to the error in q_{xy}, the second part of the indexing procedure is pursued. The obtained unit cell solutions from the *Results Panel* are plotted and compared with the experimental peak positions using the *Data Points and Representation Panel*.

As the graphical comparison did not yield any comprehensive solution with the chosen settings, the routine is restarted. This time, the initial partial results are sorted regarding the smallest area of the parallelogram. Additionally, the program parameter 'Lines to permute' as well as the parameter 'Max. hk for LSE' are increased to a value of 4. The routine is stopped after 50 sweeps as the root mean square error, which is printed in the *Error Panel*, fell below 1 % in q_{xyz} a number of times. The unit cell constants a, b and c accumulate in regions around 10 Å, 20 Å and 25 Å, respectively. Similar patterns and repetitive behaviour is observed for the angles α, β and γ. The obtained solutions were added to the *Data representation* graph one after another for graphical comparison (*cf.* Figure 8.3). In addition, similar unit cell solutions were tested using the *Add Crystal Panel*. Indexing of the data set using general contact plane settings yields symmetric solutions with regard to the ones stated above.

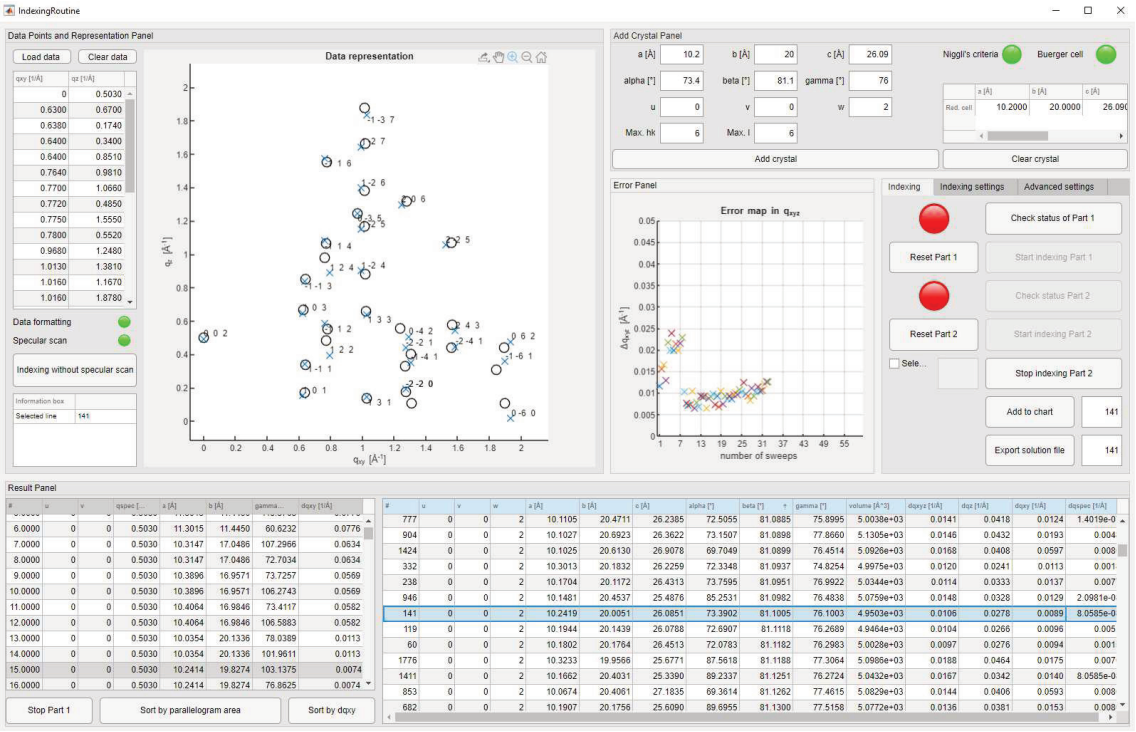


Figure 8.3: Screenshot of the main window of the Indexing Routine upon indexing of the provided data. The emphasized solution from the *Result Panel* is used to generate a simulated diffraction pattern. The pattern is added to the *Data Points and Representation Panel* for graphical comparison.

The best graphical agreement is recorded for a unit cell solution with the following parameters $u = 0$, $v = 0$, $w = 2$, $a = 10.24$ Å, $b = 20.01$ Å, $c = 26.09$ Å, $\alpha = 73.4°$, $\beta = 81.1°$ and $\gamma = 76.1°$. The volume of the reduced cell is $V = 4950.3$ Å^3 and the mean deviation in q_{xyz} is 0.01 Å^{-1}. A screenshot with the Indexing Routine's main window containing the as discussed solution is printed Figure 8.3 above. The overall time from loading the diffraction data down to a unit cell solution amounts to approximately one hour in this particular case. It has to be emphasized that this solution is not the only possible solution. No other solution than $(uvw) = (002)$ is observed for the contact plane with these settings. Sets with lattice constants similar to the one obtained appear frequently with only small differences in the numerical values. These differences, however, could lead to differently assigned Laue indices what makes the diffraction patterns overall different. Indexing and the task of searching a unit cell solution with this program is therefore an iterative exercise where the obtained possible solutions have to be verified eventually by the user. The routine does provide different numerically fitting solutions. A unique or indisputable solution for the provided GIXD data, however, can not be derived by application of this algorithm. In general, it is advised to compare the diffraction data

with the reciprocal space map in GIDVis, as demonstrated for one sample from the tutorial. For the sample of a Naproxen thin film and the two following samples, however, no data are available for a comprehensive comparison with the associated reciprocal space maps.

With this fourth example, this chapter is complete. Application of the Indexing Routine and the accompanying memory usage did not lead to any overflow or computational restrictions whatsoever. If the program parameter are kept in the ranges as demonstrated, the program should not face any programmatic problems. Nevertheless, no claim is made to completeness. Neither should the impression arise that the program has no limitations or bottlenecks, nor that the obtained solutions are unambiguous.

Chapter 9

Summary

For indexing of grazing-incidence X-ray diffraction patterns, a semi-automated MATLAB routine is presented in this master thesis. A review of the available indexing methods and algorithms is elaborated and the fundamentals of X-ray diffraction are summarized in the beginning. Crystallographic conventions and basic terms are outlined for a comprehensive image. The methods necessary to obtain the GIXD data as well as the specular peak were introduced. A published formalism, based on the Laue condition for diffraction and its relation to reciprocal lattice vectors, is embedded in an algorithm to derive the six lattice constants a, b, c, α, β and γ from diffraction peak positions. This is achieved by dividing the process in smaller blocks, thereby gradually assigning Laue indices hkl to the experimentally derived data. The derived unit cell solutions are sorted and rated by deviation in the components of the scattering vector q as well as by crystallographic conventions. A special case, with the (001) lattice plane parallel to the substrate surface, as well as the general case (uvw) are considered and distinguished in the parallely implemented algorithm. The reduced form of the unit cell is derived and a numerical optimization process completes the routine. The indexing algorithm is complemented by a ready-to-use graphical user interface based on MATLAB source code. The applicability of the Indexing Routine is demonstrated by the means of a tutorial as well as on three different organic thin film samples.

References

[1] A. E. Watts et al. “Combining the Advantages of Powder X-ray Diffraction and NMR Crystallography in Structure Determination of the Pharmaceutical Material Cimetidine Hydrochloride”. In: *Crystal Growth and Design* 16.4 (2016), pp. 1798–1804. ISSN: 1528-7483.

[2] G.-C. Yuan et al. “Microstructure transformations induced by modified-layers on pentacene polymorphic films and their effect on performance of organic thin-film transistor”. eng. In: *Organic Electronics* 10.7 (2009), pp. 1388–1395. ISSN: 1566-1199.

[3] Y. Nakahara et al. “Ultra-thin films of polysilsesquioxanes possessing 3-methacryloxypropyl groups as gate insulator for organic field-effect transistors”. eng. In: *Thin Solid Films* 520.24 (2012), pp. 7195–7199. ISSN: 0040-6090.

[4] D. Braga et al. “Crystal Polymorphism and Multiple Crystal Forms”. In: *Structure and Bonding* 132 (2009), pp. 25–50.

[5] A. O. F. Jones et al. “Substrate-Induced and Thin-Film Phases: Polymorphism of Organic Materials on Surfaces”. In: *Advanced Functional Materials* 26.14 (2016), pp. 2233–2255. ISSN: 1616-301X.

[6] J. P. Glusker and K. N. Trueblood. “Crystal Structure Analysis: A Primer (Iucr Texts on Crystallography), Third Edition”. In: *Oxford University Press* (2010). ISSN: 978-0-19-957635-7.

[7] W. Demtröder. *Experimentalphysik 3*. Springer-Spektrum, 2015. ISBN: 978-3-662-49094-5. DOI: 10.1007/978-3-662-49094-5.

[8] D. W. Bennett. *Understanding single-crystal X-ray crystallography*. eng. Wiley-VCH, 2010. ISBN: 9783527326778.

[9] Q. Pan et al. “Comparative crystal structure determination of griseofulvin: Powder X-ray diffraction versus single-crystal X-ray diffraction”. In: *Chinese Science Bulletin* 57.30 (2012), pp. 3867–3871. ISSN: 1861-9541. DOI: 10.1007/s11434-012-5245-5.

[10] J. Simbrunner et al. “Indexing of grazing-incidence X-ray diffraction patterns: the case of fibre-textured thin films”. In: *Acta Crystallographica Section A* 74.4 (2018), pp. 373–387.

[11] J. Simbrunner et al. “Indexing grazing-incidence X-ray diffraction patterns of thin films: lattices of higher symmetry”. In: *Journal of Applied Crystallography* 52.2 (2019), pp. 428–439.

[12] J. Simbrunner et al. “An efficient method for indexing grazing-incidence X-ray diffraction data of epitaxially grown thin films”. In: *Acta Crystallographica Section A* 76.3 (2020), pp. 345–357.

[13] A. Pichler. “Master Thesis: Crystal Structure Solutions from Thin Films - Examples of Conjugated Molecules”. In: *Graz University of Technology* (2013).

[14] Wikipedia contributors. *Crystal structure — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Crystal_structure&oldid=964868454. [Online; accessed 11-June-2020]. 2020.

[15] S. Dag and L.-W. Wang. “Packing Structure of Poly(3-hexylthiophene) Crystal: Ab Initio and Molecular Dynamics Studies”. In: *The Journal of Physical Chemistry B* 114.18 (2010), pp. 5997–6000. ISSN: 1520-6106. DOI: 10.1021/jp1008219.

[16] Ch. Lercher et al. “Polymorphism of dioctyl-terthiophene within thin films: The role of the first monolayer”. In: *Chemical Physics Letters* 630 (2015), pp. 12–17. ISSN: 0009-2614.

[17] N. E. Widjonarko. “Introduction to Advanced X-ray Diffraction Techniques for Polymeric Thin Films”. In: *Coatings* 6.4 (2016). ISSN: 2079-6412. DOI: 10.3390/coatings6040054.

[18] H. R. Powell. “X-ray data processing”. eng. In: *Bioscience reports* 37.5 (2017), BSR20170227. ISSN: 1573-4935.

[19] Unknown Publisher. *Single-crystal XRD setup*. https://serc.carleton.edu/download/images/8399/BrukerK3_for_web.jpg. [Online; accessed 11-June-2020]. 2020.

[20] T. Oeser. *Methoden der Röntgenanalytik. In: Kristallstrukturanalyse durch Röntgenbeugung. essentials*. Springer Spektrum, Wiesbaden, 2019. ISBN: 978-3-658-25439-1.

[21] G. M. Sheldrick. "A short history of *SHELX*". In: *Acta Crystallographica Section A* 64.1 (2008), pp. 112–122.

[22] G. M. Sheldrick. "Crystal structure refinement with SHELXL". In: *Acta Crystallographica Section C* 71.1 (2015), pp. 3–8.

[23] George Sheldrick. *The SHELX homepage*. http://shelx.uni-goettingen.de/. [Online; accessed 19-July-2020]. 2019.

[24] A. Morawiec. "Ind_X: program for indexing single-crystal diffraction patterns". In: *Journal of Applied Crystallography* 50.2 (2017), pp. 647–650.

[25] M. J. Buerger. "Reduced Cells". In: *Zeitschrift fuer Kristallographie, Kristallgeometrie, Kristallphysik, Kristallchemie* 109 (1957), pp. 42–60.

[26] International Centre for Diffraction Data. *PDF-4*. https://www.icdd.com/. [Online; accessed 11-June-2020]. 2020.

[27] Speakman S. A. *Training to Become an Independent User of the X-Ray SEF at the Center for Materials Science and Engineering at MIT*. http://prism.mit.edu/xray. [Online; accessed 15-June-2020]. 2020.

[28] Unknown Publisher. *Diffraction pattern of a LaB6 reference powder specimen*. https://www.helmholtz-berlin.de/pubbin/igama_output?modus=einzel&sprache=en&gid=1972&typoid=68892. [Online; accessed 16-June-2020]. 2020.

[29] CCP14. *Available Software for Powder Diffraction Indexing including a Literature Search List*. http://www.ccp14.ac.uk/solution/indexing/. [Online; accessed 11-July-2020]. 2020.

[30] A. Boultif and D. Louër. "Powder pattern indexing with the dichotomy method". In: *Journal of Applied Crystallography* 37.5 (2004), pp. 724–731.

[31] A. A. Coelho. "Indexing of powder diffraction patterns by iterative use of singular value decomposition". In: *Journal of Applied Crystallography* 36.1 (2003), pp. 86–95.

[32] M.A. Neumann. "*X-CELL*: a novel indexing algorithm for routine tasks and difficult cases". In: *Journal of Applied Crystallography* 36.2 (2003), pp. 356–365.

[33] N. Bedoya-Martínez et al. “DFT-Assisted Polymorph Identification from Lattice Raman Fingerprinting”. In: *The Journal of Physical Chemistry Letters* 8.15 (2017), pp. 3690–3695.

[34] U. Shmueli (Editor). *International Tables for Crystallography, Vol. B: Reciprocal Space, 2nd ed.* Dordrecht: Springer, 2006.

[35] D. W. Breiby et al. “Simulating X-ray diffraction of textured films”. In: *Journal of Applied Crystallography* 41.2 (2008), pp. 262–271.

[36] A. K. Hailey et al. “*The Diffraction Pattern Calculator (DPC)* toolkit: a user-friendly approach to unit-cell lattice parameter identification of two-dimensional grazing-incidence wide-angle X-ray scattering data”. In: *Journal of Applied Crystallography* 47.6 (2014), pp. 2090–2099.

[37] Z. Jiang. “*GIXSGUI*: a MATLAB toolbox for grazing-incidence X-ray scattering data visualization and reduction, and indexing of buried three-dimensional periodic nanostructured films”. In: *Journal of Applied Crystallography* 48.3 (2015), pp. 917–926.

[38] V. Savikhin. “*GIWAXS-SIIRkit*: scattering intensity, indexing and refraction calculation toolkit for grazing-incidence wide-angle X-ray scattering of organic materials”. In: *Journal of Applied Crystallography* 53.4 (2020), pp. 1108–1129.

