Reconstruction of Three-Dimensional Microstructures from Non-Destructive 3DXRD Microscopy Measurements

Development of a computational methodology

Appendix

A.C.P. van der Zijden

Section of Microstructural Control in Metals
Department of Materials Science and Engineering
Faculty of Mechanical, Maritime and Materials Engineering
Delft University of Technology

TU Delft
Delft University of Technology

MCM

Contents

1. **Introduction** **1**

2. **Software package** **3**

2.1. Global architecture 3
2.2. Pre-analysis 6
2.2.1. Spatial distortion reconstruction – SDCorrection.m 6
2.2.1.1. Translation process 7
2.2.1.2. Spline reconstruction 8
2.2.1.3. Distortion correction 9
2.2.2. Beam center determination – DetermineBC.m 10
2.3. Diffraction image visualization – FIT2D 11
2.4. Calling program – Analyze_4d.m 13
2.5. Peak detection – FindPeaks_4d.m, AnalyzeLayer_4d.m 14
2.5.1. Global outline 14
2.5.2. Diffraction ring definition 16
2.5.3. Intensity threshold 18
2.5.4. Mask computation 22
2.5.5. File read-in 24
2.5.6. Peak detection criteria 25
2.6. Spot characterization – ShellCheck_4d.m 27
2.6.1. Global outline 28
2.6.2. Non-electronic background correction 29
2.6.3. Initial settings 33
2.6.4. Box and shell construction; background intensity computation 34
2.6.5. Criterion evaluation 36
2.6.6. Spot characterization 37
2.6.7. Spatial distortion correction 38
2.7. Spot grouping – ArrangeSpots.m 43
2.7.1. Global outline 43

2.7.2. Initial settings 44
2.7.3. Locating matching spots 46
2.7.4. Grouping and numbering of spots 47
2.7.5. Center of mass, intensity profile computation 51
2.8. Reflection coupling – GrainSpotter 55
2.8.1. GrainSpotter input 56
2.9. Grain characterization – CharacterizeGrains.m 58

3. MATLAB code........ 60
3.1. SDCorrection.m 61
3.2. bispev.m 65
3.3. fpbisp.m 68
3.4. fpbspl.m 70
3.5. DetermineBC.m 71
3.6. readfrelon2k.m 73
3.7. Analyze_4d.m 74
3.8. FindPeaks_4d.m 76
3.9. AnalyzeLayer_4d.m 77
3.10. CreateMask2DTilt.m 85
3.11. ShellCheck_4d.m 87
3.12. ArrangeSpots.m 96
3.13. InputGrainSpotter.m 119
3.14. CharacterizeGrains.m 120

Bibliography........ 122

1. Introduction

This report forms the appendix to the M.Sc. thesis 'Reconstruction of Three-Dimensional Microstructures from Non-Destructive 3DXRD Microscopy Measurements'. That thesis describes the development of a computational methodology designed for the reconstruction of three-dimensional microstructures from three-dimensional x-ray diffraction microscopy data. It contains a relatively basic description of the layout and workings of the associated software package, and presents the results of application of the software to two available datasets.

However, in case one would like to apply this software package to any other datasets in the future, it is imperative the user has a thorough understanding of the structure of the package and especially of the procedures carried out by each individual subroutine. For such a detailed understanding, the author believes the description of the package contained in the thesis report does not suffice. This supplement aims at providing the necessary level of insight. It contains two main chapters: chapter 2 describes the software package, and chapter 3 contains the exact MATLAB code of the individual subroutines. The workings of the individual routines are clarified using numerical examples from the two datasets used in this theses research. For any details on such matters as x-ray diffraction theory, the experimental setup, the characteristics of the datasets used in this projects etc., the reader is referred to the main report.

Chapter 2 starts off by presenting a general overview of the package. Subsequently, the individual routines are treated one by one in a meticulous manner. Whereas the thesis report only mentions the main theoretical difficulties associated with each computational step, this supplement also discusses the way in which these issues have been solved in terms of coding, as well as the way in which certain peculiarities of the specific datasets under consideration have been handled. In general, chapter 2 contains some overlap with the software description provided in the main report, yet goes into much more detail when treating the individual subroutines. Numerical examples based on the datasets under consideration in this project are included. The detailed descriptions of the routines, together with the occasional numerical examples, should allow for this appendix to be used as a stand-alone manual for the developed software package. References back to the main report are made here and there, but these are not vital to understanding of the software.

Chapter 3 contains the exact code of the MATLAB routines written in this project. The code is given in MATLAB font, including the color designations used in the program, to allow for improved readability. The descriptions of the individual routines provided in chapter 2 will often refer to specific lines of code; these can then be retrieved from chapter 3.

2. Software package

This chapter provides an in-depth treatment of the software package that was outlined in chapter 4 of the thesis report. Section 2.1 restates the global architecture of the package. It identifies the different steps required for the microstructure reconstruction, and introduces the programs written to execute these various tasks. The rest of the chapter provides detailed discussions of the individual routines. Based on the code of the programs, most of which is included in chapter 3, the reader is guided through the entire package while being explained what operations are being performed.

For the sake of completeness, this chapter discusses each individual part of the software package, including some which have already been discussed in considerable detail in the main report. In some cases, this results in significant overlap with the contents of the thesis report. The author felt this overlap was justified, however, since it allows the more experienced 3DXRD experimentalist to use this appendix as a stand-alone manual of the software package. Numerical examples based on the datasets analyzed in the main report are included at several locations in this chapter.

2.1. Global architecture

This section explains the basic outline of the newly written software package. It is mainly a reiteration of the corresponding section of the main report.

First of all, a small word is required on the terminology used in the remainder of this appendix. The reader should be aware of the distinction made between the terms 'reflection', 'spot' and 'peak'. A reflection is defined as the diffraction event from a specific grain within the sample; the locations of these reflections are determined by the crystallographic orientation of the grain. By definition, for any reflection the entire grain obeys the Bragg criterion and diffracts. However, due to the small beam sizes used in this research, it is quite unlikely that any grain in the sample will be fully illuminated at any time. Therefore, when the Bragg criterion is fulfilled, not the entire grain but instead only parts thereof will diffract. The results of these partial diffraction events are termed spots; they are the actual intensity objects visible within the diffraction images. In other words, a single reflection will often manifest itself as multiple spots in the diffraction images. Furthermore, in the ideal case ('ideal' meaning Lorentzian diffraction spots), each spot is represented by a single peak: the pixel within that spot with the highest intensity. A peak is defined as a single pixel which has an intensity higher than a certain threshold value and which forms a maximum with respect to all its nearest neighbors in (x,y,ω)-space.

The software was written in MATLAB, version 7.0.4. MATLAB, short for MATrix LABoratory, is a popular software system especially suited for matrix computations. It is produced by The Mathworks, a U.S.A.-based company [2]. The reasons for MATLAB being the programming environment of choice were twofold. First of all, the recorded diffraction patterns are nothing more than large matrices of pixels containing a single value (the number of counts) each. Therefore, the fact that MATLAB is optimized for matrix computations would be a very useful feature. And secondly, most of the already existing software was written in MATLAB. So, compatibility would be much less troublesome if new programs would also be written in MATLAB.

Figure 2.1 repeats the flow chart for the software package created for the three-dimensional microstructure reconstruction. It depicts how the raw data, in the form of a large amount of diffraction patterns, are transformed into a reproduction of the original microstructure. First of all, the diffraction patterns are read in and scanned for peaks in intensity. This produces a list of the positions of these peaks in terms of in which image the peaks have been found and on which exact pixel. This list is then carried over to the next step, where for each peak its exact dimensions are determined. Now that for each peak its dimensions and hence its total intensity are known, the resultant list can be seen as an enumeration of all the spots in the analyzed diffraction patterns. Subsequently, these spots are grouped together, combining peaks that come from different parts of the same grain. After all, since the beam is only 15 μm wide in its smallest dimension, it is quite conceivable that a certain reflection of a specific grain will in fact show up at multiple subsequent slit positions. When this grouping has been carried out, spots that originate from different parts of the same grain at the same diffraction angle have been matched, and such a group of spots together forms a single reflection of that specific grain. In other words, the result of the grouping of the spots is a list of the reflections coming from the grains in the gauge volume. These reflections all have associated center of mass locations, total integrated intensities etc.