[39] B. Schrode et al. “*GIDVis*: a comprehensive software tool for geometry-independent grazing-incidence X-ray diffraction data analysis and pole-figure calculations”. In: *Journal of Applied Crystallography* 52.3 (2019), pp. 683–689.

[40] D. M. Smilgies and D. R. Blasini. “Indexation scheme for oriented molecular thin films studied with grazing-incidence reciprocal-space mapping”. In: *Journal of Applied Crystallography* 40.4 (2007), pp. 716–718.

[41] J. Perlich et al. “Grazing incidence wide angle x-ray scattering at the wiggler beamline BW4 of HASYLAB”. In: *Review of Scientific Instruments* 81.10 (2010), pp. 1798–1804.

[42] P. Niggli. *Handbuch der Experimentalphysik*. Leipzig: Akademische Verlagsgesellschaft, 1928.

[43] Z. Dauter and M. Jaskolski. “How to read (and understand) Volume A of International Tables for Crystallography: an introduction for nonspecialists”. In: *JAC* 43 (2010), pp. 1150–1171.

[44] T. Hahn Editor. *International Tables for Crystallography, Vol. A*. Springer, 2005. Chap. 9.2, pp. 750–755.

[45] B. Schrode. *Doctoral Thesis: Mapping Large Reciprocal-Space Volumes by Rotating Grazing Incidence X-ray Diffraction*. Graz University of Technology, 2020. Chap. 2, pp. 24–27.

[46] A. Santoro and A. D. Mighell. "Determination of reduced cells". In: *Acta Cryst.* 26 (1970), pp. 124–127.

[47] W. H. Miller. *A Treatise on Crystallography*. Cambridge, 1839.

[48] M. Birkholz. *Thin Film Analysis by X-Ray Scattering*. WILEY-VCH, 2006. Chap. 4, pp. 161–165.

[49] MATLAB. *version 9.7.0.1319299 (R2019b) Update 5*. Natick, Massachusetts: The MathWorks Inc., 2019.

[50] Inc. The MathWorks. *MATLAB Documentation - MathWorks Deutschland*. `https://de.mathworks.com/help/index.html`. [Online; accessed 21-September-2020]. 2020.

[51] Stefan Kowarik et al. "Dewetting of an Organic Semiconductor Thin Film Observed in Real-time". In: *Advanced Engineering Materials* 11.4 (2009), pp. 291–294. DOI: 10.1002/adem.200800289.

Appendices

Appendix A

Diffraction data for indexing

A.1 fIna-04

Table A.1: Used peak positions for indexing fIna-04

q_{xy} [Å^{-1}]	q_z [Å^{-1}]
0.00000000	0.71115071
0.00000000	1.42145995
0.43687415	1.77769534
0.44482484	1.06574358
0.54148678	2.13065968
0.54226213	0.71026371
0.54250230	1.42057630
0.70075570	1.42372840
0.70843278	0.71058747
0.86269051	2.86649740
0.88529146	1.77676139
0.89009197	2.13067395
0.89216683	0.70901601
0.89374256	1.06110155
0.89555585	1.41849222
1.08173690	1.42493430
1.29845207	0.71006017
1.40855533	1.06057100
1.41827354	0.71226618
1.42377737	1.75819980
1.47928481	2.46386187
1.51842944	1.42000181
1.52069691	0.70863122
1.63072165	0.70999558
1.77333930	0.72669570
1.77896341	1.42525922
2.16200556	0.71334928
2.28577544	0.71354625

A.2 Diindenoperylene

Table A.2: Used peak positions for indexing DIP

q_{xy} [Å^{-1}]	q_z [Å^{-1}]
0.000	1.776
0.368	0.091
0.396	1.160
0.428	1.069
0.689	0.984
0.734	0.182
0.806	1.232
1.140	0.171
1.223	0.083
1.529	0.790
1.625	0.697
2.274	0.953

A.3 Pentacenequinone

Table A.3: Used peak positions for indexing PQ

q_{xy} [Å^{-1}]	q_z [Å^{-1}]
0.0000	1.9460
0.4520	1.3982
0.4550	0.5461
0.7740	1.9962
0.7810	0.0559
0.8850	1.3422
0.9090	0.8521
0.9090	1.0892
0.9120	1.4512
0.9140	0.4901
1.1750	1.1432
1.1790	0.7981
1.2160	1.0342
1.2170	0.9081
1.3640	1.6322
1.3670	0.3090
1.5460	1.6862
1.5580	0.1096
1.5980	0.3620
1.6050	1.2872
1.6080	0.6521
1.6340	1.5062
1.6380	0.4361
1.7750	1.1972
1.7760	0.7441
1.8230	0.2359
1.9560	0.2920
2.0100	0.1818
2.1080	0.4201

A.4 Naproxen

Table A.4: Used peak positions for indexing Naproxen

q_{xy} [Å^{-1}]	q_z [Å^{-1}]
0.000	0.503
0.630	0.670
0.638	0.174
0.640	0.340
0.640	0.851
0.764	0.981
0.770	1.066
0.772	0.485
0.775	1.555
0.780	0.552
0.968	1.248
1.013	1.381
1.016	1.878
1.018	0.881
1.018	1.667
1.024	0.659
1.026	0.137
1.240	0.559
1.269	0.333
1.275	0.177
1.280	1.318
1.307	0.403
1.312	0.107
1.563	1.070
1.563	0.444
1.568	0.577
1.846	0.311
1.894	0.444
1.899	0.107

Appendix B

MATLAB® Source Code

B.1 Main program

```
%%% START %%%
%%% INITIALIZATION START
disp('>>>start')
redundanzhk=4;
lindep_limit = 0.01;
limit_fuer_hk_1=6;
limit_fuer_hk_2=8;
limit_fuer_l=8;
dqspec=0.01;
limit_fuer_uv=2;
wmax=limit_fuer_uv+1;
dqxy_cutoff=0.05;

filename='example_file';
warning off

UV_permutations=LPermutation(-limit_fuer_uv:limit_fuer_uv,2);
UV_permutations=[0 0]; % for case (001)

lines_to_permute=5;

[raw_data]=readmatrix(filename,'OutputType','double');

restriction_trigger=true;
a_lower_limit=5;
```

```
a_upper_limit=30;
b_lower_limit=5;
b_upper_limit=30;
gamma_lower_limit=60;
gamma_upper_limit=120;

[qspecs,datapoints,qxym,line_indices] = ...
    function_INITIALIZE_GUI(raw_data,lines_to_permute);
%%% INITIALIZATION END
%%% INDEXING PART I START
stop_trigger=false;
[names_w_rows,indexed] = ...
function_PART_ONE_PARALLEL_GUI_RESTRICTIONS(stop_trigger,...
restriction_trigger,redundanzhk,limit_fuer_hk_1,...
limit_fuer_hk_2,UV_permutations,qspecs,...
datapoints,qxym,dqxy_cutoff,...
a_lower_limit,a_upper_limit,b_lower_limit,b_upper_limit,...
gamma_lower_limit,gamma_upper_limit);
%%% INDEXING PART I END
area_parallelogram=names_w_rows(:,5).*names_w_rows(:,6)...
.*sind(names_w_rows(:,7));
[¬,ind_area]=sort(area_parallelogram,1);
names_ascending_area=names_w_rows(ind_area,:);
indices_ascending_area=indexed(:,:,ind_area);
c_lower_limit=10;
c_upper_limit=20;
alpha_lower_limit=60;
alpha_upper_limit=120;
beta_lower_limit=60;
beta_upper_limit=120;
volume_lower_limit=50;
volume_upper_limit=100000;
niggli_on=true;
qspecc=qspecs(1,2);
parameters_output=[];
indices_output=[];
max_solutions=20000;

%%% INDEXING PART II START
%%% POSTPROCESSING START
for possible_abg=[1:1000]

    aV7bg_fields_to_evaluate=...
```

```
69      indices_ascending_area(:,:,possible_abg);
70
71      [cell_parameters_output,indices_abg_KLR_NEW] = ...
72          function_PART_TWO_v12(qspecc,niggli_on,...
73          datapoints,redundanzhk,limit_fuer_l,wmax,...
74          aV7bg_fields_to_evaluate,dqspec,...
75          c_lower_limit,c_upper_limit,...
76          alpha_lower_limit,alpha_upper_limit,...
77          beta_lower_limit,beta_upper_limit,...
78          volume_lower_limit,volume_upper_limit);
79
80
81      parameters_output=[parameters_output;cell_parameters_output];
82      indices_output=cat(3,indices_output,indices_abg_KLR_NEW);
83
84      min_error=min(parameters_output(:,11))
85
86
87      [¬,ind_sort_sol]=sort(parameters_output(:,11));
88
89      parameters_output=parameters_output(ind_sort_sol,:);
90      indices_output=indices_output(:,:,ind_sort_sol);
91
92
93
94      if size(parameters_output,1)>max_solutions;
95          break
96      end
97
98  end
99  %%% INDEXING PART II END
100 %%% POSTPROCESSING END
101 %%% END %%%
```

B.2 Functions

B.2.1 function_INITIALIZE_GUI.m

```
1  function [qspecs,datapoints,qxym_matrix,uniqued_line_indices] = ...
2      function_INITIALIZE_GUI(raw_data,lines_to_permute)
```

```
3
4
5  disp('>>>load data file')
6  disp('>>>prepare initial values')
7  raw_data(all(¬raw_data,2),:)=[];
8  Messwerte_choice=raw_data(:,1:2);
9  [¬,idxmw] = sort(Messwerte_choice(:,1));
10 datapoints=Messwerte_choice(idxmw,:);
11 nzero=1;
12 qspecs=Messwerte_choice(sum(Messwerte_choice==0,2)...
13     ==nzero,:);
14 if isempty(qspecs)==1
15     error('INTERNAL ERROR: No specular data detected. Please ...
           provide specular data and restart script.')
16 else
17     disp('>>>specular data detected')
18 end
19
20 qcalc=(Messwerte_choice(:,1).^2+Messwerte_choice(:,2).^2).^(1/2);
21 MWundQ=horzcat(Messwerte_choice,qcalc);
22
23 tol_diff_qxym=0.0005;
24 tolconverted=tol_diff_qxym/max(abs(MWundQ(:,3)));
25 [¬,ia,ic]=uniquetol(MWundQ(:,3),tolconverted);
26 q_values_uniqued=MWundQ(ia,:);
27
28 [¬,iq]=sort(q_values_uniqued(:,3));
29 q_in_asc_order=q_values_uniqued(iq,:);
30
31 permi=1:lines_to_permute;
32 m_vec=LPermutation(permi,3);
33 m_vec(all(¬m_vec,2),:)=[];
34 m1=m_vec((m_vec(:,1)≠m_vec(:,2)),:);
35 m2=m1(m1(:,2)≠m1(:,3),:);
36 m3=m2(m2(:,1)≠m2(:,3),:);
37 msumsum=m3(:,1).^2+m3(:,2).^2+m3(:,3).^2;
38
39 new_arranged_m3=zeros(size(m3));
40
41 for msi=1:size(m3,1)
42     currentm3=m3(msi,:);
43     [¬,indm]=sort(currentm3,2);
44     new_arranged_m3(msi,:)=m3(msi,indm);
```

```
end
[uniqued_line_indices,¬,¬]=unique(new_arranged_m3,'rows','first');
mconc=horzcat(m3,msumsum);
[¬,ia,¬]=unique(mconc(:,4),'stable');
mn_index_qxym=mconc(ia,1:end-1);
q_in_asc_order(any(¬q_in_asc_order,2),:)=[];
m_poss_starter=q_in_asc_order(permi,1:2);
qxym_matrix=zeros(3,2*size(mn_index_qxym,1));
mn_index_qxym=uniqued_line_indices;
for m=1:size(mn_index_qxym,1)
    indexline=mn_index_qxym(m,:);
    kmu=2*m-1;
    kmo=2*m;
    qxym_matrix(:,kmu:kmo)=[m_poss_starter(indexline(3),:);...
        m_poss_starter(indexline(2),:);...
        m_poss_starter(indexline(1),:)];
end
disp('start generation of matrices')
end
```

B.2.2 function_LPermutation.m

```
function L = LPermutation(hkvalues,n)
ns = hkvalues(:);
n_numel = numel(hkvalues);
idx = (1:n_numel).';
rows = n_numel ^ n;
M = reshape((1:rows * n).', rows, n);
    for i = 1 : n
        hl_lines = ones(1, n_numel^(i - 1))
        cols = repmat(idx(:,hl_lines).', 1, n_numel ^ (n - i));
        M(:, i) = fliplr(cols(:));
    end
L = single(ns(M));
end
```

B.2.3 function_PART_ONE_PARALLEL _GUI_RESTRICTIONS.m

```
function [names,indexed] = ...
function_PART_ONE_PARALLEL_GUI_RESTRICTIONS(stop_trigger,...
restriction_trigger,redundanzhk,limit_fuer_hk_1,...
limit_fuer_hk_2,UV_permutations,qspecs,datapoints,...
qxym,dqxy_cutoff,a_lower_limit,a_upper_limit,...
b_lower_limit,b_upper_limit,...
gamma_lower_limit,gamma_upper_limit)

f = uifigure;
d = uiprogressdlg(f,'Title','Status Part I',...
    'Message','Opening the application');
HK_permutations_6er=LPermutation(-limit_fuer_hk_1...
:limit_fuer_hk_1,6);

[HK] = function_PERMVEC_NEW(limit_fuer_hk_2);


ABG_out=zeros(size(HK_permutations_6er,1),6,...
size(UV_permutations,1),'single');

size_of_ABG_12=size(ABG_out);
d.Value=.1;
d.Message = 'Initialize data points';
pause(2)

d.Value=.30;
d.Message='Create and solve possible matrices for a, b and ...
    gamma. Request parallel pool.'
pause(3)

if stop_trigger==true
    names=[];indexed=[];
    close(d)
    close(f)
    return
end
d.Value=.50;
d.Message='Calculate possible sets of [a,b,gamma]. Depending on ...
    your setting, this part could take longer. '
pause(2)
parfor uv_index=1:size(UV_permutations,1)

```

```
41      uv=UV_permutations(uv_index,:);
42      ABG_out(:,:,uv_index) = ...
            function_ABG_RESTRICTED(restriction_trigger,...
43          HK_permutations_6er,...
44          qspecs,uv,qxym,size_of_ABG_12,...
45          a_lower_limit,a_upper_limit,b_lower_limit,b_upper_limit,...
46          gamma_lower_limit,gamma_upper_limit);
47
48  end
49
50  valset=1;
51
52  if valset==1
53      d.Value=.80;
54      d.Message='ABG solutions obtained. Indexing qxy data.';
55      pause(2)
56      d.Message='ABG solutions obtained. Indexing qxy data..';
57      pause(2)
58      d.Message='ABG solutions obtained. Indexing qxy data...';
59  end
60
61  if stop_trigger==true
62      names=[];indexed=[];
63      close(d)
64      close(f)
65      return
66  end
67
68  names_w_zeros=reshape(permute(ABG_out,[1 3 2]),[],...
69      size(ABG_out,2),1);
70
71  idefix=find(¬all(names_w_zeros==0,2));
72  ABG_for_indexing=names_w_zeros(idefix,:);
73
74  [indexed_datapoints_red,label_of_indexed_datapoints_red] =...
75      function_NEWINDEXING_26(HK,ABG_for_indexing,...
76      datapoints,redundanzhk);
77  if stop_trigger==true
78      names=[];indexed=[];
79      close(d)
80      close(f)
81      return
82  end
```

```
83
84 d.Value=.95;
85 d.Message = 'Indexing part 1 finished. Calculate RMSD for qxy';
86 pause(2)
87
88 [names,indexed] = function_NEWRMSD_GXY(...
89     indexed_datapoints_red,label_of_indexed_datapoints_red,...
90     redundanzhk,dqxy_cutoff);
91
92 if stop_trigger==true
93     names=[];indexed=[];
94     close(d)
95     close(f)
96     return
97 end
98 d.Value=1;
99 d.Message = 'Indexing Part 1 finished. Solutions shown in ...
       Result Panel below';
100 close(d)
101 close(f)
102 end
```

B.2.4 function_PERMVEC_NEW

```
1  function [HK] = function_PERMVEC_NEW(limit_fuer_hk)
2  hklim=limit_fuer_hk;
3  hk=(-hklim:hklim).';
4  M=zeros(length(hk),2*length(hk));
5  M(:,2:2:end)=repmat(hk,1,length(hk));
6  M(:,1:2:end)=repmat(hk,1,length(hk)).';
7  Mtrans=M.';
8  P=zeros(2,length(hk)^2);
9  P(1,:)=Mtrans(1:2:end);
10 P(2,:)=Mtrans(2:2:end);
11 end
```

B.2.5 function_ABG_RESTRICTED.m

```
function [ABG_out] = ...
    function_ABG_RESTRICTED(restriction_trigger,...
    HK_permutations_6er,...
    qspecs,uv,qxym,size_of_ABG_12,...
    a_lower_limit,a_upper_limit,b_lower_limit,...
    b_upper_limit,gamma_lower_limit,gamma_upper_limit)

disp('current uv=')
display(uv)
u=uv(1,1);
v=uv(1,2);

qspec=qspecs(1,2);

[XYZ]=function_XYZ(function_NEWMATRIXFILLER(HK_permutations_6er,...
    u,v,qxym,qspec));


[C,¬,¬]=unique(XYZ,'rows','first');

C(C(:,1)<0,:)=[];
C(C(:,2)<0,:)=[];
C(all(¬C,2),:)=[];

disp('compute ABG out of XYZ sets for')
display(uv)

if isempty(C)==true
    C=zeros(1,3);
end

[ABG_out] = ...
    function_CALC_ABGfromXYZ_RESTRICTED_v7(restriction_trigger,...
    C(:,1),C(:,2),C(:,3)...
    ,u,v,qspec,size_of_ABG_12,...
    a_lower_limit,a_upper_limit,b_lower_limit,...
    b_upper_limit,gamma_lower_limit,gamma_upper_limit);

end
```

B.2.6 function_NEWMATRIXFILLER.m

```
function [FMATRICES] = ...
    function_NEWMATRIXFILLER(hk_matrix_permutions,...
     u,v,qxym,qspec)

FMpart1=zeros(3,3,size(hk_matrix_permutions,1));
FMpart2=zeros(size(FMpart1,1),size(FMpart1,2),...
     size(FMpart1,3),size(qxym,2)/2);
numer=1;

for indcol=1:2:size(qxym,2)
     qxy1=qxym(1,indcol); qz1=qxym(1,indcol+1);
     qxy2=qxym(2,indcol); qz2=qxym(2,indcol+1);
     qxy3=qxym(3,indcol); qz3=qxym(3,indcol+1);

     for nrp=1:size(hk_matrix_permutions,1)
          h1=hk_matrix_permutions(nrp,1);
          k1=hk_matrix_permutions(nrp,2);
          h2=hk_matrix_permutions(nrp,3);
          k2=hk_matrix_permutions(nrp,4);
          h3=hk_matrix_permutions(nrp,5);
          k3=hk_matrix_permutions(nrp,6);

          position1=(qxy1.^-2).*(h1-((u.*qz1)./qspec)).^2+...
          u.^2/qspec.^2;
          position2=(qxy2.^-2).*(h2-((u.*qz2)./qspec)).^2+...
          u.^2/qspec.^2;
          position3=(qxy3.^-2).*(h3-((u.*qz3)./qspec)).^2+...
          u.^2/qspec.^2;

          position4=(qxy1.^-2).*(k1-((v.*qz1)./qspec)).^2+...
          v.^2/qspec.^2;
          position5=(qxy2.^-2).*(k2-((v.*qz2)./qspec)).^2+...
          v.^2/qspec.^2;
          position6=(qxy3.^-2).*(k3-((v.*qz3)./qspec)).^2+...
          v.^2/qspec.^2;

          position7=-2.*((qxy1.^-2).*(h1-((u.*qz1)./qspec)).*(k1...
          -(v.*qz1./qspec))...
              +(u.*v)./qspec.^2);

          position8=-2.*((qxy2.^-2).*(h2-((u.*qz2)./qspec)).*(k2...
          -(v.*qz2./qspec))...
```

```
            +(u.*v)./qspec.^2);

        position9=-2.*((qxy3.^-2).*(h3-((u.*qz3)./qspec)).*(k3...
        -(v.*qz3./qspec))...
            +(u.*v)./qspec.^2);

        FMpart1(:,:,nrp)=[position1,position4,position7;...
        position2,position5,position8;...
        position3,position6,position9];
    end

    FMpart2(:,:,:,numer)=FMpart1;
    FMATRICES=reshape(permute(FMpart2,[1 2 3 4]),...
        [3,3,size(FMpart2,3)*size(FMpart2,4)]);

    numer=numer+1;
end

end
```

B.2.7 function_XYZ.m

```
function [XYZ] = function_XYZ(FMATRICES)
    XYZ = zeros(size(FMATRICES,3),3,'single');
    for i=1:size(FMATRICES,3)
        det_test=det(FMATRICES(:,:,i));
        if abs(det_test)>=1e-9
            XYZ(i,:) = FMATRICES(:,:,i)\ones(3,1);
        end
    end
end
```

B.2.8 function_CALC_ABGfromXYZ_RESTRICTED_v7.m

```
function [standard_ABG_out] = ...
    function_CALC_ABGfromXYZ_RESTRICTED_v7(restriction_trigger,...
    A,B,C,u,v,qspec,size_of_ABG_12,...
    a_lower_limit,a_upper_limit,b_lower_limit,...
```

```
    b_upper_limit,gamma_lower_limit,gamma_upper_limit)

standard_ABG_out=zeros(size_of_ABG_12(1),...
    size_of_ABG_12(2),'single');

LAMBDA=(A.*B-C.^2)./(qspec.^2-u.^2.*A-v.^2.*B+2.*u.*v.*C);

za=real(sqrt(A+v.^2.*LAMBDA));
zb=real(sqrt(B+u.^2.*LAMBDA));

cosg=(C+u.*v.*LAMBDA)./(za.*zb);
gamma=acosd(cosg);

a=(2.*pi)./(za.*sind(gamma));
b=(2.*pi)./(zb.*sind(gamma));

u_v_qspec_a_b_gamma=real([ones(size(A,1),...
    1).*u,ones(size(A,1),1).*v,
    ones(size(A,1),1).*qspec,a,b,gamma]);

u_v_qspec_a_b_gamma(any(isnan(u_v_qspec_a_b_gamma), 2), :) = [];

u_v_qspec_a_b_gamma(u_v_qspec_a_b_gamma(:,4)⩽0,:)=[];
u_v_qspec_a_b_gamma(u_v_qspec_a_b_gamma(:,5)⩽0,:)=[];

u_v_qspec_a_b_gamma(all(¬u_v_qspec_a_b_gamma,2),:)=[];
u_v_qspec_a_b_gamma(any(isinf(u_v_qspec_a_b_gamma),2),:) = [];

ind_unten=u_v_qspec_a_b_gamma(:,end)⩾60;
sotti1=u_v_qspec_a_b_gamma(ind_unten,:);
ind_oben=sotti1(:,end)⩽120;
ABG_before_restrictions=sotti1(ind_oben,:);

switch(restriction_trigger)

    case true
        aind=ABG_before_restrictions(:,end-2)⩾a_lower_limit & ...
            ABG_before_restrictions(:,end-2)⩽a_upper_limit;
        abg_after_a=ABG_before_restrictions(aind,:);

        bind=abg_after_a(:,end-1)⩾b_lower_limit & ...
            abg_after_a(:,end-1)⩽b_upper_limit;
        abg_after_b=abg_after_a(bind,:);
```

```
47
48          gammaind=abg_after_b(:,end)≥gamma_lower_limit & ...
49              abg_after_b(:,end)≤gamma_upper_limit;
50          ABG=abg_after_b(gammaind,:);
51
52      case false
53          ABG=ABG_before_restrictions;
54  end
55
56  [ABG1,¬,¬]=unique(ABG,'rows','first');
57
58  [ABG_out_checkpoint_1] = function_SUB2_CONDITION(ABG1,qspec);
59
60  [ABG_out] = function_ABGNIGGLI(ABG_out_checkpoint_1);
61
62  if isempty(ABG_out)==true
63      standard_ABG_out=zeros(size_of_ABG_12(1),...
64          size_of_ABG_12(2),'single');
65      disp("ABG_out empty")
66  else
67
68
69      tol_diff_a=0.15;
70      tol_diff_b=0.15;
71      tol_diff_gamma=0.3;
72
73      tolconverteda=tol_diff_a/max(abs(ABG_out(:,4)));
74      [¬,¬,ia3]=uniquetol(ABG_out(:,4),tolconverteda);
75
76      tolconvertedb=tol_diff_b/max(abs(ABG_out(:,5)));
77      [¬,¬,ib3]=uniquetol(ABG_out(:,5),tolconvertedb);
78
79      tolconvertedg=tol_diff_gamma/max(abs(ABG_out(:,6)));
80      [¬,¬,ig3]=uniquetol(ABG_out(:,6),tolconvertedg);
81
82      inditest=horzcat(ia3,ib3,ig3);
83      [¬,ia]=unique(inditest,'rows','first');
84      abglistuniqued=ABG_out(ia,:);
85
86
87      standard_ABG_out(1:size(abglistuniqued,1),:)=abglistuniqued;
88  end
89  end
```

B.2.9 function_SUB2_CONDITION.m

```
function [ABG_out_checkpoint_1] = ...
    function_SUB2_CONDITION(ABG,qspec)

limit_for_zero=1e-5;
za=(2*pi)./(ABG(:,end-2).*sind(ABG(:,end)));
zb=(2*pi)./(ABG(:,end-1).*sind(ABG(:,end)));
u=ABG(:,1);
v=ABG(:,2);
cosg=cosd(ABG(:,end));
argum=qspec.^2-u.^2.*za.^2-v.^2.*zb.^2+2.*u.*v.*za.*zb.*cosg;
condi=argum>limit_for_zero;
ABG_out_checkpoint_1=ABG(condi,:);

end
```

B.2.10 function_ABGNIGGLI.m

```
function [ABG_out] = function_ABGNIGGLI(ABG_in)

epsilon=1e-1;
indexabg=ABG_in(:,end-2).^2⩽(ABG_in(:,end-1)+epsilon).^2;
abg_agreaterb=ABG_in(indexabg,:);

ind_gamma_kl_90=abg_agreaterb(:,end)<90;
abg_gamma_kl_90=abg_agreaterb(ind_gamma_kl_90,:);

bcosgamma=abg_gamma_kl_90(:,end-1).*cosd(abg_gamma_kl_90(:,end));
ahalbe=abg_gamma_kl_90(:,end-2)./2;
index_true_kl_90=bcosgamma(:)⩽ahalbe(:);
rest_nach_niggli1=abg_gamma_kl_90(index_true_kl_90,:);

ind_gamma_gr_90=abg_agreaterb(:,end)⩾90;
abg_gamma_gr_90=abg_agreaterb(ind_gamma_gr_90,:);

```

```
bcosgamma2=abg_gamma_gr_90(:,end-1).*...
abs(cosd(abg_gamma_gr_90(:,end)));
ahalbe2=abg_gamma_gr_90(:,end-2)./2;
index_true_gr_902=bcosgamma2(:)≤ahalbe2(:);
rest_nach_niggli2=abg_gamma_gr_90(index_true_gr_902,:);

ABG_out=vertcat(rest_nach_niggli1,rest_nach_niggli2);

end
```

B.2.11 function_PART_TWO_v12.m

```
function [final_parameters_output,final_indices_output] = ...
    function_PART_TWO_v12(qspecc,niggli_on,datapoints,redundanzhk,...
    limit_fuer_l,wmax,best_abg_indexed,dqspec,...
    c_lower_limit,c_upper_limit,...
    alpha_lower_limit,alpha_upper_limit,beta_lower_limit,...
    beta_upper_limit,...
    volume_lower_limit,volume_upper_limit)

f = uifigure;
d = uiprogressdlg(f,'Title','Status Part 2',...
    'Message','Opening the application');

qxyz_datapoints=(best_abg_indexed(2:end,1,1).^2+...
    best_abg_indexed(2:end,2,1).^2).^(1/2);

lvec=-limit_fuer_l:limit_fuer_l;
ML=LPermutation(lvec,3);

warning off

[INDEXED_with_subs]=function_CALCSUBSIS(best_abg_indexed);

[INDEXED_with_subs_NEW] = ...
    function_NEWPACKING(INDEXED_with_subs,redundanzhk);
clear INDEXED_with_subs

hk_aus_qz=LPermutation(1:redundanzhk,3);