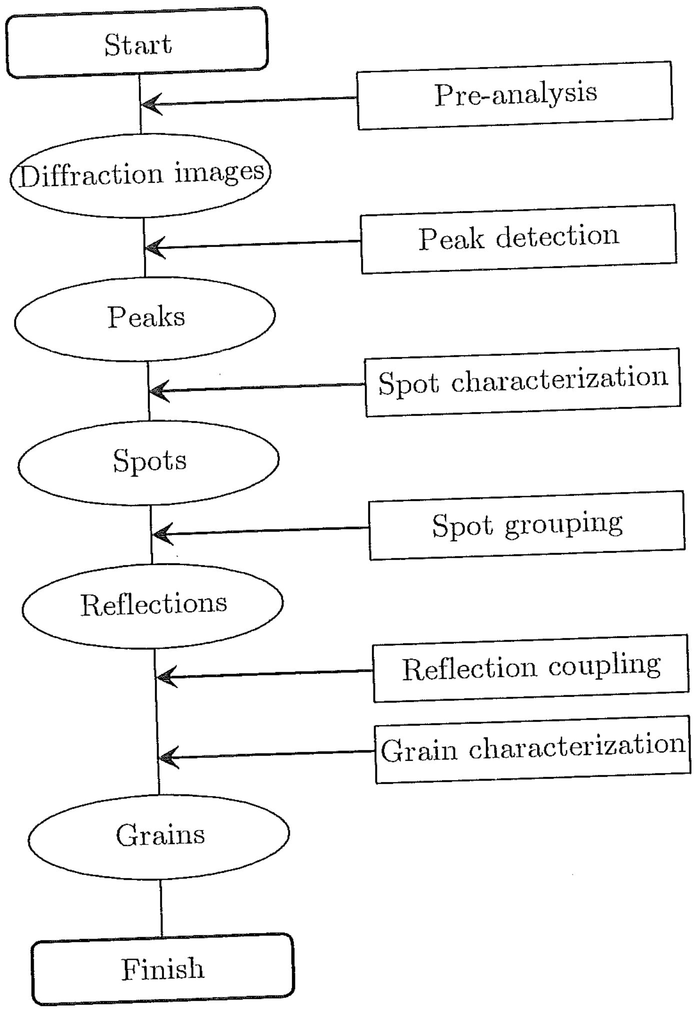


Figure 2.1: Flow chart for the package of MATLAB routines written for the three-dimensional reconstruction of a polycrystalline microstructure from x-ray diffraction data. The diffraction patterns are transformed in a stepwise manner into a reproduction of the original microstructure by the various operations listed on the right.

The final step is the coupling of the individual reflections to real grains. This is done by the matching of different reflections that originate from the same grain. On the basis of the crystal symmetry of the phase under consideration (in the case of the data investigated in this thesis: austenite), given a certain reflection, one can predict where (in ω-space) the other reflections originating from the same grain should lie. This step is performed by GrainSpotter [3]. This is a piece of software which was created at the Risø National Laboratory in Denmark and was based on an earlier software package called GRAINDEX [4]. GrainSpotter's output is a list of groups of reflections, with each group corresponding to a real grain. By combining the centers of mass of the individual reflections, the center of mass of the corresponding grain can then be computed.

Now that the global architecture of the software package has been outlined, the following sections will present in-depth treatments of the individual programs. This is done on the basis of the routines written for the *d*-set, since the *d*-series is slightly more complicated to analyze than the *e*-series. If significant differences exist between the *d*- and *e*-versions of a specific routine, these differences are highlighted in the corresponding section. If no mention is made of such differences, the routines are nearly identical and the author felt no need to pay the *e*-version any individual attention.

2.2. Pre-analysis

The pre-analysis part of the reconstruction serves to determine some parameters required for subsequent analysis of the diffraction images. Figure 2.2 depicts a flow chart of the pre-analysis process. The pre-analysis can be seen to consist of five main tasks. All five tasks provide input which is required for the correct interpretation of the diffraction images. Furthermore, the output of some of the tasks is required as input for some of the other operations; these dependencies are the main determinants of the exact order in which the individual processes are carried out.

Note that not for all of the operations listed in Figure 2.2 new software needed to be constructed. In fact, this section of the appendix describes only two of the pre-analysis processes into more detail; the other processes have been carried out using already existing software, most notably a piece of software called FIT2D, available from the ESRF [5]. Firstly, the routines written to describe the spatial distortion present in each diffraction image are outlined. The second subsection describes the routine written for the determination of the location of the beam center.

2.2.1. Spatial distortion reconstruction – SDCorrection.m

The correction for the spatial distortion introduced by the detector optics was previously performed within FIT2D. It was manually applied to the entire diffraction pattern, after which the corrected image could be saved again. Though useful for analyses involving only a limited number of images, clearly this procedure becomes impracticable when the number of images becomes larger. The original FIT2D code was therefore translated into several MATLAB routines. For more information on the distortion correction within FIT2D, the reader is referred back to the main report. Suffice it to say here that FIT2D uses two bivariate splines of the third degree to approximate the horizontal and vertical distortion of each pixel on the detector's surface.

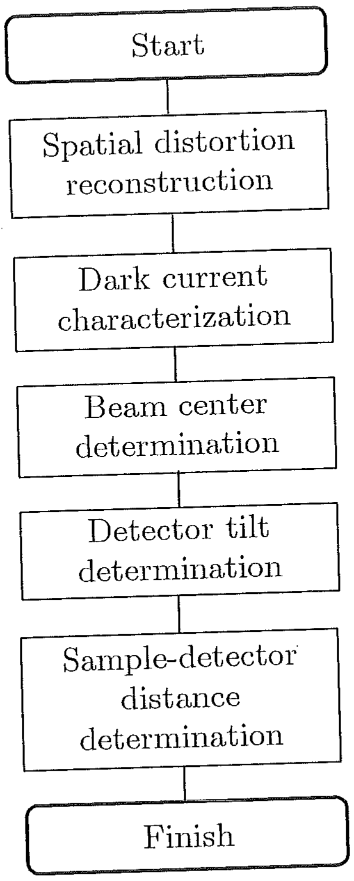

Start
Spatial distortion reconstruction
Dark current characterization
Beam center determination
Detector tilt determination
Sample-detector distance determination
Finish

Figure 2.2: Flow chart of the pre-analysis part of the microstructure reconstruction procedure. The results from the various processes listed in the chart are required for the subsequent analysis of the diffraction patterns.

2.2.1.1. Translation process

Upon inquiry, the FIT2D routine turned out to be written in a programming language called Python. The Python program, in turn, mainly acted as a handle for some routines from the Fortran package FITPACK, a freely available package specifically written for the calculation of smoothing splines (also called DIERCKX, after its author) [6]. It was decided to translate this Fortran code into MATLAB routines, and write a MATLAB handle for these routines on the basis of the original Python one.

For the Fortran-to-MATLAB conversion the package f2matlab (version 1.90) was used, which is a package of MATLAB routines that translates Fortran90 subroutines or functions into MATLAB m-files [7]. The Fortran code used for FITPACK, however, was Fortran77, and could not be used directly as input to f2matlab. It was therefore first translated into Fortran90 using the freely available converter by Alan Miller, to_f90 [8]. After a few manual alterations (mainly required because to_f90 could not handle DO...END DO statements) the resulting F90 routines were translated into m-files by f2matlab. A new routine, SDCorrection.m, was written to provide easy access to this set of subroutines. The code of SDCorrection.m is given in the appendix and starts on page 61.

2.2.1.2. Spline reconstruction

The input to SDCorrect.m consists of the full location of the splinefile (the file containing all the spline coefficients as explained in the previous subsection). This location is used in line 28 of the MATLAB code to open the corresponding file. Lines 23 and 24 have already declared the degrees of the splines in both directions. These degrees are hard-coded, and need to match the degrees of the splines as they were originally constructed when the spatial distortion characterization of the detector setup was performed. In this case, the splines were of the third degree in both dimensions.

The splinefile that is opened in line 28 has a very specific format. Therefore, the analysis of this splinefile can be performed by simply running over the lines of the file and picking up the required data at the right points. Lines 38 through 41, for instance, effectively skip the first three lines of the splinefile. After line 41 has read the fourth line, this line is analyzed further because this line is expected to contain some important parameters. Indeed, by using the str2num command (line 42), the values are read and subsequently assigned to the appropriate variables (lines 43 through 46); in this case, they define the area for which the spline is valid, in other words the size of the detector. Comparable operations are performed up till line 60, defining grid spacing (the distance in real space between the holes in the mask), the pixel sizes of the detector (these follow from combining the grid spacing with the average number of pixels between two hole projections on the image), and the number of knots (in both dimensions) of the spline describing the distortion in the x direction. After the amount of knots has been declared within the splinefile, their positions are given. This is done in a block of lines with each line containing a maximum of five values. Line 67 determines how many lines this is for the knots in FIT2D's x direction, by dividing the number of knots by 5 and rounding upwards. These lines are subsequently read and the values are written into *tx1* (lines 68 through 72). The procedure is repeated for the knots in the y direction. After the positions of the knots have been declared, the values of the spline coefficients are given. Since the spline is of the third degree in both dimensions, the number of spline coefficients (*nc1*) follows from the number of knots in the x direction (*nx1*) and in the y direction (*ny1*) as $nc1 = (nx1 - 4)(ny1 - 4)$ [9]. Once this number is computed (line

81), the values of the coefficients are transferred to the matrix *c1*. Subsequently, the entire sequence is repeated for the spline to the y direction distortion (lines 88 through 116).

Now that the spline coefficients have been written into the two matrices *c1* and *c2*, the original splines can be reconstructed. This is done in lines 130 and 131. Each of these lines calls the subroutine bispev.m. This routine is a translation of the bispev.f routine from FITPACK [6]; its code is given in the appendix, starting on page 65. The routine is called once for each spline. Its input is the locations of the knots in both dimensions (*tx1*, *tx2*), the value of the spline coefficients (*c1*), the degrees of the spline (*kx*, *ky*) and two matrices defining the area over which to compute the spline. The latter are designated x and y, corresponding to the two directions defined earlier. The combination of the two defines the detector area of interest. Since in this case the spatial distortion of the entire detector is desired, x and y are both set to $x = y = [1 \ 2 \ .. \ 2047 \ 2048]$ so that together they define the entire 2048×2048 pixel detector area.