```

```
29 selection_hohe=size(INDEXED_with_subs_NEW(2:redundanzhk:end,1,1),1);
30
31 n_l=numel(lvec);
32 w_vec=single(-wmax:wmax);
33 n_w=size(w_vec,2);
34 nr_of_initial_matrices=single(redundanzhk.^3.*n_l.^3.*n_w);
35 initial_matrices=zeros(4,4,nr_of_initial_matrices,'single');
36
37 output_matrices_for_solve=repmat(initial_matrices,[1 1 1 ...
38         size(INDEXED_with_subs_NEW,3)]);
39
40 d.Value=.2;
41 d.Message = 'Assemble matrices for linear system of equations';
42 pause(1)
43 for ind_abgpage=1:size(INDEXED_with_subs_NEW,3)
44
45     abg_page=INDEXED_with_subs_NEW(:,:,ind_abgpage);
46
47     output_matrices_for_solve(1,1,:,ind_abgpage)=...
48         ones(1,1,size(initial_matrices,3),'single').*abg_page(1,1);
49
50     output_matrices_for_solve(1,2,:,ind_abgpage)=...
51         ones(1,1,size(initial_matrices,3),'single').*abg_page(1,2);
52
53     output_matrices_for_solve(1,4,:,ind_abgpage)=...
54         ones(1,1,size(initial_matrices,3),'single').*abg_page(1,8);
55
56     select_ausgabe_mw=abg_page(1:selection_hohe,11:end);
57     qzi_values_unsorted=select_ausgabe_mw(:,2);
58     [¬,idx_qzi]=sort(qzi_values_unsorted,1);
59     qzi_sorted=select_ausgabe_mw(idx_qzi,:);
60
61     tol_diff_qz=0.01;
62     tolconverted=tol_diff_qz/max(abs(qzi_sorted(:,2)));
63     [¬,ia,¬]=uniquetol(qzi_sorted(:,2),tolconverted);
64     qz_for_KLR_matrix=qzi_sorted(ia,:);
65
66     welches_qz=[1;2;3];
67
68     qz123_cell_ges_loop=cell(size(welches_qz,1),redundanzhk);
69     qz_sortiert_nur_indi=qz_for_KLR_matrix(1:3,3:end);
70     qzi_sel=qz_for_KLR_matrix(1:3,2);
71     qzi_abglwich=abg_page(1:end,2);
```

```
72
73      qz1_index=qzi_abglwich==qzi_sel(1,:);
74      lineqz1=abg_page(qz1_index,6);
75      qz2_index=qzi_abglwich==qzi_sel(2,:);
76      lineqz2=abg_page(qz2_index,6);
77      qz3_index=qzi_abglwich==qzi_sel(3,:);
78      lineqz3=abg_page(qz3_index,6);
79
80      qziterme_rowvec=vertcat(lineqz1,lineqz2,lineqz3);
81      placeholder_qzihki=zeros(size(qz_sortiert_nur_indi));
82
83      takeval=1;
84      for hkrow=1:3
85          st=1;
86          for hkcol=1:redundanzhk
87              placeholder_qzihki(hkrow,st:st+2)=...
88              qziterme_rowvec(takeval,:);
89              st=st+3;
90              takeval=takeval+1;
91          end
92      end
93
94      qzis_choice=placeholder_qzihki(:,1:3:end);
95
96      for lnz=1:3
97          lenzhk=1;
98          for rhks=1:redundanzhk
99              qz123_cell_ges_loop(lnz,rhks)=...
100             {qz_sortiert_nur_indi(lnz,lenzhk:lenzhk+1)};
101             lenzhk=lenzhk+3;
102         end
103     end
104
105     hk_cell_matrices=zeros(3,4,size(hk_aus_qz,1),'single');
106
107     for poss_hk=1:size(hk_aus_qz,1)
108
109
110         ind_hk=hk_aus_qz(poss_hk,:);
111
112         hk_cell_matrices(1,1:2,poss_hk)=...
113         qz123_cell_ges_loop{1,ind_hk(1)};
114         hk_cell_matrices(1,4,poss_hk)=...
```

```
115         qzis_choice(1,ind_hk(1));
116         hk_cell_matrices(2,1:2,poss_hk)=...
117         qz123_cell_ges_loop{2,ind_hk(2)};
118         hk_cell_matrices(2,4,poss_hk)=...
119         qzis_choice(2,ind_hk(2));
120         hk_cell_matrices(3,1:2,poss_hk)=...
121         qz123_cell_ges_loop{3,ind_hk(3)};
122         hk_cell_matrices(3,4,poss_hk)=...
123         qzis_choice(3,ind_hk(3));
124
125     end
126
127     output_matrices_for_solve(2:end,:,:,ind_abgpage)=...
128         repmat(repmat(hk_cell_matrices,[1 1 n_l.^3]),[1 1 n_w]);
129 end
130
131 size_same_w=redundanzhk.^3.*n_l.^3;
132 w_arrow=reshape(repmat(w_vec,[size_same_w 1 ...
      1]),nr_of_initial_matrices,1);
133 w_arrow_kloo=repmat(w_arrow,[1 1 ...
      size(output_matrices_for_solve,4)]);
134 output_matrices_for_solve(1,3,:,:)=w_arrow_kloo;
135
136 ml=ML';
137 ml_expanded=repmat(ml,[redundanzhk^3 1]);
138 l_series_same_w=reshape(ml_expanded,[3 1 size_same_w]);
139 l_series_expanded_for_w=repmat(l_series_same_w, [1 1 n_w]);
140
141 l_series_expanded_for_w_kloo=repmat(l_series_expanded_for_w,...
142     [1 1 size(output_matrices_for_solve,4)]);
143 output_matrices_for_solve(2:end,3,:)=l_series_expanded_for_w_kloo;
144 disp('KLR Matrices preparated. Calculate Matrice divisions')
145
146 d.Value=.4;
147 d.Message = 'Compute overdeterminded systems. Depending on your ...
      setting, this part could take longer.';
148 pause(2)
149 d.Value=.5;
150 d.Message = 'Compute overdeterminded systems. Depending on your ...
      setting, this part could take longer.';
151 pause(2)
152 d.Value=.6;
```

```
153 d.Message = 'Compute overdeterminded systems. Depending on your ...
      setting, this part could take longer.';
154 pause(2)
155
156 klr_for_all_abg=zeros(7,size(output_matrices_for_solve,3),...
157     size(output_matrices_for_solve,4),'single');
158
159 switch niggli_on
160     case false
161
162     for all_abg=1:size(klr_for_all_abg,3)
163         disp('Niggli criteria switched off')
164         klr=zeros(7,size(output_matrices_for_solve,3),'single');
165         abg_aktuell=INDEXED_with_subs_NEW(1,4:6,all_abg);
166         for i=1:size(output_matrices_for_solve,3)
167
168             working_mat=output_matrices_for_solve(:,:,i,all_abg);
169
170             if working_mat(1,1:3)==zeros(1,3)
171                 klr(1:end,i)=zeros(size(klr,1),1);
172
173             else
174
175                 KLR2check=working_mat(:,1:3)\...
176                     working_mat(:,end);
177
178                 deld=-1./(2.*pi).*KLR2check(1)...
179                 .*abg_aktuell(1).*sind(abg_aktuell(3));
180                 mu=-1./(2.*pi).*KLR2check(2).*...
181                 abg_aktuell(2).*sind(abg_aktuell(3));
182                 sin_eps=sind(abg_aktuell(3))...
183                 ./sqrt(sind(abg_aktuell(3)).^2...
184                 +deld.^2+mu.^2+...
185                     2.*deld.*mu.*..
186                     cosd(abg_aktuell(3)));
187
188                 alpha2check=acosd((sin_eps...
189                 .*mu+sin_eps.*deld.*...
190                 cosd(abg_aktuell(3)))...
191                 ./sind(abg_aktuell(3)));
192                 beta2check=acosd(...
193                 (sin_eps.*deld+sin_eps...
194                 .*mu.*cosd(abg_aktuell(3)))...
```

```
195                 ./sind(abg_aktuell(3)));
196                 c2check=(2.*pi)./...
197                 (sin_eps.*KLR2check(3));
198
199                 test_c = c2check < c_lower_limit || c2check > ...
                        c_upper_limit;
200
201                 if test_c == true
202                     continue
203                 else
204                     test_alpha = alpha2check(1) > ...
                            alpha_upper_limit || alpha2check < ...
                            alpha_lower_limit;
205                     if test_alpha==true
206                         continue
207                     else
208                         test_beta = beta2check > ...
                                beta_upper_limit || beta2check < ...
                                beta_lower_limit;
209                         if test_beta==true
210                             continue
211                         else
212                             volume2check = abg_aktuell(1).*...
213                                 abg_aktuell(2).*c2check.*...
214                                 (1-cosd(alpha2check).*...
215                                 cosd(alpha2check)-...
216                                 cosd(beta2check)...
217                                 .*cosd(beta2check)-...
218                                 cosd(abg_aktuell(3)).*...
219                                 cosd(abg_aktuell(3))...
220                                 +2.*cosd(alpha2check).*...
221                                 cosd(beta2check).*...
222                                 cosd(abg_aktuell(3))).^(1/2);
223                             test_vol = volume2check > ...
                                    volume_upper_limit ...
224                                     || volume2check < ...
                                        volume_lower_limit;
225                             if test_vol == true
226                                 continue
227                             else
228                                 klr(1:3,i) = KLR2check;
229                                 klr(4:6,i) = working_mat(1,1:3)';
230                                 klr(7,i)=...
```

```
                                    (working_mat(1,1)*klr(1,i)+...
                                        working_mat(1,2)*klr(2,i)+...
                                        working_mat(1,3)*klr(3,i))^2;
                                end
                            end
                        end
                    end
                end
            end
            klr_for_all_abg(:,:,all_abg)=klr;

    end

    case true
        disp('Niggli criteria switched on')
% ALIGNMENT RULES CANCELLED
for all_abg=1:size(klr_for_all_abg,3)
klr=zeros(7,size(output_matrices_for_solve,3),'single');
abg_aktuell=INDEXED_with_subs_NEW(1,4:6,all_abg);
for i=1:size(output_matrices_for_solve,3)
working_mat=...
output_matrices_for_solve(:,:,i,all_abg);
if working_mat(1,1:3)==zeros(1,3)
klr(1:end,i)=zeros(size(klr,1),1);
else
KLR2check=working_mat(:,1:3)\...
working_mat(:,end);
deld=-1./(2.*pi).*KLR2check(1).*...
abg_aktuell(1).*sind(abg_aktuell(3));
mu=-1./(2.*pi).*KLR2check(2).*...
abg_aktuell(2)...
.*sind(abg_aktuell(3));
sin_eps=sind(abg_aktuell(3))./sqrt(...
sind(abg_aktuell(3)).^2+deld.^2+mu.^2+...
2.*deld.*mu.*cosd(abg_aktuell(3)));

alpha2check=acosd((sin_eps.*mu+sin_eps.*...
deld.*cosd(abg_aktuell(3)))./...
sind(abg_aktuell(3)));
beta2check=acosd((sin_eps.*deld+sin_eps.*...
mu.*cosd(abg_aktuell(3)))./...
sind(abg_aktuell(3)));
c2check=(2.*pi)./(sin_eps.*KLR2check(3));
```

```
274
275 test_c = c2check < c_lower_limit || c2check > c_upper_limit;
276 if test_c == true
277 continue
278 else
279 test_alpha = alpha2check > alpha_upper_limit || alpha2check < ...
       alpha_lower_limit;
280 if test_alpha==true
281 continue
282 else
283 test_beta = beta2check > beta_upper_limit || beta2check < ...
       beta_lower_limit;
284 if test_beta==true
285 continue
286 else
287 test_b_gt_c = abg_aktuell(2)^2 ⩽ c2check^2;
288 if test_b_gt_c == false
289 continue
290 else
291 volume2check = abg_aktuell(1).*abg_aktuell(2).*c2check.*...
292 (1-cosd(alpha2check).*cosd(alpha2check)-cosd(beta2check)...
293 .*cosd(beta2check)-cosd(abg_aktuell(3)).*...
294 cosd(abg_aktuell(3))+2.*cosd(alpha2check).*cosd(beta2check).*...
295 cosd(abg_aktuell(3))).^(1/2);
296 test_vol = volume2check > volume_upper_limit || volume2check < ...
       volume_lower_limit;
297 if test_vol == true
298 continue
299 else
300 test_type_1= lt(alpha2check,91) && lt(beta2check,91) && ...
       lt(abg_aktuell(3),91);
301 if test_type_1 == true
302 type1_1 = c2check*cosd(alpha2check) ⩽ abg_aktuell(2)/2;
303 if type1_1==false
304 continue
305 else
306 type1_2 = c2check*cosd(beta2check) ⩽ abg_aktuell(1)/2;
307 if type1_2==false
308 continue
309 else
310 klr(1:3,i) = KLR2check;
311 klr(4:6,i) = working_mat(1,1:3)';
312 klr(7,i)=(working_mat(1,1)*klr(1,i)+...
```

```
313 working_mat(1,2)*klr(2,i)+...
314 working_mat(1,3)*klr(3,i))^2;
315 end
316 end
317 else
318 test_type_2= gt(alpha2check,89) && gt(beta2check,89) && ...
       gt(abg_aktuell(3),89);
319 if test_type_2 == false
320 continue
321 else
322 type2_1 = c2check*abs(cosd(alpha2check)) ⩽ abg_aktuell(2)/2;
323 if type2_1 == false
324 continue
325 else
326 type2_2 = c2check*abs(cosd(beta2check)) ⩽ abg_aktuell(1)/2;
327 if type2_2==false
328 continue
329 else
330 type2_3=abg_aktuell(2)*c2check*...
331 abs(cosd(alpha2check))+...
332 abg_aktuell(1)*c2check*...
333 abs(cosd(beta2check))+...
334 abg_aktuell(1)*abg_aktuell(1)*...
335 abs(cosd(abg_aktuell(3))) ⩽...
336 (abg_aktuell(1)^2+abg_aktuell(2)^2)/2;
337 if type2_3 == false
338 continue
339 else
340 klr(1:3,i) = KLR2check;
341 klr(4:6,i) = working_mat(1,1:3)';
342 klr(7,i)=(working_mat(1,1)*klr(1,i)+...
343 working_mat(1,2)*klr(2,i)+...
344 working_mat(1,3)*klr(3,i))^2;
345 end
346 end
347 end
348 end
349 end
350 end
351 end
352 end
353 end
354 end
```

```
end
end
klr_for_all_abg(:,:,all_abg)=klr;
end
end

clear output_matrices_for_solve
OUTPUT_COLLECTION_parameter=...
zeros(round(size(klr_for_all_abg,2)/10,0),13,...
    size(klr_for_all_abg,3),'single');
OUTPUT_COLLECTION_indexed_data_NEW=...
zeros(size(datapoints,1),5,...
    round(size(klr_for_all_abg,2)/10,0),...
    size(klr_for_all_abg,3),'single');

disp('Start indxing part 2')
d.Value=.7;
d.Message = 'Busy with indexing part 2';
pause(2)
d.Message = 'Busy with indexing part 2 .';
pause(2)
d.Message = 'Busy with indexing part 2 . .';
pause(2)
d.Message = 'Busy with indexing part 2 . . .';

for abg_i=1:size(klr_for_all_abg,3)

    choose_abg=INDEXED_with_subs_NEW(:,:,abg_i);
    sub2=choose_abg(1,9);
    sub3=choose_abg(2:end,7);

    za_i=(2*pi)./(choose_abg(1,4).*sind(choose_abg(1,6)));
    zb_i=(2*pi)./(choose_abg(1,5).*sind(choose_abg(1,6)));
    cosg_i=cosd(choose_abg(1,6));

    qspec=choose_abg(1,3);
    eins_d_qspec=1/choose_abg(1,3);

    qzi_list=choose_abg(2:end,1:6);

    KLR_raw_us=klr_for_all_abg(:,:,abg_i)';
    KLR_raw_us(all(¬KLR_raw_us,2),:)=[];