The bispev routine contains some artifacts from the original Fortran code, related mainly to the allocation of the workspaces and the checking of the input data. These have been left in since they take hardly any computational time. The function of bispev is to call another subroutine called fpbisp (lines 120-121), which is also a translation of a Fortran routine from FITPACK by the same name. The code of fpbisp is given on pages 68 and 69. The routine computes the desired spline by executing two calls to yet another subroutine, fpbspl.m, and subsequently meshing the results of the two together. The fpbspl routine (page 70) contains a short algorithm computing a spline on a single interval using the stable recurrence relation of Cox and De Boor [10, 11]. Together bispev.m, fpbisp.m and fpbspl.m perform the main tasks of the spatial distortion correction previously carried out within FIT2D: the computation of the distortion in both directions for all pixels on the detector.

2.2.1.3. Distortion correction

Returning to SDCorrection.m, lines 130 and 131 produce the two matrices *deltaj* and *deltai*, which contain the distortion of each pixel in FIT2D's x and y direction, respectively. Note the correlation between FIT2D's (x,y)-directions and the (**i**,**j**)-directions in the matrix representation of a diffraction image. FIT2D defines its x direction as lying along the horizontal and y along the vertical, whereas in the matrix representation of a diffraction image the **i** direction corresponds to the fast, vertical index, and **j** to the slow, horizontal index. Hence, the distortion matrix computed in line 130 with the knots lying along FIT2D's x direction (line 130) is called *deltaj* and not *deltai*, and vice versa.

Subsequently, the distortion matrices are used to compute the corrected diffraction images (lines 135 through 158). This is done by constructing a $(2048^2,6)$-sized matrix termed *SDMatrix_full*, which contains for each pixel its location as recorded (columns 1 and 2), its distortion in both directions (3 and 4), and its undistorted location (5 and 6). The matrix is stored as a tab-delimited ASCII file. The routine finishes by trimming *SDMatrix_full* to get rid of entries that will not be used because they lie beyond the rectangle enclosing the outermost ring of interest. In this way, the size of the correction matrix is reduced significantly, speeding up the analysis. Lines 169 through 172 describe the perimeter of this rectangle. They contain the row and column indices of the outermost rows/columns that are of interest to the analysis. Lines 176 through 187 subsequently get rid of the unnecessary entries. See also section 2.5, page 23 for more on this reduction of the peak search area. The resulting matrix, *SDMatrix*, is written to disk.

SDCorrection.m therefore produces two matrices which are both saved on disk: a matrix *SDMatrix_full* containing the spatial distortion and the undistorted coordinates of all the pixels on the detector, and matrix *SDMatrix* which only contains the entries of the pixels of interest. The former could be useful when one would like to check whether the spatial distortion computation has been performed correctly, by comparing a plot of the corrected pixels and their intensities against a corrected image from FIT2D. The latter is more suitable for the actual on-line analysis, since the reduced size of this file offers significant speed-up in computational time.

2.2.2. Beam center determination – DetermineBC.m

As described in the corresponding section of the main report, the location of the beam center is determined by establishing the weighted average beam center coordinates on the basis of the direct beam mark. The routine written to this end is called DetermineBC.m. Its code starts on page 71.

The input to DetermineBC is called 'image' and should represent the location of the LaB_6 diffraction pattern. Lines 22 through 25 of the routine's code define the estimated location of the direct beam mark, and the size of the box over which the weighted average will be computed. The values are given in FIT2D x and y coordinates. The first estimate of the beam center, $(xest, yest) = (981,1011)$, is taken as the pixel within the direct beam mark with the highest intensity. The size of the box was taken large enough to contain all of the beam mark intensity, but small enough to lie entirely in the third quadrant of the detector surface, so that the (minor) effect of the difference in background intensity between the different quadrants of the detector did not play a role (for more on this, see subsection 5.2.2 of the main report). Lines 27 through 32 convert the FIT2D coordinates to matrix indices m and n.

The next step is the computation of the dark current intensity. The dark current is a near-constant electronic background to the diffraction images, present even when no sample is mounted and when the beam' shutters are closed, which should be subtracted from the intensities. The dark current intensities are determined by averaging 22 dark current measurements. The corresponding files are read (line 39) using a small MATLAB-routine called readfrelon2k.m that was already written in an earlier project (the code of which is also included in chapter 3). These 22 specific measurements (file-numbers 26 through 47) have been used because they were recorded using the same exposure time as the d dataset: one second. For the e dataset, the exposure time was only half a second; in this case, dark current images with half a second of exposure time were used. Each of the 22 images was recorded at a specific ω-setting; however, since no sample was present the value for ω should not have had any influence on the intensity. To subtract the dark current intensities as accurately as possible, the pixels in the LaB_6 image are corrected pixel for pixel by subtracting the average dark current intensity found for that specific pixel (line 48). This also corrects for a phenomenon known as 'hot pixels': faulty pixels which register a constantly elevated value for the intensity. Since this defect is independent of whether or not a sample has been mounted, the dark current intensity of such a pixel will show the same increase in intensity, and therefore on subtraction of the dark current intensity the hot pixel will be neutralized.

After the LaB_6 image is corrected for the dark current intensities, the refining of the location of the beam center commences. The beam center is determined by computing a weighted average of the row and column indices of the pixels within the box defined at the start of the routine, using the intensities as weights. The final beam center location, (*mBC*,*nBC*), is returned as the routine's output. This value can be used in the rest of the analysis as the definitive (albeit still uncorrected for spatial distortion) beam center location. Furthermore two vectors *rowprof* and *colprof* are created, which contain the average intensity per pixel for each row and column, respectively. These can be plotted to obtain an idea of the variation of the intensity with horizontal and vertical position.

2.3. Diffraction image visualization – FIT2D

The already mentioned FIT2D is a piece of software available from the website of the ESRF [5]. Multiple versions of the program can be downloaded there; the version used in this project is v12.077. FIT2D can be used for various tasks; its main use in the current project was the visualizing of the diffraction images. The data collected by the Frelon2K camera were stored as .edf-files, a file-type used only by the ESRF. FIT2D is able to translate these files into images, plotting the intensities against their horizontal and vertical positions and thus creating a reconstruction of the diffraction patterns. Figure 2.3 shows an example of such a reconstructed diffraction pattern. This specific pattern was recorded at stripe 0, layer 0, ω=-17°, with beam dimensions of 15×100 μm^2 (dataset *d*). The grayscale of the picture has been inverted to allow for easier spot identification. Spots now appear as dark marks on a lighter background.

The diffraction pattern of Figure 2.3 shows some interesting characteristics. First of all, the spots can be seen to lie on circles that center near the middle of the detector, as follows from the application of Bragg's law to the experimental situation at hand. Furthermore, the regions of the diffraction image where no peaks are found obviously display some considerable background intensity. Clearly, this background needs to be taken into account when performing a search for peaks. The background intensity does not appear to be constant over the entire detector; for instance, Figure 2.3 shows a difference between the background intensity on the left-hand side and on the right-hand side of the detector. Though this difference is quite pronounced in the

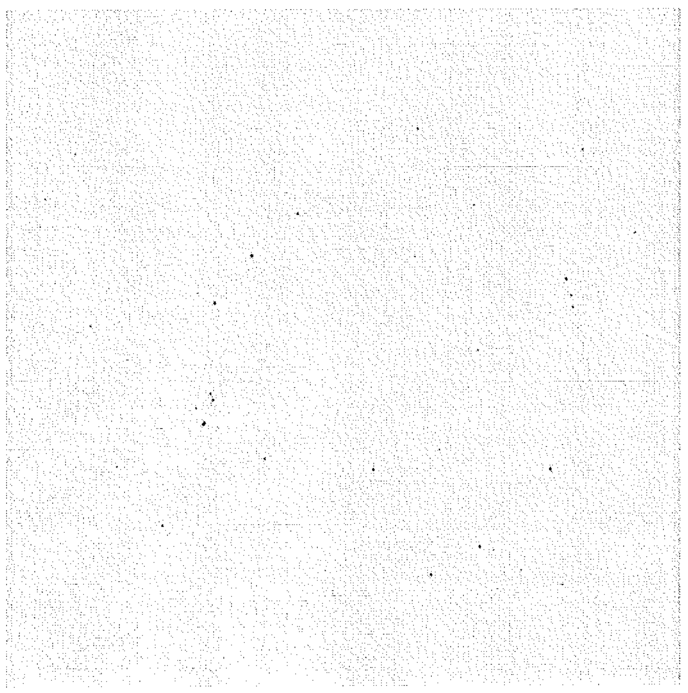


Figure 2.3: Example of a diffraction pattern reconstructed using FIT2D. This pattern corresponds to stripe 0, layer 0, ω = -17°. For easier spot recognition, the grayscale has been inverted. Notice the diffraction spots lie on circles approximately around the center of the detector. The difference in color between the left-hand side and right-hand side of the image is indicative of a small difference in background intensity between the two. This is caused by a software anomaly.

figure due to the intensity scaling, the effect is only in the order of a few counts, i.e. in the order of 0.1 % of the average background intensity. It is probably caused by a software anomaly of the Frelon2K detector. Still, given the presence of this non-constant background, the newly written software was required to be able to correct the diffraction images for this effect.

2.4. Calling program – Analyze_4d.m

This section presents the main routine of the analysis of the *d*-dataset, called Analyze_4d.m. The code of Analyze_4d is presented in chapter 3 (starting on page 74). Analyze_4d is nothing more than a convenient way of calling all subroutines in the correct order. The individual routines are treated in depth in the subsequent sections. Therefore, the following treatment of Analyze_4d will be relatively brief.

The various subroutines deal with most of the aspects of the analysis as depicted in Figure 2.1. The first process handled within Analyze_4d is the detection of the peaks within the individual diffraction images. This is done using the routine FindPeaks_4d.m, on line 21 of the code. The result of this procedure is a list called *PeakList* containing all pixels that have been identified as peaks: having an intensity above a certain threshold value, and also being a maximum with respect to all their nearest neighbors. Ideally, each spot within the diffraction images is represented by a single entry in this list. Alternatively, line 22 can be used to retrieve a user-defined *PeakList*; this possibility is useful in case the peak detection has already been performed at an earlier stage.

After peak detection, the dark current intensities are computed and the file containing the required spatial distortion corrections is read. When these actions have been performed, the routine moves on to the third step as indicated in Figure 2.1: the characterization of the individual spots. This is done using the routine ShellCheck_4d (lines 64-65). The spots are identified and characterized in terms of total intensity, diffraction angle etc.

Due to the scanning in the ω-direction and the overlap of subsequent slit settings, individual reflections are likely to be split up into multiple diffraction spots. So, when all spots have been characterized, those spots which are only partial reflections are searched and grouped together so that the resultant list is one containing only complete reflections. This is performed by the routine ArrangeSpots. When this routine is finished, a list is returned of all complete reflections with their associated total intensities, center of mass locations etc.

As described in section 2.1, the matching of reflections that originate from the same grain is not done within Analyze_4d, but instead by a separate piece of software called GrainSpotter. A description hereof is included in this chapter. Finally, the software related to the grain characterization is presented.

2.5. Peak detection – FindPeaks_4d.m, AnalyzeLayer_4d.m

The Frelon2K CCD camera used for collection of the data of the diffraction patterns stores these patterns as two-dimensional matrices, each entry being the number of counts of the corresponding pixel. A diffraction peak will reveal itself as a significantly higher number of counts on a specific pixel. Therefore, the first step of the analysis is to scan the diffraction patterns for local maxima in the number of counts recorded by the camera's pixels. This is done by the combination of the two routines FindPeaks_4d.m and AnalyzeLayer_4d.m.

FindPeaks_4d is a small routine that repeatedly calls AnalyzeLayer_4d; the latter is responsible for the actual detection of the maxima. This construction was chosen so as not be restricted to having to analyze the entire *d* dataset at once; by having the actual analysis done on a single layer, and continue performing this analysis on subsequent layers until the entire set was analyzed the possibility remained to consider only a subset of the data. This could be useful for instance when one would like to carry out a quick check of the data or of the software, or when one is interested only in a specific part of the gauge volume. The code of FindPeaks_4d is given on page 76; AnalyzeLayer_4d starts on page 77.

2.5.1. Global outline

As mentioned above, FindPeaks_4d is only a shell that repeatedly calls the actual analysis routine. The amount of code, therefore, is limited. The routine starts by computing the dark current intensities in the same way as is done in Analyze_4d – by averaging the intensities from 22 separate dark current images. The code could have been written so as to carry over the dark current intensities computed in Analyze_4d into FindPeak_4d, removing the need for recomputing them there. However, by having them recomputed the possibility remains to operate FindPeaks_4d as a stand-alone routine, without having to call Analyze_4d. This is useful for instance when one is only interested in a first estimate of the number of peaks. The dark current computation is contained in lines 19 through 25. After the dark current has been computed, the actual peak search is initiated. First of all the matrix *Peaks* is initialized, which will eventually contain the output of the program. Then a double loop is started over all three stripes (numbered 0 through 2) and all layers (50 per stripe, numbered 0 through 49). For each combination of stripe and layer, line 35 calls AnalyzeLayer_4d and collects its output as a matrix *u*. This matrix is then appended to *Peaks*, which in this manner is filled stepwise with the output of all the different calls of AnalyzeLayer_4d. Finally, line 42 writes *Peaks* away into a tab-delimited text file at the specified location.

AnalyzeLayer_4d performs most of the work in the peak search part of the analysis. It is partly based on routines written at an earlier stage by dr. Enrique Jimenez-Melero of the Reactor Institute Delft. Figure 2.4 shows a flow chart of the routine. After commencing by declaring some settings, masks are created that cancel all data that are not of interest to the current analysis. A loop is then started over all images corresponding to the layer under investigation. Each image is read, after which the peak search is executed in two steps – one for each diffraction ring. The results of the search are appended to the list of peaks identified earlier, thus creating a single list containing all peaks for this layer.

The following subsections provide in-depth treatments of the various processes identified in Figure 2.4. The input variables to the routine are the stripe number and layer number, which together define the slit setting of interest. Furthermore, the matrix of dark current values is carried over from FindPeaks_4d.m. For AnalyzeLayer_4d to be used as a stand-alone program, one would only have to perform a simple cut-and-paste operation of lines 19 through 25 of FindPeaks_4d, in

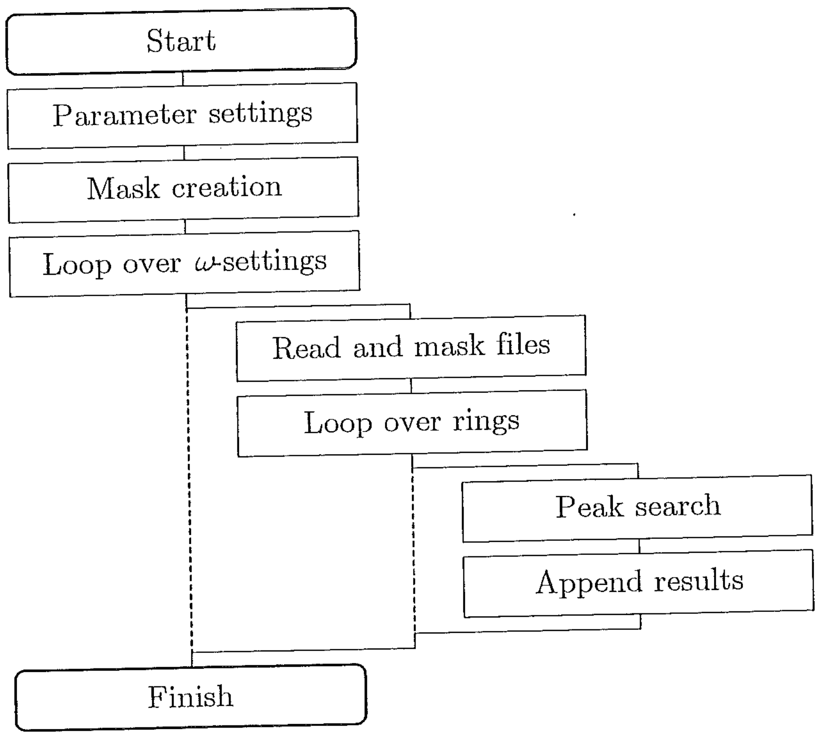


Figure 2.4: Flow chart of the peak detection process. The routine searches for diffraction peaks within the images recorded for a specific slit setting. This is done by looping over ω and over the diffraction rings, and then performing a peak search for each iteration.

combination with some minor adjustments in the routine's code, to include the dark current computation in the routine itself.

2.5.2. Diffraction ring definition

The routine commences with a quick check of the input values for the stripe and layer number (lines 15 through 23). If these are not within the expected ranges, an error message is displayed and control is passed back to the calling program. After this, some parameter settings are declared. These values require changing when a different material or dataset is analyzed. Lines 28 through 31 declare the minimum and maximum diffraction angles for diffractions from the *{200}* and *{220}*-planes. The theoretical diffraction angles follow from the Bragg criterion. For the steel under consideration in this thesis project, at a temperature of 1273 K, the austenite lattice parameter and the diffraction angles are $a_\gamma = 3.6579\,\text{Å}$, $2\theta_{200} = 4.9\,°$ and $2\theta_{220} = 6.9\,°$, respectively. Based on visual inspection of the diffraction rings (using FIT2D), a band around these angles was defined to allow for small deviations of the theoretical angle. Effects contributing to the formation of a range of observed diffraction angles for a specific *{hkl}*-family include the incoming beam not being monochromatic, the beam being slightly divergent, and the occurrence of local deviations in lattice parameter.

Figure 2.5 illustrates the influence of yet another effect: the intrinsic assumption of a point-sized sample. It displays a top view (simplified and not to scale) of the diffraction process. Note that strictly speaking the front and back side of the sample were slightly curved, since the sample was cylindrical, but this curvature is omitted in the illustration. Using a single value for the sample-detector distance L_{sd} and the location of the beam center *b.c.* effectively introduces the assumption of a point-sized diffraction source, since it fixes the origin of the diffracted beams to a single point (the center of the illuminated volume, as shown in the figure). In reality, however, the diffracting grain can be located anywhere within the illuminated volume, thus introducing an error of half the length/height/width of the volume in each of the corresponding dimensions. Figure 2.5 depicts (albeit two-dimensionally) the situation for which the error is largest: when in reality the diffracting grain lies in the corner of the illuminated area instead of in the middle. The diffraction spot is an extra ΔR further away from the beam center, and therefore when the diffraction angle is calculated this leads to a higher value of $2\theta'$ instead of 2θ. An estimation of this error can be made in the following manner.