```

```
398     KLR_raw_us(abs(KLR_raw_us(:,3))<1e-4,:)=[];
399
400     gspeccalc=sqrt(choose_abg(1,10)+KLR_raw_us(:,7));
401     ind_qspec=abs(gspeccalc-qspec)⩽dqspec;
402     KLR_raw=KLR_raw_us(ind_qspec,:);
403     KLR_raw(:,8)=abs(gspeccalc(ind_qspec,:)-qspec);
404
405     colwid=size(qzi_list,2)+size(KLR_raw,2)+2;
406     field=zeros(size(qzi_list,1),colwid,size(KLR_raw,1),'single');
407
408     for klr_line=1:size(KLR_raw,1)
409         expanded_klr=repmat(KLR_raw(klr_line,:),...
410         [size(qzi_list,1) 1]);
411         field(:,1:colwid-3,klr_line)=...
412         horzcat(qzi_list,expanded_klr(:,1:7));
413         field(:,17,klr_line)=expanded_klr(:,8);
414     end
415
416     l_calculated=(field(:,6,:)-field(:,3,:).*...
417     field(:,7,:)-field(:,4,:).*field(:,8,:)).*...
418         field(:,9,:).^-1;
419     l_tolzero=1e-4;
420     l_calculated(abs(l_calculated)⩽l_tolzero)=0;
421     l_calculated_round=round(l_calculated,0);
422     field(1:end,14,:)=l_calculated_round;
423     klr_hkl_term=field(:,3,:).*field(:,7,:)+field(:,4,:)...
424     .*field(:,8,:)+...
425         field(:,14,:).*field(:,9,:);
426     gzi=(klr_hkl_term.*sub2+sub3).*eins_d_qspec;
427     qz_diff=(field(:,2,:)-gzi).^2;
428     field(:,15,:)=qz_diff;
429
430     pt1=field(:,3,:).^2.*za_i.^2+field(:,4,:).^2.*zb_i.^2-2...
431     .*field(:,3,:).*...
432         field(:,4,:).*za_i.*zb_i.*cosg_i;
433
434     gxyzi=sqrt(pt1+((gzi.*qspec-sub3).^2)./(sub2.^2));
435     ▵_qxyz=(gxyzi-qxyz_datapoints).^2;
436     field(:,16,:)=▵_qxyz;
437
438     nsize4=size(field,1)/redundanzhk;
439     output_abg_KLR_all=zeros(nsize4,size(field,2),...
440     size(field,3),'single');
```

```
441     indzaehler=1;
442
443      for indvierer=1:redundanzhk:size(field,1)-redundanzhk+1
444
445         qz_4er_mit_indizes=...
446         field(indvierer:indvierer+redundanzhk-1,:,:);
447         qz_4er_nur_diff_qxyz=qz_4er_mit_indizes(:,16,:);
448         [¬,ind_qmin]=mink(qz_4er_nur_diff_qxyz,1);
449         d1=reshape(ind_qmin,[size(ind_qmin,3) 1]);
450
451         for klo=1:size(ind_qmin,3)
452             output_abg_KLR_all(indzaehler,:,klo)=...
453                 qz_4er_mit_indizes(d1(klo),:,klo);
454         end
455         indzaehler=indzaehler+1;
456
457      end
458
459     clear INDEXED_with_subs_NEW
460     summieren_qxy=sum(output_abg_KLR_all(:,5,:));
461     summieren_qz=sum(output_abg_KLR_all(:,15,:));
462     summieren_qxyz=sum(output_abg_KLR_all(:,16,:));
463     N_anzahl=size(output_abg_KLR_all,1);
464     RMSD_qxy=(summieren_qxy./N_anzahl).^(1/2);
465     RMSD_qz=(summieren_qz./N_anzahl).^(1/2);
466     RMSD_qxyz=(summieren_qxyz./N_anzahl).^(1/2);
467     size(output_abg_KLR_all);
468     RMSD_qspec=output_abg_KLR_all(1,end,:);
469     outputformat_abg_i=...
470     zeros(1,13,size(output_abg_KLR_all,3),'single');
471     outputformat_abg_i(:,1:3,:)=repmat(choose_abg(1,4:6),[1 1 ...
           size(output_abg_KLR_all,3)]);
472     outputformat_abg_i(:,4:9,:)=output_abg_KLR_all(1,[7 8 9 10 ...
           11 12],:);
473     outputformat_abg_i(:,10,:)=RMSD_qxy;
474     outputformat_abg_i(:,11,:)=RMSD_qz;
475     outputformat_abg_i(:,12,:)=RMSD_qxyz;
476     outputformat_abg_i(:,13,:)=RMSD_qspec;
477     output_abg_i_with_errors_par=...
478     reshape(permute(outputformat_abg_i,[3 2 1]),...
479         [size(outputformat_abg_i,3),size(outputformat_abg_i,2)]);
480
481
```

```
482     OUTPUT_COLLECTION_parameter(1:size(KLR_raw,1),:,abg_i)=...
483     output_abg_i_with_errors_par;
484     OUTPUT_COLLECTION_indexed_data_NEW...
485     (:,:,1:size(KLR_raw,1),abg_i)=...
486         output_abg_KLR_all(:,[1 2 3 4 14],:,:);
487
488 end
489 disp('indxing part 2 finished')
490 clear klr_for_all_abg
491 parameter_abg_KLR=reshape(permute(OUTPUT_COLLECTION_parameter,[1 ...
        3 2]),[],...
492     size(OUTPUT_COLLECTION_parameter,2),1);
493 clear OUTPUT_COLLECTION_parameter
494 indices_abg_KLR_NEW_unsorted=...
495 reshape(OUTPUT_COLLECTION_indexed_data_NEW...
496     ,[size(OUTPUT_COLLECTION_indexed_data_NEW,1),...
497     size(OUTPUT_COLLECTION_indexed_data_NEW,2),...
498     size(OUTPUT_COLLECTION_indexed_data_NEW,3)*...
499     size(OUTPUT_COLLECTION_indexed_data_NEW,4)]);
500 clear OUTPUT_COLLECTION_indexed_data_NEW
501 index_empty=all(¬parameter_abg_KLR,2);
502 parameter_abg_KLR(index_empty,:)=[];
503 indices_abg_KLR_NEW_unsorted(:,:,index_empty)=[];
504 disp('Calculate real space parameter, unsorted')
505 [cell_parameters_output_unsorted] = ...
506     function_NEWCELLPARAMETER_RESTRICTED(parameter_abg_KLR);
507 [¬,indsmallest_q]=sort(cell_parameters_output_unsorted(:,11));
508 rownumber=1:size(cell_parameters_output_unsorted,1);
509 cell_parameters_output_noline=...
510 cell_parameters_output_unsorted(indsmallest_q,:);
511 cell_parameters_output=horzcat(rownumber',...
512 cell_parameters_output_noline);
513 indices_abg_KLR_NEW=...
514 indices_abg_KLR_NEW_unsorted(:,:,indsmallest_q);
515 clear indices_abg_KLR_NEW_unsorted
516 clear cell_parameters_output_noline
517 d.Value=.9;
518 d.Message = 'Indexing Part II finished..';
519 pause(2)
520
521 d.Value=.95;
522 d.Message = 'Parameter optimization and search for reduced cell.';
523 pause(2)
```

```
A1_parameter_redu_and_opti=zeros(size(cell_parameters_output,1),14);

for solution_number=1:size(cell_parameters_output,1)
    par_test=cell_parameters_output(solution_number,:);
    ind_test=indices_abg_KLR_NEW(:,:,solution_number);
    if par_test(2)==0 && par_test(3)==0
        A1_parameter_redu_and_opti(solution_number,:)=...
            function_REDUCED_CELL_MY_001(par_test,ind_test,qspecc);
    else
        A1_parameter_redu_and_opti(solution_number,:)=...
            function_REDUCED_CELL_MY_UVW(par_test,ind_test,qspecc);
    end
end
[¬,final_sort]=sort(A1_parameter_redu_and_opti(:,11));
final_parameters=A1_parameter_redu_and_opti(final_sort,:);
final_indices=indices_abg_KLR_NEW(:,:,final_sort);
if size(final_parameters,1)>50
    final_parameters_output=final_parameters(1:50,:);
    final_indices_output=final_indices(:,:,1:50);
else
    final_parameters_output=final_parameters;
    final_indices_output=final_indices;
end
d.Value=1;
d.Message = 'Indexing sweep finished.';
close(d)
close(f)
end
```

B.2.12 function_CALCSUBSIS.m

```
function [INDEXED_with_subs_out]=function_CALCSUBSIS(INDEXED_try)
INDEXED_with_subs=INDEXED_try;
u=INDEXED_try(1,1,:);
v=INDEXED_try(1,2,:);
za=INDEXED_try(1,8,:);
zb=INDEXED_try(1,9,:);
cosg=cosd(INDEXED_try(1,6,:));
qspecc=INDEXED_try(1,3,:);

```

```
guv=u.^2.*za.^2+v.^2.*zb.^2-2.*u.*v.*za.*zb.*cosg;
sub1=real((qspecc.^2-guv).^(0.5));
sub2=(qspecc.^2-u.^2.*za.^2-v...
.^2.*zb.^2+2.*u.*v.*za.*zb.*cosg).^(0.5);

INDEXED_with_subs(1,8,:)=sub1;
INDEXED_with_subs(1,9,:)=sub2;
INDEXED_with_subs(1,10,:)=guv;

gzi=INDEXED_with_subs(2:end,2,:);
h=INDEXED_with_subs(2:end,3,:);
k=INDEXED_with_subs(2:end,4,:);

gziqspec=gzi.*INDEXED_with_subs(1,3,:);
zaehler=gziqspec-h.*u.*za.^2-k.*v.*zb.^2+(h.*v+k.*u).*za.*zb.*cosg;
sub3=h.*u.*za.^2+k.*v.*zb.^2-(h.*v+k.*u).*za.*zb.*cosg;
gziterme=zaehler./sub2;
INDEXED_with_subs(2:end,6,:)=gziterme;
INDEXED_with_subs(2:end,7,:)=sub3;
INDEXED_with_subs_out=INDEXED_with_subs;
end
```

B.2.13 function_NEWPACKING.m

```
function [INDEXED_with_subs_NEW] = ...
    function_NEWPACKING(INDEXED_with_subs,redundanzhk)

rlength=2+3*redundanzhk;
number_of_solutions=size(INDEXED_with_subs,3);
INDEXED_with_subs_NEW=zeros(size(INDEXED_with_subs,1),...
    size(INDEXED_with_subs,2)+rlength,'single');

Messwerte_qz=INDEXED_with_subs(2:redundanzhk:end,1:2,1);
package_height=size(zeros(size(Messwerte_qz,1),rlength),1);

for ind_lsg=1:number_of_solutions
    current_page_abg=INDEXED_with_subs(:,:,ind_lsg);
    lsg_abg_ar=current_page_abg(2:end,1:5);
    ausgabe_mw=zeros(size(Messwerte_qz,1),rlength,'single');
    ind_fuer_paket=1;
```

```
17      for ind_mw=1:redundanzhk:size(lsg_abg_ar,1)-redundanzhk+1
18          paket_redhk_er=lsg_abg_ar(ind_mw:ind_mw+redundanzhk-1,:);
19          macht_zeile_pro_qz_redhk_ohne_MW= zeros(1,rlength-2);
20          irhk=1;
21          for rhk=1:redundanzhk
22              macht_zeile_pro_qz_redhk_ohne_MW...
23              (:,irhk:irhk+2)=paket_redhk_er(rhk,3:end);
24              irhk=irhk+3;
25          end
26          macht_zeile_pro_qz_mit_redhk=...
27          horzcat(paket_redhk_er(1,1:2),...
28          macht_zeile_pro_qz_redhk_ohne_MW);
29          ausgabe_mw(ind_fuer_paket,:)=macht_zeile_pro_qz_mit_redhk;
30          ind_fuer_paket=ind_fuer_paket+1;
31      end
32
33      INDEXED_with_subs_NEW(1:size(INDEXED_with_subs,1),...
34          1:size(INDEXED_with_subs,2),ind_lsg)=...
35          INDEXED_with_subs(:,:,ind_lsg);
36
37      INDEXED_with_subs_NEW(1:package_height,...
38          size(INDEXED_with_subs,2)+...
39          1:size(INDEXED_with_subs_NEW,2),ind_lsg)=ausgabe_mw;
40
41  end
42  end
```

B.2.14 function_NEWCELLPARAMETER_RESTRICTED.m

```
1
2   function [output] = function_NEWCELLPARAMETER_RESTRICTED(...
3       parameter_abg_KLR)
4
5   u=parameter_abg_KLR(:,7);
6   v=parameter_abg_KLR(:,8);
7   w=parameter_abg_KLR(:,9);
8
9   a_in=parameter_abg_KLR(:,1);
10  b_in=parameter_abg_KLR(:,2);
11  gamma_in=parameter_abg_KLR(:,3);
```

```
12
13 kappa=parameter_abg_KLR(:,4);
14 lambda=parameter_abg_KLR(:,5);
15 rho=parameter_abg_KLR(:,6);
16
17 RMSD_qxy_in=parameter_abg_KLR(:,10);
18 RMSD_qz_in=parameter_abg_KLR(:,11);
19 RMSD_q_in=parameter_abg_KLR(:,12);
20 RMSD_qspec_in=parameter_abg_KLR(:,13);
21
22 ∆=-1./(2.*pi).*kappa.*a_in.*sind(gamma_in);
23 mu=-1./(2.*pi).*lambda.*b_in.*sind(gamma_in);
24
25 sin_eps=sind(gamma_in)./sqrt(sind(gamma_in).^2+∆.^2+mu.^2+...
26     2.*∆.*mu.*cosd(gamma_in));
27
28 cos_alpha=(sin_eps.*mu+sin_eps.*∆.*cosd(gamma_in))./sind(gamma_in);
29 alpha_out=acosd(cos_alpha);
30
31 cos_beta=(sin_eps.*∆+sin_eps.*mu.*cosd(gamma_in))./sind(gamma_in);
32 beta_out=acosd(cos_beta);
33 gamma_out=gamma_in;
34
35 a_out=a_in;
36 b_out=b_in;
37 c_out=(2.*pi)./(sin_eps.*rho);
38
39 vol_sq=a_out.*b_out.*c_out.*(1-cosd(alpha_out).*cosd(alpha_out)-...
40     cosd(beta_out).*cosd(beta_out)-cosd(gamma_out).*cosd(gamma_out)...
41     +2.*cosd(alpha_out).*cosd(beta_out).*cosd(gamma_out)).^(1/2);
42
43 output=horzcat(u,v,w,a_out,b_out,c_out,alpha_out,beta_out,gamma_out,...
44     vol_sq,RMSD_q_in,RMSD_qz_in,RMSD_qxy_in,RMSD_qspec_in);
45 end
```

B.2.15 function_REDUCED_CELL_MY_001.m

```
1 function [A1_parameter_OPTIMIERT] = ...
      function_REDUCED_CELL_MY_001(par_test,ind_test,qspec)
2
```

```
u=par_test(1,2);
v=par_test(1,3);
w=par_test(1,4);
a=  par_test(1,5);
b= par_test(1,6);
c= par_test(1,7);
alpha=par_test(1,8);
beta=par_test(1,9);
gamma=par_test(1,10);

[A001_star,GSPEC] = ...
function_CALC_A001_STAR(a,b,c,alpha,beta,gamma,u,v,w);

if size(ind_test,1)>50
    N=50;
else
    N=size(ind_test,1);
end

hkl=ind_test(:,[3 4 5]);
row_variations_indices=nchoosek(1:N,3);

HKL=zeros(3,3,size(row_variations_indices,1));
hkl_det=zeros(size(row_variations_indices,1),1);

for i=1:size(row_variations_indices,1)
    linen=row_variations_indices(i,:);
    testmat=horzcat(hkl(linen(1),:)',...
        hkl(linen(2),:)',hkl(linen(3),:)');
    detcalc=det(testmat);
    if abs(detcalc)⩽1e-6
        detcalc=0;
    end
    hkl_det(i,:)=round(detcalc,0);
    HKL(:,:,i)=testmat;
end

hkl_det_double=zeros(size(hkl_det,1),2);
hkl_det_double(:,1)=hkl_det;
hkl_det_double(:,2)=abs(hkl_det);


[¬,shkl]=sort(hkl_det_double(:,2));
```

```
hkl_double_sorted=hkl_det_double(shkl,:);
HKL=HKL(:,:,shkl);

indzerodet=hkl_double_sorted(:,1)≠0;

hkl_determinant_ascending=hkl_double_sorted(indzerodet,:);
HKL_as_columns=HKL(:,:,indzerodet);

hk_dets_pos_1er_test=hkl_determinant_ascending(:,1)==1;
hkl_dets_pos_1er=HKL_as_columns(:,:,hk_dets_pos_1er_test);

hk_dets_neg_1er_test=hkl_determinant_ascending(:,1)==-1;
hkl_dets_neg_1er=HKL_as_columns(:,:,hk_dets_neg_1er_test);

HKL_vec_mit_1er_determinante=...
cat(3,hkl_dets_pos_1er,hkl_dets_neg_1er);

if size(HKL_vec_mit_1er_determinante,3)>0
    hkl_matti1=HKL_vec_mit_1er_determinante(:,:,1);
    g1t=A001_star*hkl_matti1(:,1);
    g2t=A001_star*hkl_matti1(:,2);
    g3t=A001_star*hkl_matti1(:,3);


    G=[g1t';g2t';g3t'];
    G_inv=inv(G);
    m_vec=LPermutation(-8:8,3);
    m_vec(all(¬m_vec,2),:)=[];

    v_stored=zeros(3,1,size(m_vec,1));
    v_length_stored=zeros(size(v_stored,1),2);
    for dope=1:size(m_vec)
        vveco=2.*pi.*G_inv*m_vec(dope,:)';
        v_stored(:,:,dope)=vveco;
        v_length_stored(dope,1)=norm(vveco);
        v_length_stored(dope,2)=round(vveco(3).*GSPEC./(2.*pi));
    end

    [¬,ski]=sort(v_length_stored(:,1));

    lengths_sorted=v_length_stored(ski,:);
    vectors_sorted=v_stored(:,:,ski);

```

```
89      [V,ial,¬]=unique(lengths_sorted(:,1));
90      Vv=vectors_sorted(:,:,ial);
91
92      for coline=1:size(row_variations_indices,1)
93          combine=row_variations_indices(coline,:);
94          det_test=det([Vv(:,:,combine(1)),...
95          Vv(:,:,combine(2)),Vv(:,:,combine(3))]);
96          if abs(det_test)>0.0001
97              par_test(1,5)=V(combine(1));
98              par_test(1,6)=V(combine(2));
99              par_test(1,7)=V(combine(3));
100             break
101         end
102
103     end
104
105     qxyz=sqrt(ind_test(:,1).^2+ind_test(:,2).^2);
106
107     [astern,bstern,cstern,alphastern,betastern,gammastern] = ...
108         function_REAL_TO_RECIPROCAL_v9( par_test(:,5),...
109          par_test(:,6), par_test(:,7),...
110          par_test(:,8), par_test(:,9),par_test(:,10));
111
112     [gz_uvw,gxyz_uvw] = ...
        function_GZ_GXYZ_UVW(GSPEC,u,v,w,astern,bstern,...
113     cstern,alphastern,betastern,gammastern,ind_test(:,3),...
114     ind_test(:,4),ind_test(:,5));
115
116     qg_feld_mit_hkl_uvw=...
117     horzcat(qxyz-gxyz_uvw,ind_test(:,2)-gz_uvw,...
118         ind_test(:,3),ind_test(:,4),ind_test(:,5));
119
120     [epsilon_vector_uvw] = ...
121         function_EPSILON_UVW(GSPEC,par_test(2),par_test(3),...
122         par_test(4),qg_feld_mit_hkl_uvw,astern,bstern,...
123         cstern,alphastern,betastern,gammastern);
124
125     rec_par_optimized=[astern;bstern;cstern;alphastern;...
126         betastern;gammastern]+epsilon_vector_uvw;
127
128     alpha_neu=acosd((cosd(rec_par_optimized(5))*...
129         cosd(rec_par_optimized(6))...
130         -cosd(rec_par_optimized(4)))/(sind(rec_par_optimized(5))...
```

```
131         *sind(rec_par_optimized(6))));
132
133     beta_neu=acosd((cosd(rec_par_optimized(4))*...
134         cosd(rec_par_optimized(6))-cosd(rec_par_optimized(5)))...
135         /(sind(rec_par_optimized(4))*sind(rec_par_optimized(6))));
136
137     gamma_neu=acosd((cosd(rec_par_optimized(4))*...
138         cosd(rec_par_optimized(5))-cosd(rec_par_optimized(6)))/...
139         (sind(rec_par_optimized(4))*sind(rec_par_optimized(5))));
140
141     astarn=rec_par_optimized(1);
142     bstarn=rec_par_optimized(2);
143     cstarn=rec_par_optimized(3);
144
145     alpstarn=rec_par_optimized(4);
146     betstarn=rec_par_optimized(5);
147     gamstarn=rec_par_optimized(6);
148
149     Volstarn=astarn*bstarn*cstarn*...
150     sqrt(1-cosd(alpstarn)^2-cosd(betstarn)^2-...
151         cosd(gamstarn)^2+2*cosd(alpstarn)*...
152         cosd(betstarn)*cosd(gamstarn));
153
154     a_neu=2*pi*bstarn*cstarn*sind(alpstarn)/Volstarn;
155     b_neu=2*pi*astarn*cstarn*sind(betstarn)/Volstarn;
156     c_neu=2*pi*astarn*bstarn*sind(gamstarn)/Volstarn;
157
158     V_neu=a_neu*b_neu*c_neu*sind(alpstarn)*...
159     sind(beta_neu)*sind(gamma_neu);
160
161     [gz_new,gxyz_new] = ...
           function_GZ_GXYZ_UVW(GSPEC,par_test(:,2),...
162         par_test(:,3),par_test(1,4),...
163         rec_par_optimized(1,:),rec_par_optimized(2,:),...
164         rec_par_optimized(3,:),rec_par_optimized(4,:),...
165         rec_par_optimized(5,:),rec_par_optimized(6,:),...
166         ind_test(:,3),ind_test(:,4),ind_test(:,5));
167
168     gxy_new=sqrt(((2.*pi)./(a_neu.*sind(gamma_neu))).^2.*...
169         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).^2+...
170         (par_test(:,2).*ind_test(:,1)./GSPEC).^2)+ ...
171         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*((ind_test(:,4)...
172         -par_test(:,3).*ind_test(:,2)./GSPEC).^2+(par_test(:,3).*...
```

```
173         ind_test(:,1)./GSPEC).^2)-...
174         2.*(2.*pi./(a_neu.*sind(gamma_neu))).*(2.*pi./...
175         (b_neu.*sind(gamma_neu))).*cosd(gamma_neu).*...
176         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).*...
177         (ind_test(:,4)-par_test(:,3).*ind_test(:,2)./GSPEC)+...
178         par_test(:,2).*par_test(:,3).*(ind_test(:,1)./GSPEC).^2)...
179         -((ind_test(:,3).*par_test(:,3)-ind_test(:,4).*...
180         par_test(:,2)).^2./GSPEC.^2).*(2.*pi./(a_neu.*...
181         sind(gamma_neu))).^2.*...
182         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*sind(gamma_neu).^2);
183
184     num=size(qxyz,1);
185
186     RMSD_qxy_new=(sum((gxy_new-ind_test(:,1)).^2)/num).^(1/2);
187     RMSD_qz_new=(sum((gz_new-ind_test(:,2)).^2)/num).^(1/2);
188     RMSD_qxyz_new=(sum((gxyz_new-qxyz).^2)/num).^(1/2);
189
190
191     A1_parameter_OPTIMIERT=[u,v,w,a_neu,b_neu,c_neu,...
192     alpha_neu,beta_neu,gamma_neu,...
193         V_neu,RMSD_qxyz_new,RMSD_qz_new,...
194         RMSD_qxy_new,abs(GSPEC-qspec)];
195
196 else
197     qxyz=sqrt(ind_test(:,1).^2+ind_test(:,2).^2);
198
199     [astern,bstern,cstern,alphastern,betastern,gammastern] = ...
200         function_REAL_TO_RECIPROCAL_v9( par_test(:,5),...
201          par_test(:,6), par_test(:,7),...
202          par_test(:,8), par_test(:,9),par_test(:,10));
203
204     [gz_uvw,gxyz_uvw] = ...
        function_GZ_GXYZ_UVW(GSPEC,u,v,w,astern,bstern,...
205         cstern,alphastern,betastern,gammastern,ind_test(:,3),...
206         ind_test(:,4),ind_test(:,5));
207
208     qg_feld_mit_hkl_uvw=horzcat(qxyz-gxyz_uvw,...
209     ind_test(:,2)-gz_uvw,...
210         ind_test(:,3),ind_test(:,4),ind_test(:,5));
211
212     [epsilon_vector_uvw] = ...
213         function_EPSILON_UVW(GSPEC,par_test(2),par_test(3),...
214         par_test(4),qg_feld_mit_hkl_uvw,astern,bstern,...
```

```
215         cstern,alphastern,betastern,gammastern);
216
217     rec_par_optimized=[astern;bstern;cstern;alphastern;...
218         betastern;gammastern]+epsilon_vector_uvw;
219
220     alpha_neu=acosd((cosd(rec_par_optimized(5))*...
221         cosd(rec_par_optimized(6))...
222         -cosd(rec_par_optimized(4)))/(sind(rec_par_optimized(5))...
223         *sind(rec_par_optimized(6))));
224
225     beta_neu=acosd((cosd(rec_par_optimized(4))*...
226         cosd(rec_par_optimized(6))-cosd(rec_par_optimized(5)))...
227         /(sind(rec_par_optimized(4))*sind(rec_par_optimized(6))));
228
229     gamma_neu=acosd((cosd(rec_par_optimized(4))*...
230         cosd(rec_par_optimized(5))-cosd(rec_par_optimized(6)))/...
231         (sind(rec_par_optimized(4))*sind(rec_par_optimized(5))));
232
233     astarn=rec_par_optimized(1);
234     bstarn=rec_par_optimized(2);
235     cstarn=rec_par_optimized(3);
236
237     alpstarn=rec_par_optimized(4);
238     betstarn=rec_par_optimized(5);
239     gamstarn=rec_par_optimized(6);
240
241     Volstarn=astarn*bstarn*cstarn*...
242     sqrt(1-cosd(alpstarn)^2-cosd(betstarn)^2-...
243         cosd(gamstarn)^2+2*cosd(alpstarn)*...
244         cosd(betstarn)*cosd(gamstarn));
245
246     a_neu=2*pi*bstarn*cstarn*sind(alpstarn)/Volstarn;
247     b_neu=2*pi*astarn*cstarn*sind(betstarn)/Volstarn;
248     c_neu=2*pi*astarn*bstarn*sind(gamstarn)/Volstarn;
249
250     V_neu=a_neu*b_neu*c_neu*sind(alpstarn)*...
251     sind(beta_neu)*sind(gamma_neu);
252
253     [gz_new,gxyz_new] = ...
        function_GZ_GXYZ_UVW(GSPEC,par_test(:,2),...
254         par_test(:,3),par_test(1,4),...
255         rec_par_optimized(1,:),rec_par_optimized(2,:),...
256         rec_par_optimized(3,:),rec_par_optimized(4,:),...
```

```
257         rec_par_optimized(5,:),rec_par_optimized(6,:),...
258         ind_test(:,3),ind_test(:,4),ind_test(:,5));
259
260     gxy_new=sqrt(((2.*pi)./(a_neu.*sind(gamma_neu))).^2.*...
261         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).^2+...
262         (par_test(:,2).*ind_test(:,1)./GSPEC).^2)+ ...
263         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*((ind_test(:,4)...
264         -par_test(:,3).*ind_test(:,2)./GSPEC).^2+(par_test(:,3).*...
265         ind_test(:,1)./GSPEC).^2)-...
266         2.*(2.*pi./(a_neu.*sind(gamma_neu))).*(2.*pi./...
267         (b_neu.*sind(gamma_neu))).*cosd(gamma_neu).*...
268         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).*...
269         (ind_test(:,4)-par_test(:,3).*ind_test(:,2)./GSPEC)+...
270         par_test(:,2).*par_test(:,3).*(ind_test(:,1)./GSPEC).^2)...
271         -((ind_test(:,3).*par_test(:,3)-ind_test(:,4).*...
272         par_test(:,2)).^2./GSPEC.^2).*(2.*pi./(a_neu.*...
273         sind(gamma_neu))).^2.*...
274         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*sind(gamma_neu).^2);
275
276     num=size(qxyz,1);
277
278     RMSD_qxy_new=(sum((gxy_new-ind_test(:,1)).^2)/num).^(1/2);
279     RMSD_qz_new=(sum((gz_new-ind_test(:,2)).^2)/num).^(1/2);
280     RMSD_qxyz_new=(sum((gxyz_new-qxyz).^2)/num).^(1/2);
281
282     A1_parameter_OPTIMIERT=[u,v,w,a_neu,b_neu,c_neu,...
283     alpha_neu,beta_neu,gamma_neu,...
284         V_neu,RMSD_qxyz_new,RMSD_qz_new,...
285         RMSD_qxy_new,abs(GSPEC-qspec)];
286 end
287 end
```