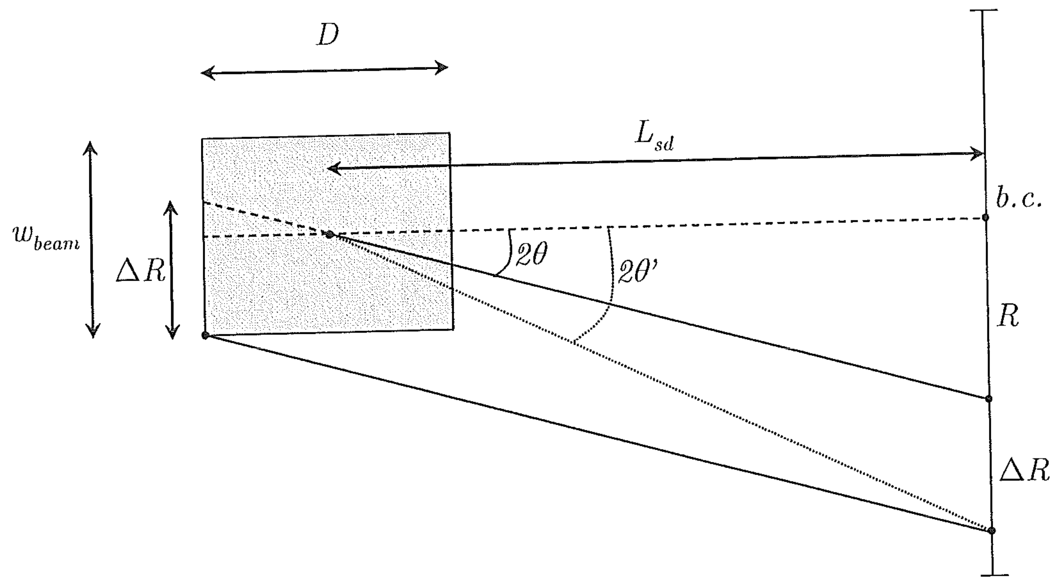


Figure 2.5: Outline of how the assumption of a point-sized diffraction source leads to errors in the diffraction angle. The assumption is introduced implicitly by taking a single value for the sample-detector distance L_{sd} and the beam center location $b.c.$ If a grain is not located in the center of the illuminated volume but in one of the corners, this leads to an increase in the distance from the beam center of ΔR, translating into a deviation of the measured diffraction angle of $\Delta(2\theta)=2\theta'-2\theta$.

The value of ΔR can be seen to equal:

$$\Delta R = \tfrac{1}{2} w_{beam} + \tfrac{1}{2} D \tan(2\theta) \tag{2.1}$$

in which w_{beam} represents the width of the beam, and D represents the diameter of the sample. The expression for the observed diffraction angle $2\theta'$ is:

$$\tan(2\theta') = \frac{R+\Delta R}{L_{sd}} = \tan(2\theta) + \frac{\Delta R}{L_{sd}} \tag{2.2}$$

For small angles, a straightforward Taylor expansion shows that the tangent of an angle is approximately equal to the angle itself (this requires the angle to be expressed in radians instead of in degrees). For instance, for an angle $\alpha = 10°$, we have:

$$\tan\left(\alpha[°]\right) = \tan\left(\alpha \times \frac{\pi}{180}[\text{rad}]\right) = \tan(0.175[\text{rad}]) = 0.176 \approx \alpha[\text{rad}] \tag{2.3}$$

Given the fact that the derivative of tan(α), $[\tan(\alpha)]' = 1 + \tan^2(\alpha)$, increases continuously from 1 to approximately 1.03 over the interval $0° \le \alpha \le 10°$, it follows that the assumption of $\tan(\alpha) = \alpha$ is a valid one on this interval.

Using this assumption, the difference between the computed diffraction angle and the theoretical diffraction angle, $\Delta(2\theta)$, can be expressed as:

$$\Delta(2\theta) = 2\theta' - 2\theta = \frac{\Delta R}{L_{sd}} = \frac{w_{beam} + D\tan(2\theta)}{2L_{sd}} \tag{2.4}$$

The amount of deviation therefore depends on the ring of interest (reflected through the tan(2*θ*)-term), the dimensions of the beam and sample, and the sample-detector distance. For the experimental settings of the *e*-series (cylindrical sample with diameter 1 mm., beam width 300 μm.), the {*220*} diffraction ring (which is associated with a 2*θ*-angle of 6.9° or 0.12 rad), and a sample-detector distance of 241 mm. (for the exact derivation of this value, see the main report), equation (2.4) predicts a systematic error in the observed diffraction angle of about 0.87 mrad or 0.05°. So, for these experimental settings the implicit assumption of a point-sized diffracting volume introduces a total bandwidth of 0.1°. Add to this the influence of the other broadening effects mentioned above, and it becomes clear that the visually determined bandwidth of a few tenths of a degree in either direction can well be explained on a physical basis. Note that this analysis also holds for the *d*-series, in which case Figure 2.5 represents a side view instead of a top view.

2.5.3. Intensity threshold

Line 57 of AnalyzeLayer_4d lays down the intensity threshold for this particular dataset. That is, the intensity from a pixel is required to be larger than this threshold value in order for the pixel to be considered a peak. In this case, this threshold equals twice the square root of the average dark current intensity:

$$I_{Min,d} = 2\sqrt{\langle I_{DC}\rangle} \tag{2.5}$$

in which $I_{D.C.}$ represents the dark current intensity. The average of this intensity is calculated over all pixels in the *DarkCurrent* matrix.

The justification of the criterion described by equation (2.5) is as follows. A criterion of the form used in this routine is usually built around an expression of the form

$$I_{Min} = n\sigma_{BG} \tag{2.6}$$

in which I_{Min} represents the threshold and σ_{BG} the standard deviation in the background intensity. Generally, the arrival of background counts is modeled using a Poisson distribution [12]. When the average value of such a Poisson distribution is high enough (say, larger than 10), this distribution can be approximated using a normal distribution with mean and variance equal to the average of the Poisson distribution [13]. This implies that equation (2.6) can be rewritten as

$$I_{Min} = n\sqrt{\langle I_{BG} \rangle} \tag{2.7}$$

with $<I_{BG}>$ representing the average background intensity. A value for n of 2, for instance, implies that about 2.3% of the pixels will have a background intensity higher than the threshold value [12]. For a value of 3, this fraction drops to only 0.1%.

The average background intensity is computed by defining two background rings. In this manner, if a dependence of the background intensity on the distance from the beam center exists this should be visible from the data. The first and second background ring are located between the austenite's {*200*}- and {*220*}-, and {*220*}- and {*311*}-ring, respectively, implying that no diffraction spots are expected within these rings. Therefore, by determining the average intensities of these rings the average background intensity of the dataset can be determined, and hence the threshold criterion of the form of equation (2.7) can be constructed. The value of n is determined by a trial-and-error type process, in which 2 is taken as the starting value (a value commonly used for these types of criteria) which can subsequently be refined based on the amount of pixels incorrectly identified as peaks.

Note that equation (2.5) implies that the only contribution to the background intensity comes from the dark current. In general, there will also exist a non-electronic background contribution. This contribution comes from effects like thermal scattering of the x-rays, and is expected to be dependent on such parameters as beam dimensions, beam current within the synchrotron storage ring, and temperature. For the d dataset, however, this contribution turned out to be negligible. The background intensities for the diffraction images of the d dataset were computed by defining two background rings per image (defined using minimum and maximum 2θ-values: $5.5 \leq 2\theta \leq 6.5$ and $7.5 \leq 2\theta \leq 7.8$ for background rings 1 and 2, respectively). Since these rings lie well outside of the range of the austenite diffraction rings as given by the Bragg criterion and listed at the start of the routine, the only intensity expected within these rings is background intensity. Computation of the average background intensity within such rings constantly produced values within one or two counts of the dark current intensities. Table 2.1 lists the average intensities of the two rings,

$<I_{BG1}>$ and $<I_{BG2}>$, for the diffraction images from the *d* dataset, together with the average dark current intensity $<I_{DC}>$, averaged over all pixels from all the 22 dark current images with an exposure time of 1 second. It can be seen that the average values of the intensities for the *d*-series background rings are within only a single count of the average dark current intensity per pixel. Therefore, the threshold intensity for a peak in the *d* measurement as described by equation (2.5) can be based solely on the average dark current intensity.

Table 2.1: Average intensities for the background rings in the two datasets, $<I_{BG1}>$ and $<I_{BG2}>$, compared to the average intensity of the dark current images used for correction of the respective dataset, $<I_{DC}>$. Whereas the values for the *d*-series are nearly equal, the *e*-series appears to contain a significant contribution from the non-electronic background.

	$<I_{B.G.1}>$ (# counts)	$<I_{B.G.2}>$ (# counts)	$<I_{D.C.}>$ (# counts)
d-*series*	1001.0	1001.1	1000.3
e-*series*	1019.3	1020.7	1000.4

Table 2.1 also lists the average background intensities for the *e*-series diffraction images together with the average dark current intensity, the latter being computed using all dark current files with an exposure time of 0.5 second. In contrast to the *d* measurement, for the *e*-series the values do differ significantly, so there is a clear contribution of the non-electronic background to the total average background intensity. This difference between *d* and *e* could be caused by the fact that the illuminated area for the *e*-series is three times as high as for the *d*-series, leading to increased scattering and consequently a higher background level. Note that the average background intensity of the inner background ring, $<I_{BG1}>$, is somewhat smaller than the average intensity of the outer background ring, $<I_{BG2}>$. In other words, the thermal scattering of the x-rays appears to increase slightly with increasing radial distance from the beam center. This phenomenon can be understood in light of the increased thermal scattering with increasing diffraction angle, visible also in the $\sin(\theta)$-term in the expression for *M* in the Debye-Waller factor [14] (see subsection 3.1.3 of the main report).