B.2.16 function_CALC_A001_STAR.m

```
1 function [A_001_star,GSPEC2] = ...
      function_CALC_A001_STAR(a,b,c,alpha,beta,gamma,u,v,w)
2 volume=a*b*c*(1-cosd(alpha)^2-cosd(beta)^2-...
3 cosd(gamma)^2+2*cosd(alpha)*...
4     cosd(beta)*cosd(gamma))^(1/2);
5
```

```
astern=(2*pi*b*c*sind(alpha))/volume;
bstern=(2*pi*a*c*sind(beta))/volume;
cstern=(2*pi*a*b*sind(gamma))/volume;

sin_alpha_stern=volume/(a*b*c*sind(beta)*sind(gamma));
cos_alpha_stern=...
(cosd(beta)*cosd(gamma)-cosd(alpha))/(sind(beta)*sind(gamma));
cos_beta_stern=...
(cosd(alpha)*cosd(gamma)-cosd(beta))/(sind(alpha)*sind(gamma));
sin_beta_stern=...
volume/(a*b*c*sind(alpha)*sind(gamma));
cos_gamma_stern=...
(cosd(alpha)*cosd(beta)-cosd(gamma))/(sind(alpha)*sind(beta));
A_001_star=zeros(3,3);
A_001_star(1,1)=astern*sin_beta_stern*sind(gamma);
A_001_star(2,1)=-astern*sin_beta_stern*cosd(gamma);
A_001_star(3,1)=astern*cos_beta_stern;
A_001_star(2,2)=bstern*sin_alpha_stern;
A_001_star(3,2)=bstern*cos_alpha_stern;
A_001_star(3,3)=cstern;

GSPEC2=sqrt(u^2*astern^2+v^2*bstern^2+w^2*cstern^2+...
2*u*v*astern*bstern*cos_gamma_stern+...
    2*u*w*astern*cstern*cos_beta_stern+...
    2*v*w*bstern*cstern*cos_alpha_stern);

end
```

B.2.17 function_REAL_TO_RECIPROCAL_v9.m

```
function [astern,bstern,cstern,alphastern,betastern,gammastern] ...
    = ...
    function_REAL_TO_RECIPROCAL_v9(a,b,c,alpha,beta,gamma)
V=a.*b.*c.*(1-cosd(alpha).*cosd(alpha)-...
    cosd(beta).*cosd(beta)-cosd(gamma).*cosd(gamma)...
    +2.*cosd(alpha).*cosd(beta).*cosd(gamma)).^(1/2);
astern=(2*pi).*(b.*c.*sind(alpha))./V;
bstern=(2*pi).*(a.*c.*sind(beta))./V;
cstern=(2*pi).*(a.*b.*sind(gamma))./V;
alphastern=acosd((cosd(beta).*cosd(gamma)-cosd(alpha))./...
```

```
10      (sind(beta).*sind(gamma)));
11  betastern=acosd((cosd(alpha).*cosd(gamma)-cosd(beta))./...
12      (sind(alpha).*sind(gamma)));
13  gammastern=acosd((cosd(alpha).*cosd(beta)-cosd(gamma))./...
14      (sind(alpha).*sind(beta)));
15  end
```

B.2.18 function_EPSILON_UVW.m

```
1   function [epsilon_vektor,f_matrix,f_right] = ...
        function_EPSILON_UVW(qspec,...
2       u,v,w,qg_feld_mit_hkl,astern,bstern,...
3       cstern,alphastern,betastern,gammastern)
4
5   dx=1e-5;
6   f_para_kartei=zeros(6,1,size(qg_feld_mit_hkl,1));
7   f_matrix_kartei=zeros(6,6,size(qg_feld_mit_hkl,1));
8
9   for mwi=1:size(qg_feld_mit_hkl,1)
10
11      h=qg_feld_mit_hkl(mwi,3);
12      k=qg_feld_mit_hkl(mwi,4);
13      l=qg_feld_mit_hkl(mwi,5);
14      delta_q_i=[qg_feld_mit_hkl(mwi,1);...
15      qg_feld_mit_hkl(mwi,2)];
16
17      func_gxyz = @(h,k,l,astern,bstern,cstern,alphastern,...
18          betastern,gammastern) ...
                ((h.^2.*astern.^2+k.^2.*bstern.^2+l.^2.*...
19          cstern.^2+2.*h.*k.*astern.*bstern...
20          .*cosd(gammastern)+2.*h.*l.*...
21          astern.*cstern.*cosd(betastern)+...
22          2.*k.*l.*bstern.*cstern.*cosd(alphastern)).^(1/2));
23
24      func_gz = ...
            @(u,v,w,qspec,h,k,l,astern,bstern,cstern,alphastern,...
25          betastern,gammastern) ((h.*u.*astern.^2+k.*v.*bstern.^2+...
26          l.*w.*cstern.^2+...
27          (h.*v+k.*u).*astern.*bstern.*cosd(gammastern)+...
28          (h.*w+l.*u).*astern.*cstern.*cosd(betastern)+...
```

```
        (k.*w+l.*v).*bstern.*cstern.*cosd(alphastern))./qspec);

    fa_gxyz=((func_gxyz(h,k,l,astern+dx,bstern,cstern,...
        alphastern,betastern,gammastern)-func_gxyz(h,k,l,...
        astern-dx,bstern,cstern,...
        alphastern,betastern,gammastern))./(2*dx));

    fa_gz=(func_gz(u,v,w,qspec,h,k,l,astern+dx,bstern,...
    cstern,alphastern,betastern,...
        gammastern)-func_gz(u,v,w,qspec,h,k,l,...
        astern-dx,bstern,cstern,alphastern,betastern,...
        gammastern))./(2*dx);

    fa_i=[fa_gxyz,fa_gz];

    fb_gxyz=((func_gxyz(h,k,l,astern,bstern+dx,cstern,...
        alphastern,betastern,gammastern)-func_gxyz(h,k,l,...
        astern,bstern-dx,cstern,...
        alphastern,betastern,gammastern))./(2*dx));

    fb_gz=(func_gz(u,v,w,qspec,h,k,l,astern,...
    bstern+dx,cstern,alphastern,betastern,...
        gammastern)-func_gz(u,v,w,qspec,h,k,l,astern,...
        bstern-dx,cstern,alphastern,betastern,...
        gammastern))./(2*dx);

    fb_i=[fb_gxyz,fb_gz];

    fc_gxyz=((func_gxyz(h,k,l,astern,bstern,cstern+dx,...
        alphastern,betastern,gammastern)-func_gxyz(h,k,l,...
        astern,bstern,cstern-dx,...
        alphastern,betastern,gammastern))./(2*dx));

    fc_gz=(func_gz(u,v,w,qspec,h,k,l,astern,...
    bstern,cstern+dx,alphastern,betastern,...
        gammastern)-func_gz(u,v,w,qspec,h,k,l,astern,...
        bstern,cstern-dx,alphastern,betastern,...
        gammastern))./(2*dx);

    fc_i=[fc_gxyz,fc_gz];

    falpha_gxyz=((func_gxyz(h,k,l,astern,bstern,cstern,...
        alphastern+dx,betastern,gammastern)-func_gxyz(h,k,l,...
```

```
72          astern,bstern,cstern,...
73          alphastern-dx,betastern,gammastern))./(2*dx));
74
75      falpha_gz=(func_gz(u,v,w,qspec,h,k,l,astern,bstern,...
76      cstern,alphastern+dx,betastern,...
77          gammastern)-func_gz(u,v,w,qspec,h,k,l,...
78          astern,bstern,cstern,alphastern-dx,betastern,...
79          gammastern))./(2*dx);
80
81      falpha_i=[falpha_gxyz,falpha_gz];
82
83      fbeta_gxyz=((func_gxyz(h,k,l,astern,bstern,cstern,...
84          alphastern,betastern+dx,gammastern)-func_gxyz(h,k,l,...
85          astern,bstern,cstern,...
86          alphastern,betastern-dx,gammastern))./(2*dx));
87
88      fbeta_gz=(func_gz(u,v,w,qspec,h,k,l,astern,bstern,...
89      cstern,alphastern,betastern+dx,...
90          gammastern)-func_gz(u,v,w,qspec,h,k,l,astern,...
91          bstern,cstern,alphastern,betastern-dx,...
92          gammastern))./(2*dx);
93
94      fbeta_i=[fbeta_gxyz,fbeta_gz];
95
96      fgamma_gxyz=((func_gxyz(h,k,l,astern,bstern,cstern,...
97          alphastern,betastern,gammastern+dx)-func_gxyz(h,k,l,...
98          astern,bstern,cstern,...
99          alphastern,betastern,gammastern-dx))./(2*dx));
100
101     fgamma_gz=(func_gz(u,v,w,qspec,h,k,l,astern,...
102     bstern,cstern,alphastern,betastern,...
103         gammastern+dx)-func_gz(u,v,w,qspec,h,k,l,...
104         astern,bstern,cstern,alphastern,betastern,...
105         gammastern-dx))./(2*dx);
106
107     fgamma_i=[fgamma_gxyz,fgamma_gz];
108
109     f_para_pro_messwert=[fa_i;fb_i;fc_i;falpha_i;fbeta_i;fgamma_i];
110
111     f_para_kartei(:,:,mwi)=f_para_pro_messwert*▵_q_i;
112
113     f_matrix_kartei(1,1,mwi)=dot(fa_i,fa_i);
114     f_matrix_kartei(2,1,mwi)=dot(fa_i,fb_i);
```

```
115     f_matrix_kartei(3,1,mwi)=dot(fa_i,fc_i);
116     f_matrix_kartei(4,1,mwi)=dot(fa_i,falpha_i);
117     f_matrix_kartei(5,1,mwi)=dot(fa_i,fbeta_i);
118     f_matrix_kartei(6,1,mwi)=dot(fa_i,fgamma_i);
119
120     f_matrix_kartei(1,2,mwi)=dot(fa_i,fb_i);
121     f_matrix_kartei(2,2,mwi)=dot(fb_i,fb_i);
122     f_matrix_kartei(3,2,mwi)=dot(fb_i,fc_i);
123     f_matrix_kartei(4,2,mwi)=dot(fb_i,falpha_i);
124     f_matrix_kartei(5,2,mwi)=dot(fb_i,fbeta_i);
125     f_matrix_kartei(6,2,mwi)=dot(fb_i,fgamma_i);
126
127     f_matrix_kartei(1,3,mwi)=dot(fa_i,fc_i);
128     f_matrix_kartei(2,3,mwi)=dot(fb_i,fc_i);
129     f_matrix_kartei(3,3,mwi)=dot(fc_i,fc_i);
130     f_matrix_kartei(4,3,mwi)=dot(fc_i,falpha_i);
131     f_matrix_kartei(5,3,mwi)=dot(fc_i,fbeta_i);
132     f_matrix_kartei(6,3,mwi)=dot(fc_i,fgamma_i);
133
134     f_matrix_kartei(1,4,mwi)=dot(fa_i,falpha_i);
135     f_matrix_kartei(2,4,mwi)=dot(fb_i,falpha_i);
136     f_matrix_kartei(3,4,mwi)=dot(fc_i,falpha_i);
137     f_matrix_kartei(4,4,mwi)=dot(falpha_i,falpha_i);
138     f_matrix_kartei(5,4,mwi)=dot(falpha_i,fbeta_i);
139     f_matrix_kartei(6,4,mwi)=dot(falpha_i,fgamma_i);
140
141     f_matrix_kartei(1,5,mwi)=dot(fa_i,fbeta_i);
142     f_matrix_kartei(2,5,mwi)=dot(fb_i,fbeta_i);
143     f_matrix_kartei(3,5,mwi)=dot(fc_i,fbeta_i);
144     f_matrix_kartei(4,5,mwi)=dot(falpha_i,fbeta_i);
145     f_matrix_kartei(5,5,mwi)=dot(fbeta_i,fbeta_i);
146     f_matrix_kartei(6,5,mwi)=dot(fbeta_i,fgamma_i);
147
148     f_matrix_kartei(1,6,mwi)=dot(fa_i,fgamma_i);
149     f_matrix_kartei(2,6,mwi)=dot(fb_i,fgamma_i);
150     f_matrix_kartei(3,6,mwi)=dot(fc_i,fgamma_i);
151     f_matrix_kartei(4,6,mwi)=dot(falpha_i,fgamma_i);
152     f_matrix_kartei(5,6,mwi)=dot(fbeta_i,fgamma_i);
153     f_matrix_kartei(6,6,mwi)=dot(fgamma_i,fgamma_i);
154
155 end
156
157 f_right=sum(f_para_kartei,3);
```

```
158 f_matrix=sum(f_matrix_kartei,3);
159 epsilon_vektor=f_matrix\f_right;
160
161 if all(isnan(epsilon_vektor))==true
162     epsilon_vektor=zeros(6,1,'single')
163 elseif any(isinf(epsilon_vektor))==true
164     epsilon_vektor=zeros(6,1,'single')
165 end
166 end
```

B.2.19 function_GZ_GXYZ_UVW.m

```
1  function [gz_calculated,gxyz_calculated] = ...
       function_GZ_GXYZ_UVW(qspec,u,v,w,astern,bstern,cstern,...
2      alphastern,betastern,gammastern,H_vec,K_vec,L_vec)
3
4  gz_calculated=(H_vec.*u.*astern.^2+K_vec.*v.*bstern.^2+...
5  L_vec.*w.*cstern.^2+...
6      (H_vec.*v+K_vec.*u).*astern.*bstern.*cosd(gammastern)+...
7      (H_vec.*w+L_vec.*u).*astern.*cstern.*cosd(betastern)+...
8      (K_vec.*w+L_vec.*v).*bstern.*cstern.*cosd(alphastern))./qspec;
9
10 gxyz_calculated=sqrt(H_vec.^2.*astern.^2+K_vec.^2.*bstern.^2+...
11 L_vec.^2.*cstern.^2+...
12     2.*H_vec.*K_vec.*astern.*bstern.*cosd(gammastern)+...
13     2.*H_vec.*L_vec.*astern.*cstern.*cosd(betastern)+...
14     2.*K_vec.*L_vec.*bstern.*cstern.*cosd(alphastern));
15 end
```

B.2.20 function_REDUCED_CELL_MY_UVW.m

```
1  function [A1_parameter_OPTIMIERT] = ...
       function_REDUCED_CELL_MY_UVW(par_test,ind_test,qspec)
2
3  u=par_test(1,2);
4  v=par_test(1,3);
5  w=par_test(1,4);
6  a=  par_test(1,5);
```

```
b= par_test(1,6);
c= par_test(1,7);
alpha=par_test(1,8);
beta=par_test(1,9);
gamma=par_test(1,10);

[A001_star,GSPEC] = ...
    function_CALC_A001_STAR(a,b,c,alpha,beta,gamma,u,v,w);

if size(ind_test,1)>50
    N=50;
else
    N=size(ind_test,1);
end

hkl=ind_test(:,[3 4 5]);
row_variations_indices=nchoosek(1:N,3);

HKL=zeros(3,3,size(row_variations_indices,1));
hkl_det=zeros(size(row_variations_indices,1),1);

for i=1:size(row_variations_indices,1)
    linen=row_variations_indices(i,:);
    testmat=horzcat(hkl(linen(1),:)',...
        hkl(linen(2),:)',hkl(linen(3),:)');
    detcalc=det(testmat);
    if abs(detcalc)⩽1e-6
        detcalc=0;
    end
    hkl_det(i,:)=round(detcalc,0);
    HKL(:,:,i)=testmat;
end

hkl_det_double=zeros(size(hkl_det,1),2);
hkl_det_double(:,1)=hkl_det;
hkl_det_double(:,2)=abs(hkl_det);

[¬,shkl]=sort(hkl_det_double(:,2));
hkl_double_sorted=hkl_det_double(shkl,:);
HKL=HKL(:,:,shkl);

indzerodet=hkl_double_sorted(:,1)≠0;
hkl_determinant_ascending=hkl_double_sorted(indzerodet,:);
```

```
HKL_as_columns=HKL(:,:,indzerodet);

hk_dets_pos_1er_test=hkl_determinant_ascending(:,1)==1;
hkl_dets_pos_1er=HKL_as_columns(:,:,hk_dets_pos_1er_test);
hk_dets_neg_1er_test=hkl_determinant_ascending(:,1)==-1;
hkl_dets_neg_1er=HKL_as_columns(:,:,hk_dets_neg_1er_test);
HKL_vec_mit_1er_determinante=...
cat(3,hkl_dets_pos_1er,hkl_dets_neg_1er);

if size(HKL_vec_mit_1er_determinante,3)>0
    hkl_matti1=HKL_vec_mit_1er_determinante(:,:,1);

    sigma1=A001_star*[0;0;1];
    sigma2=A001_star*[u;v;w];
    s1s2=cross(sigma1,sigma2);

    n=s1s2./norm(s1s2);
    n1=n(1);
    n2=n(2);
    n3=n(3);

    PHI=acosd(dot(sigma1,sigma2)./(norm(sigma1)*norm(sigma2)));

    R=zeros(3,3);
    R(1,1)=n1.^2.*(1-cosd(PHI))+cosd(PHI);
    R(2,2)=n2.^2.*(1-cosd(PHI))+cosd(PHI);
    R(3,3)=n3.^2.*(1-cosd(PHI))+cosd(PHI);
    R(2,1)=n1.*n2.*(1-cosd(PHI))-n3.*sind(PHI);
    R(3,1)=n1.*n3.*(1-cosd(PHI))+n2.*sind(PHI);
    R(1,2)=n1.*n2.*(1-cosd(PHI))+n3.*sind(PHI);
    R(3,2)=n2.*n3.*(1-cosd(PHI))-n1.*sind(PHI);
    R(1,3)=n1.*n3.*(1-cosd(PHI))-n2.*sind(PHI);
    R(2,3)=n2.*n3.*(1-cosd(PHI))+n1.*sind(PHI);

    g1t=R*A001_star*hkl_matti1(:,1);
    g2t=R*A001_star*hkl_matti1(:,2);
    g3t=R*A001_star*hkl_matti1(:,3);

    G=[g1t';g2t';g3t'];
    G_inv=inv(G);
    m_vec=LPermutation(-8:8,3);
    m_vec(all(¬m_vec,2),:)=[];

```

```
92      v_stored=zeros(3,1,size(m_vec,1));
93      v_length_stored=zeros(size(v_stored,1),2);
94      for dope=1:size(m_vec)
95          vveco=2.*pi.*G_inv*m_vec(dope,:)';
96          v_stored(:,:,dope)=vveco;
97          v_length_stored(dope,1)=norm(vveco);
98          v_length_stored(dope,2)=round(vveco(3).*GSPEC./(2.*pi));
99      end
100     [¬,ski]=sort(v_length_stored(:,1));
101
102     lengths_sorted=v_length_stored(ski,:);
103     vectors_sorted=v_stored(:,:,ski);
104
105     [V,ial,¬]=unique(lengths_sorted(:,1));
106     Vv=vectors_sorted(:,:,ial);
107
108     for coline=1:size(row_variations_indices,1)
109         combine=row_variations_indices(coline,:);
110         det_test=det([Vv(:,:,combine(1)),...
111         Vv(:,:,combine(2)),Vv(:,:,combine(3))]);
112         if abs(det_test)>0.0001
113             par_test(1,5)=V(combine(1));
114             par_test(1,6)=V(combine(2));
115             par_test(1,7)=V(combine(3));
116             break
117         end
118
119     end
120
121     qxyz=sqrt(ind_test(:,1).^2+ind_test(:,2).^2);
122     [astern,bstern,cstern,alphastern,betastern,gammastern] = ...
123         function_REAL_TO_RECIPROCAL_v9( par_test(:,5),...
124          par_test(:,6), par_test(:,7),...
125          par_test(:,8), par_test(:,9),par_test(:,10));
126
127     [gz_uvw,gxyz_uvw] = ...
        function_GZ_GXYZ_UVW(GSPEC,u,v,w,astern,bstern,...
128         cstern,alphastern,betastern,gammastern,ind_test(:,3),...
129         ind_test(:,4),ind_test(:,5));
130
131     qg_feld_mit_hkl_uvw=...
132     horzcat(qxyz-gxyz_uvw,ind_test(:,2)-gz_uvw,...
133         ind_test(:,3),ind_test(:,4),ind_test(:,5));
```

```
134
135     [epsilon_vector_uvw] = ...
136         function_EPSILON_UVW(GSPEC,par_test(2),par_test(3),...
137         par_test(4),qg_feld_mit_hkl_uvw,astern,bstern,...
138         cstern,alphastern,betastern,gammastern);
139
140     rec_par_optimized=[astern;bstern;cstern;alphastern;...
141         betastern;gammastern]+epsilon_vector_uvw;
142
143
144     alpha_neu=acosd((cosd(rec_par_optimized(5))*...
145         cosd(rec_par_optimized(6))...
146         -cosd(rec_par_optimized(4)))/(sind(rec_par_optimized(5))...
147         *sind(rec_par_optimized(6))));
148
149     beta_neu=acosd((cosd(rec_par_optimized(4))*...
150         cosd(rec_par_optimized(6))-cosd(rec_par_optimized(5)))...
151         /(sind(rec_par_optimized(4))*sind(rec_par_optimized(6))));
152
153     gamma_neu=acosd((cosd(rec_par_optimized(4))*...
154         cosd(rec_par_optimized(5))-cosd(rec_par_optimized(6)))/...
155         (sind(rec_par_optimized(4))*sind(rec_par_optimized(5))));
156
157     astarn=rec_par_optimized(1);
158     bstarn=rec_par_optimized(2);
159     cstarn=rec_par_optimized(3);
160
161     alpstarn=rec_par_optimized(4);
162     betstarn=rec_par_optimized(5);
163     gamstarn=rec_par_optimized(6);
164
165     Volstarn=astarn*bstarn*cstarn*...
166     sqrt(1-cosd(alpstarn)^2-cosd(betstarn)^2-...
167         cosd(gamstarn)^2+2*cosd(alpstarn)*...
168         cosd(betstarn)*cosd(gamstarn));
169
170     a_neu=2*pi*bstarn*cstarn*sind(alpstarn)/Volstarn;
171     b_neu=2*pi*astarn*cstarn*sind(betstarn)/Volstarn;
172     c_neu=2*pi*astarn*bstarn*sind(gamstarn)/Volstarn;
173
174     V_neu=a_neu*b_neu*c_neu*sind(alpstarn)*...
175     sind(beta_neu)*sind(gamma_neu);
176
```

```
177     [gz_new,gxyz_new] = ...
           function_GZ_GXYZ_UVW(GSPEC,par_test(:,2),...
178         par_test(:,3),par_test(1,4),...
179         rec_par_optimized(1,:),rec_par_optimized(2,:),...
180         rec_par_optimized(3,:),rec_par_optimized(4,:),...
181         rec_par_optimized(5,:),rec_par_optimized(6,:),...
182         ind_test(:,3),ind_test(:,4),ind_test(:,5));
183
184     gxy_new=sqrt(((2.*pi)./(a_neu.*sind(gamma_neu))).^2.*...
185         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).^2+...
186         (par_test(:,2).*ind_test(:,1)./GSPEC).^2)+ ...
187         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*((ind_test(:,4)...
188         -par_test(:,3).*ind_test(:,2)./GSPEC).^2+(par_test(:,3).*...
189         ind_test(:,1)./GSPEC).^2)-...
190         2.*(2.*pi./(a_neu.*sind(gamma_neu))).*(2.*pi./...
191         (b_neu.*sind(gamma_neu))).*cosd(gamma_neu).*...
192         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).*...
193         (ind_test(:,4)-par_test(:,3).*ind_test(:,2)./GSPEC)+...
194         par_test(:,2).*par_test(:,3).*(ind_test(:,1)./GSPEC).^2)...
195         -((ind_test(:,3).*par_test(:,3)-ind_test(:,4).*...
196         par_test(:,2)).^2./GSPEC.^2).*(2.*pi./(a_neu.*...
197         sind(gamma_neu))).^2.*...
198         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*sind(gamma_neu).^2);
199
200     num=size(qxyz,1);
201
202     RMSD_qxy_new=(sum((gxy_new-ind_test(:,1)).^2)/num).^(1/2);
203     RMSD_qz_new=(sum((gz_new-ind_test(:,2)).^2)/num).^(1/2);
204     RMSD_qxyz_new=(sum((gxyz_new-qxyz).^2)/num).^(1/2);
205
206     A1_parameter_OPTIMIERT=[u,v,w,a_neu,b_neu,c_neu,...
207     alpha_neu,beta_neu,gamma_neu,...
208         V_neu,RMSD_qxyz_new,RMSD_qz_new,...
209         RMSD_qxy_new,abs(GSPEC-qspec)];
210 else
211     qxyz=sqrt(ind_test(:,1).^2+ind_test(:,2).^2);
212
213     [astern,bstern,cstern,alphastern,betastern,gammastern] = ...
214         function_REAL_TO_RECIPROCAL_v9( par_test(:,5),...
215          par_test(:,6), par_test(:,7),...
216          par_test(:,8), par_test(:,9),par_test(:,10));
217
```

```
218     [gz_uvw,gxyz_uvw] = ...
           function_GZ_GXYZ_UVW(GSPEC,u,v,w,astern,bstern,...
219         cstern,alphastern,betastern,gammastern,ind_test(:,3),...
220         ind_test(:,4),ind_test(:,5));
221
222     qg_feld_mit_hkl_uvw=horzcat(qxyz-gxyz_uvw,...
223     ind_test(:,2)-gz_uvw,...
224         ind_test(:,3),ind_test(:,4),ind_test(:,5));
225
226     [epsilon_vector_uvw] = ...
227         function_EPSILON_UVW(GSPEC,par_test(2),par_test(3),...
228         par_test(4),qg_feld_mit_hkl_uvw,astern,bstern,...
229         cstern,alphastern,betastern,gammastern);
230
231     rec_par_optimized=[astern;bstern;cstern;alphastern;...
232         betastern;gammastern]+epsilon_vector_uvw;
233
234     alpha_neu=acosd((cosd(rec_par_optimized(5))*...
235         cosd(rec_par_optimized(6))...
236         -cosd(rec_par_optimized(4)))/(sind(rec_par_optimized(5))...
237         *sind(rec_par_optimized(6))));
238
239     beta_neu=acosd((cosd(rec_par_optimized(4))*...
240         cosd(rec_par_optimized(6))-cosd(rec_par_optimized(5)))...
241         /(sind(rec_par_optimized(4))*sind(rec_par_optimized(6))));
242
243     gamma_neu=acosd((cosd(rec_par_optimized(4))*...
244         cosd(rec_par_optimized(5))-cosd(rec_par_optimized(6)))/...
245         (sind(rec_par_optimized(4))*sind(rec_par_optimized(5))));
246
247     astarn=rec_par_optimized(1);
248     bstarn=rec_par_optimized(2);
249     cstarn=rec_par_optimized(3);
250
251     alpstarn=rec_par_optimized(4);
252     betstarn=rec_par_optimized(5);
253     gamstarn=rec_par_optimized(6);
254
255     Volstarn=astarn*bstarn*cstarn...
256     *sqrt(1-cosd(alpstarn)^2-cosd(betstarn)^2-...
257         cosd(gamstarn)^2+...
258         2*cosd(alpstarn)*cosd(betstarn)*cosd(gamstarn));
259
```

```
260     a_neu=2*pi*bstarn*cstarn*sind(alpstarn)/Volstarn;
261     b_neu=2*pi*astarn*cstarn*sind(betstarn)/Volstarn;
262     c_neu=2*pi*astarn*bstarn*sind(gamstarn)/Volstarn;
263
264     V_neu=a_neu*b_neu*c_neu*sind(alpstarn)*...
265     sind(beta_neu)*sind(gamma_neu);
266
267     [gz_new,gxyz_new] = ...
           function_GZ_GXYZ_UVW(GSPEC,par_test(:,2),...
268         par_test(:,3),par_test(1,4),...
269         rec_par_optimized(1,:),rec_par_optimized(2,:),...
270         rec_par_optimized(3,:),rec_par_optimized(4,:),...
271         rec_par_optimized(5,:),rec_par_optimized(6,:),...
272         ind_test(:,3),ind_test(:,4),ind_test(:,5));
273
274     gxy_new=sqrt(((2.*pi)./(a_neu.*sind(gamma_neu))).^2.*...
275         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).^2+...
276         (par_test(:,2).*ind_test(:,1)./GSPEC).^2)+ ...
277         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*((ind_test(:,4)...
278         -par_test(:,3).*ind_test(:,2)./GSPEC).^2+(par_test(:,3).*...
279         ind_test(:,1)./GSPEC).^2)-...
280         2.*(2.*pi./(a_neu.*sind(gamma_neu))).*(2.*pi./...
281         (b_neu.*sind(gamma_neu))).*cosd(gamma_neu).*...
282         ((ind_test(:,3)-par_test(:,2).*ind_test(:,2)./GSPEC).*...
283         (ind_test(:,4)-par_test(:,3).*ind_test(:,2)./GSPEC)+...
284         par_test(:,2).*par_test(:,3).*(ind_test(:,1)./GSPEC).^2)...
285         -((ind_test(:,3).*par_test(:,3)-ind_test(:,4).*...
286         par_test(:,2)).^2./GSPEC.^2).*(2.*pi./(a_neu.*...
287         sind(gamma_neu))).^2.*...
288         (2.*pi./(b_neu.*sind(gamma_neu))).^2.*sind(gamma_neu).^2);
289
290     num=size(qxyz,1);
291
292     RMSD_qxy_new=(sum((gxy_new-ind_test(:,1)).^2)/num).^(1/2);
293     RMSD_qz_new=(sum((gz_new-ind_test(:,2)).^2)/num).^(1/2);
294     RMSD_qxyz_new=(sum((gxyz_new-qxyz).^2)/num).^(1/2);
295     A1_parameter_OPTIMIERT=[u,v,w,a_neu,b_neu,c_neu,...
296     alpha_neu,beta_neu,gamma_neu,...
297         V_neu,RMSD_qxyz_new,RMSD_qz_new,...
298         RMSD_qxy_new,abs(GSPEC-qspec)];
299 end
300 end
```