Due to the non-electronic background contribution, the threshold criterion as used for the analysis of the *d*-series (whether it be with $n=2$, like in equation (2.5), or with another value for *n*) no longer suffices. A different criterion is required, which compensates directly for the higher background intensity. This criterion eventually became:

$$I_{Min,e} = 3\sqrt{1020} + \left(1020 - \langle I_{DC} \rangle\right) \tag{2.8}$$

This equation can be seen as an adaptation of the $I_{Min} = n\sigma$ mentioned earlier, however in this case $n = 3$ and an offset of (1020-$<I_{DC}>$) is included. The value of σ is changed from $<I_{DC}>$ to 1020: the value of the average background intensity in the *e*-series diffraction images. Taking a single value for σ, independent of pixel location, ignores the structural increase in the background intensity with increasing distance from the center, but this is justified since this error is only a small effect, reduced even further by the square root operation. The offset in equation (2.8) is required as a consequence of the way in which the background correction is carried out. Before the scanning for peaks is started, the diffraction image is first corrected for the dark current intensities by subtracting from each pixel in the image the average intensity of the corresponding pixel in the dark current images (see for instance line 134 of the AnalyzeLayer_4d code). The correction for the non-electronic background, however, is postponed to a later stage. The main reason for this is that the background intensity of pixels within a diffraction ring, but not belonging to a peak, might be higher than the background intensity of a pixel within a background ring. This implies an extra radial dependence of the background intensity, requiring a more complex correction scheme and prohibiting this from being carried out before peak scanning. This does mean, though, that at the moment of peak scanning all pixel intensities still contain a contribution from the non-electronic background. Since this contribution should be ignored when scanning for peaks, an offset of (1020-$<I_{DC}>$) is included. One could also deal with this problem by subtracting a value of 1020 instead of $<I_{D.C.}>$ from all pixels. In that case, however, element-wise correction would no longer be possible, and the problem of hot pixels as described in subsection 2.2.2 would reappear. Therefore the choice for the solution presented here was made.

The final deviation of equation (2.8) from equation (2.5) encompasses the choice for $n = 3$ instead of $n = 2$. As mentioned, 2 is a common starting value for n, but this value can be refined by a trial-and-error type of process. After initially running the peak search with $n = 2$, the results showed a large amount of pixels that were identified as peaks but, as turned out from visual inspection of the data, should have been discarded. Most notably the list of peaks contained many pixels that were situated near the edges of real spots, and were in fact a part of those spots, but that were mistakenly identified as separate peaks due to small perturbations in the intensities in the vicinity of these pixels. By increasing the value of n to 3, many of these pixels did not reappear in the list of peaks. Note that this increase of n could theoretically also lead to the discarding of small but real peaks. However, this is expected to only be a minor effect for dataset *e* since the average grain size is expected to be relatively large due to the austenitizing treatment of 1 hour prior to measuring.

2.5.4. Mask computation

Equation (2.5) describes the criterion that any pixel in a *d*-series diffraction image must fulfill before it can even be considered as a peak (or, equivalently, equation (2.8) for dataset *e*). This criterion is formulated in the code in line 57. The following step is the computation of the masks of the diffraction rings using the routine CreateMask2DTilt.m. Each ring has its own mask, *Mask_A200* and *Mask_A220* for the austenite {*200*}- and {*220*}-ring respectively. These masks are matrices of the same size as the diffraction images, which contain only 1's and 0's: a 1 for each pixel that lies within the diffraction ring, and a 0 for any pixel outside of this ring. In this way, when the diffraction image is multiplied by a mask in an element-wise manner, all pixels will be set to 0 except those within the diffraction ring of interest. This effectively masks off the part of the detector not of interest to the analysis; hence the terminology.

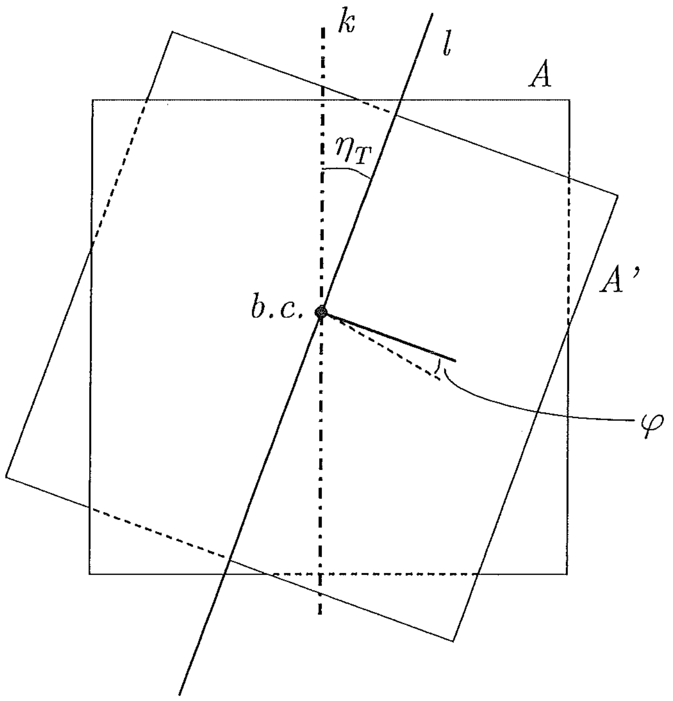


Figure 2.6: Schematic representation of the characterization of the degree of detector misalignment. When the detector screen is not placed perpendicular to the incoming beam (striking the screen at the beam center $b.c.$), the detector plane changes from the ideal plane A to A'. The degree of misalignment is completely characterized by two angles: η_T, the angle between the vertical k and the rotation axis l, and φ, the angle between A and A' (known as the tilt angle).

The routine CreateMask2DTilt.m had already been written prior to the start of this project. It was written by dr.ir. Niels van Dijk of the Reactor Institute Delft; the code of the routine is included in the appendix (page 85). The required input data are the sample-detector distance and pixel size, the characteristics of the detector misalignment, the location of the beam center, and the minimum and maximum scattering angles of the ring of interest. The detector misalignment is entered using the two angles φ and η_T as defined in Figure 2.6. This amount of misalignment is required because this determines the degree to which the diffraction rings have been distorted into ellipses. The routine compensates for this by giving the masks an ellipsoidal shape equal to that of the distorted rings, so no data are lost when the masks are applied. The location of the beam center follows from the computation of the weighted average beam center location around the direct beam mark performed by DetermineBC.m (see subsection 2.2.2).

After the mask creation, some 35 lines follow that determine the outermost pixels of each mask. In this way, the rectangle inscribed by the masks can be reconstructed, so that the actual peak scanning can be restricted to this area and does not include all of the pixels in the outer regions of the detector. This principle is illustrated in Figure 2.7. Note that all of the pixels on the detector area have already been set to zero by the masking operations, except for the pixels that lie between the

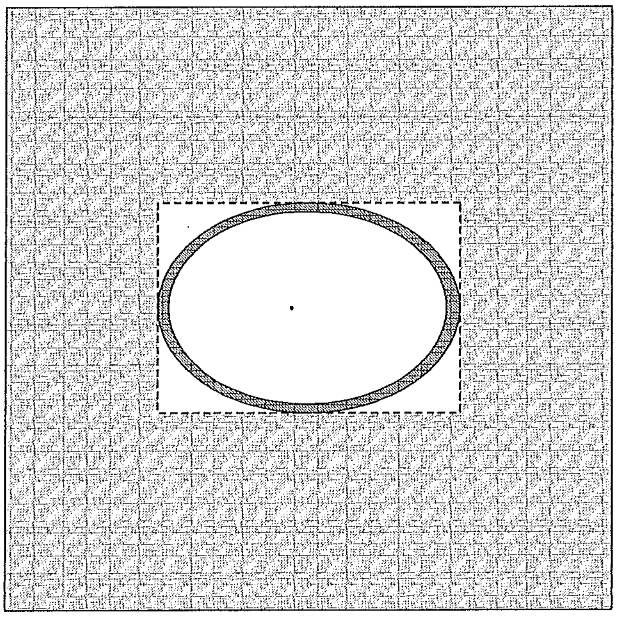


Figure 2.7: Schematic representation of the reduction in peak scanning area after the application of a mask. The masking sets all pixels to zero except those that lie in the ring of interest (dark shades). By subsequently reducing the scanning area to the rectangle enclosing the diffraction ring (dashed line), the majority of the detector area can be ignored (light shades). The remaining pixels (white) have been set to zero by the masking and therefore will not yield a peak.

two ellipses (this area is highlighted by the dark shades). By refining the scanning area to the rectangle enclosing the outer ellipse, all pixels further away from the beam center are ignored (the light shades in Figure 2.7). This greatly reduces the amount of pixels to be scanned, especially since the two rings of interest in this project (γ-200 and γ-220) are located close to the beam center. The scanning area still contains a relatively large amount of pixels outside of the area of interest (these form the white area in the illustration), but since these have been set to zero by the masking none of these pixels will show up in the resultant list of peaks.

2.5.5. File read-in

Now that the starting parameters have been set and the masks have been constructed, the actual image analysis can commence. Remember that this routine analyzes a single layer per call, and that each layer of the sample was scanned over an ω-range of 92° in 92 steps of $\Delta\omega = 1°$ (for more on the experimental approach, see subsection 3.1.2 of the thesis report). Therefore the image analysis begins by looping over these different ω-values. For the analysis of a diffraction pattern, however, the neighboring files in ω-space on either side are also required (as will be explained later on). Since these are unavailable for the files corresponding to the two outermost ω-settings (ω= -30° and ω= +61°), these two are not included in the loop. So, line 112 of the code shows the loop running from -29° to +60°. Given the small rotation of $\Delta\omega = 1°$ applied at each ω-setting, this corresponds to an angular scanning region of 90°.

If ω equals -29°, three different diffraction images are read. First of all, the diffraction image corresponding to ω= -29° is read. Line 118 computes the associated file number, after which line 133 reads the file using readfrelon2k.m, the small routine that was already mentioned earlier. The image is subsequently corrected for the dark current intensities (line 134), and the masks for the two rings of interest are applied, creating two new matrices which will be used later on for the actual peak scanning (lines 135 and 136). After this main diffraction image has been retrieved, the two neighboring files in ω-space are also read and manipulated in the same manner. As mentioned above, these files are needed for the actual peak scanning process. So, although the files corresponding to an ω of -30° are not scanned for peaks themselves, they are used in the scanning process of the neighboring files. The same goes for the files with ω= +61°.

If ω is unequal to -29°, then the process of file read-in can be simplified, because two of the three files used in the peak scanning at the previous ω-setting can be reused for the current operations; they just need to be renamed. This renaming is coded in lines 255 through 258. The only new file that needs to be read from disk is the file corresponding to the subsequent ω-setting; lines 259 through 276.

Lines 178 through 251 contain code that deals with some specific anomalies of the *d* dataset. First of all, during the analysis of the *d* dataset it turned out that one of the files (file number 0038, corresponding to stripe 0, layer 0, $\omega = +8°$) was missing. It appears something went wrong here during the data collection. The implication hereof is not only that this specific file cannot be analyzed, but that the same goes for files 0037 and 0039, the two neighboring files in ω-space. Hence these three files are skipped, using the `continue` statement to move on to the next iteration of the loop. Due to the skipping of these three files, when file 0040 is analyzed all three required images need to be renewed. So, lines 189 through 214 repeat the file read-in that is normally carried out when $\omega = -29°$. Lines 216 through 251, finally, deal with another abnormality in the dataset, namely that file 3141 represents a matrix of dimensions 2047×2048 instead of 2048×2048. For some reason the recording of the data was not performed correctly, and one row of values was lost. Inspection of the image suggested that this was probably the bottom row of the detector. The routine readfrelon2k.m works by first going to the end of the file, then counting backwards 2×2048^2 bytes (each pixel is represented by two bytes), and then reconstructing the diffraction image from that point on. Therefore, the top row as it is constructed during file read-in is discarded (this row contains corrupted data in the form of numerical representations of ASCII characters from the file's header), and an extra row of pixels with intensity 1000 (the approximate dark current intensity) is added at the bottom. From here on, the analysis can continue as usual.

2.5.6. Peak detection criteria

Now that the required files have been read and masked, a new loop is started, this time over the two rings of interest (line 280). The current ring is identified by the counting variable *ring*, which is then used in a `switch`/`case`-construction to pick out the correct masked images and scan area borders (lines 282 till 301). The correct masked image is now available together with its neighbors in ω-space, and the actual peak searching can commence. This process starts from line 305 downwards. A double loop is started over the row and column indices of all pixels in the area of interest. The intensity of each pixel is first checked against the threshold intensity as given by equation (2.5). The majority of the pixels will not pass this criterion, either because their intensity has been set to zero by the masking procedure or simply because they do not belong to a spot.

Those pixels that do have a high enough intensity are then subjected to another criterion: their intensity is required to be higher than (or equal to) the intensities of all its neighbors. This criterion is formulated in lines 308-312. Figure 2.8 presents a schematic representation of how this validation works. The pixel's intensity is first checked against that of its 8 neighbors within the same image. In this way, when a pixel belongs to a spot but does not constitute that spot's center, it will generally not be regarded as a peak on its own because its neighbor in the direction of the spot center will usually have a higher intensity. If the pixel does pass this test, it is subsequently compared to its 9 neighbors in both the preceding and succeeding file in ω-space. This ensures that the pixel is not part of the tale of another spot at a neighboring ω-setting. This is relatively unlikely, though, since lattice strains should be small due to the austenitizing treatment and since the sample wasn't deformed prior to the measurements. Therefore, the spread in lattice orientation within a single grain is expected to be small and hence the chance of a single grain producing intensity at multiple values for ω is also minimal.

When a pixel has an intensity higher than the threshold, and its intensity is also the highest one within the 3×3×3 box in (x,y,ω)-space, then that pixel is identified as a peak. Several characteristics of the peak are now calculated. These include its distance from the beam center R, its exact diffraction angle 2θ, and its azimuthal angle η. Furthermore, a first indication of the size of the peak is calculated

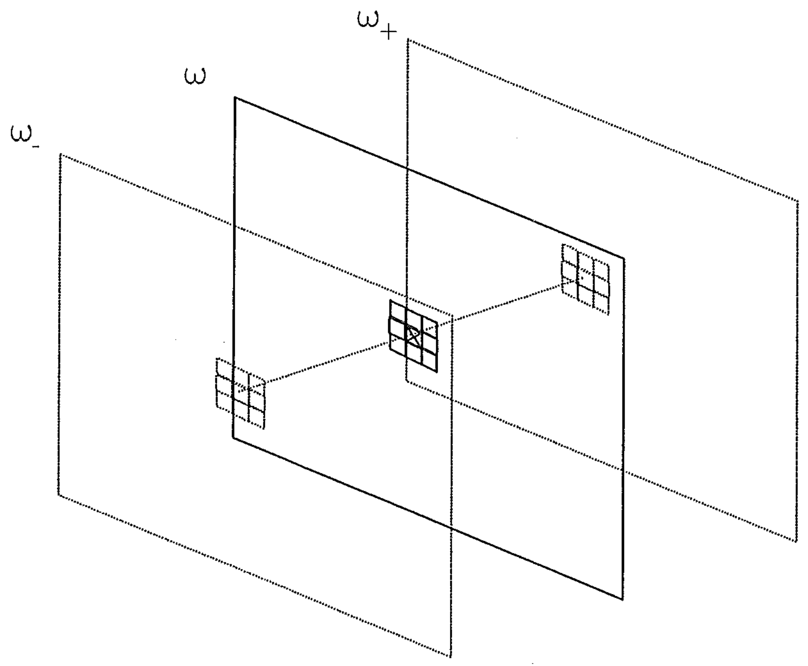


Figure 2.8: Schematic representation of the second part of the peak scanning process. The pixel indicated with a cross has an intensity higher than the threshold intensity. It is then checked against the intensities of its 26 neighbors in three-dimensional (x,y,ω)-space. When the intensity of the pixel in question is larger than or equal to the intensities of all of its neighbors, it is identified as a peak.

by determining the half-width half-maximum (HWHM) locations of the peak. For each of the six directions (x, -x, y, -y, ω, -ω) the distance to the pixel at which the intensity has dropped to half the intensity of the peak's maximum (or less) is determined. For each of the three dimensions, the average of the two corresponding values (rounded up) is then taken as the HWHM value, providing a first estimate of the peak size. Note that the HWHM value in the ω direction can never be more than 2 pixels. As mentioned earlier, a HWHM value of 2 is already quite unlikely; the possibility of a higher value is neglected. This also prevents the additional problem of having to read in extra files in order to check the pixel's intensity beyond its nearest neighbors in ω-space.

All of the peaks' characteristics are written into column matrices. As soon as the loop over all pixels in the area of interest is finished, the length k of these column matrices represents the number of peaks found in the diffraction ring under consideration. These matrices are then concatenated, and subsequently appended to the matrix u, which contains the results of the previous scans. The matrices are cleared again, and the process is repeated for the second ring. After this task has been performed, the next iteration of the loop over ω is started. In this way, at the end of all the loops (line 422), matrix u contains the characteristics of all the peaks found for the slit settings defined by the input parameters *stripe* and *layer*. This matrix is returned to the calling program as the final output of the routine. In the case of the current project the calling program was FindPeaks_4d.m, which appended the output u to the larger matrix *Peaks*, which eventually became a list of all the peaks found in the *4d* diffraction images. This list was carried forward to the next part of Analyze_4d.m: the characterization of the corresponding spots.

2.6. Spot characterization – ShellCheck_4d.m

FindPeaks_4d.m has created a list of all the pixels in the *4d* diffraction images that have been identified as a peak. This implies the intensities of these pixels are significantly higher than the background, and they are higher than (or equal to) the intensities of their 26 neighbors in (x,y,ω)-space. The list contains various attributes of these pixels, such as their location on the detector, their diffraction angle, their intensity etc. Furthermore, for each pixel the list contains an estimate of the size of the corresponding spot in the form of three half-width half-maximum (HWHM) values, one for each dimension.

Clearly, though, the spot size as follows from the HWHM values will not always be acceptable for use in further calculations. Especially for larger spots the amount of intensity in the tails of the peaks can be considerable, and a large amount of this intensity would be lost if one were to use the HWHM values for the final spot dimensions. Therefore, a refinement of the spot size is required; the HWHM values can then serve as a first spot size approximation to the routine. ShellCheck_4d.m was written with this purpose.

2.6.1. Global outline

This subsection presents the global structure of ShellCheck_4d.m. After this subsection, the individual processes carried out in this routine will be treated in more detail.

Figure 2.9 shows a flow chart of the entire routine, which analyzes a single peak per call. When the routine has been called, first some parameter settings are declared. When this has finished, a loop is started to determine the exact size of the diffraction spot in question. This is done on the basis of three objects: a peak box (representing the current assumed spot size) and two surrounding shells – for a detailed explanation of this procedure, see subsection 2.6.2. First of all, the peak box Box0 and the surrounding shells Shell1 and Shell2 are created. This is done using the HWHM values associated with the peak under consideration, which were determined

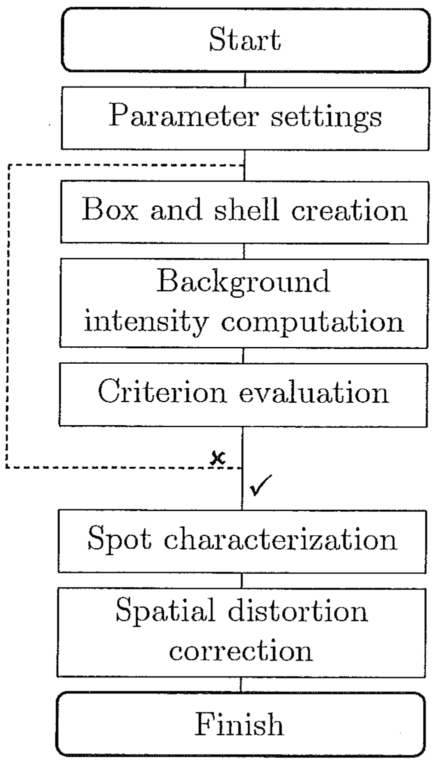


Figure 2.9: Flow chart of the spot characterization process. The routine determines the dimensions of diffraction spots represented by previously detected peaks, and characterizes them in terms of total corrected intensity and center of gravity detector coordinates. These coordinates are subsequently corrected for the spatial distortion of the image introduced by the setup's optics.

during the peak search performed by FindPeaks_4d.m. For each of the two shells, the average background intensity per pixel is determined as a function of the pixel's distance to the beam center. The outcomes are subsequently compared on the basis of a criterion determining whether or not the two are approximately equal. If the criterion is satisfied, the spot dimensions are accepted and carried on to the next part of the routine. If the criterion is violated, the peak box and shells are redefined and the analysis is repeated; this is done until the spot dimensions are adequate.

When the routine has arrived at the correct spot dimensions, the analysis continues by characterizing the entire spot in terms of its total corrected intensity (corrected, in this case, meaning that both the electronic and non-electronic background contributions have been filtered out) and of its center of gravity detector coordinates. Finally, these coordinates are corrected for the spatial distortion that was introduced by the optics system during data collection (see subsection 2.2.1). This yields the corrected center of gravity coordinates and total intensity of the spot, which are then presented as output to the calling program.

The exact MATLAB code of the routine is given in the appendix and starts on page 87. As mentioned above, the routine analyzes a single peak per call. The required input are the peak's characteristics as provided by the peak list created at an earlier stage by FindPeaks_4d.m. This includes the file in which the peak was found (in terms of *stripe*, *layer* and ω), the location of the peak within the diffraction image (*ringnumber*, *row*, *column*) and the HWHM estimates of its dimensions (*deltarow*, *deltacolumn*, *deltaomega*). Furthermore, the dark current intensities (*DarkCurrent*) and the matrix containing the spatial distortion corrections (*SDMatrix*) are carried over from the calling program. This prevents having to recompute these matrices each time ShellCheck_4d is called.

2.6.2. Non-electronic background correction

The essence of deciding if the spot dimensions have been set correctly, is determining whether or not the peak box contains all pixels that show an increase in intensity as a result of the diffraction spot. Hence, some criterion is required to determine whether or not the pixels bordering the peak box have an intensity that is higher than expected for pixels that do not belong to a spot, in other words an intensity higher than can reasonably be attributed to the background. The average background intensity will generally show a dependence on the distance from the beam center R, as visible for dataset *e* in Table 2.1. As explained in subsection 2.5.3, this is mainly caused by the increased thermal scattering at higher values for R.

So, ShellCheck_4d.m needs some way of determining whether the intensity of pixels bordering the peak box could reasonably be attributed to the background or not; however, the average background intensity is not something that can be determined simply by averaging a few pixel readings at some distance from the beam center at which no peaks are expected. Determination of the average background intensity within a diffraction ring is a complicated procedure, as computing this would require knowledge of which pixels within such a ring are part of a spot and which are not, since pixels belonging to a spot should not be included in this averaging procedure. This, in case, would require prior knowledge of the threshold intensity, which depends on the average background intensity; and this is exactly what we are trying to determine. The only way to directly determine the average background intensity would therefore be some kind of iterative procedure where one would start with a certain value for the average background intensity (for instance the average of all pixels in the diffraction ring, so including those belonging to a peak), determine which pixels belong to peaks and which don't, and use this knowledge to recompute the average background intensity. This new value can then be used for the subsequent iteration. This could be repeated until the computed intensity no longer shows any significant changes. The resulting routine, however, would be complicated and time-consuming.

Therefore, a different approach to this problem was designed. This was mostly done by dr. Enrique Jimenez-Melero of the Reactor Institute Delft. The approach is based on the definition of two shells: Shell1, which strictly envelops the peak box (designated as Box0) in all three dimensions, and Shell2, which strictly envelops Shell1. The situation is visualized in Figure 2.10. This figure depicts a pixel which has been identified as a peak, at a distance R_p from the beam center. This pixel also has HWHM values associated with it. These values are used to construct a box around the peak which serves as a first approximation to the actual spot dimensions. Around this box (Box0) the two shells are defined. Note that, although the figure only displays a single diffraction pattern and hence only visualizes the procedure in the two-dimensional (x,y)-space, the peak box and surrounding shells also extend into ω-space.

As mentioned above, the main difficulty surrounding the determination of the final peak dimensions is the dependence of the non-electronic background intensity on the distance from the beam center R. Since the pixels within Box0 – the pixels that together form the entire spot – as well as the pixels in Shell1 do not have a constant distance to the beam center, the non-electronic background correction they require is not constant either. For each distance to the beam center, a separate average non-electronic background intensity value should be computed, so that each of the peak's pixels can be corrected with the appropriate value.

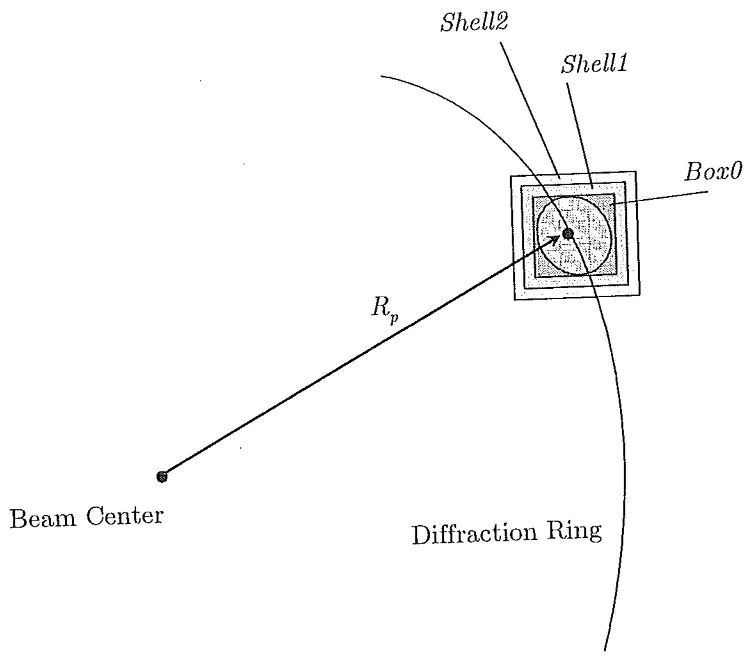


Figure 2.10: Two-dimensional visualization of the definitions of *Shell1* and *Shell2* used in the determination of the correct size of the peak box *Box0* around a peak at a distance R_p from the beam center. *Shell1* is a shell of a single pixel in width around the peak box *Box0*, whereas *Shell2*, also a single pixel wide, envelops *Shell1*. Note that, although the figure only displays the detector's x and y directions, the peak box and the two enveloping shells also extend into ω-space.

This is done in the following manner. For all of the pixels in Box0, Shell1 and Shell2, the distance to the beam center (in units of pixels) is computed. These distances are rounded to integer numbers so that a discrete distribution of distances around R_p is formed. Depending on the dimensions of the peak box (varying from 2 to 10 pixels HWHM in either direction), Shell2 for instance will typically contain pixels with radial distances varying between (R_p-10) and (R_p+10). Subsequently, for both shells, the intensities of pixels that have the same distance to the beam center are averaged. In this manner, a list is created for each shell which contains the distances of the pixels within that shell to the beam center and their corresponding average intensity. The following reasoning is now applied. If Shell1 does not contain any intensity from the spot in Box0, and Shell2 does not contain any intensity from any neighboring spot, then both shells only contain background intensity and therefore the average background intensity of pixels in Shell1 at a certain distance from the beam center should be approximately equal to that of pixels in Shell2 at the same distance. So, the average intensity of pixels within Shell1 as a function of distance to beam center, $<I_{Sh1}(R)>$, is compared to the average intensity of pixels within Shell2 at the same distance, $<I_{Sh2}(R)>$. If these are approximately equal for all values of R that fall within Shell1, then the peak box dimensions are accepted as the final dimensions. Analysis can continue, using the background intensities from Shell1 for the non-electronic background correction of the pixels in Box0. If Shell1 shows significantly higher intensities than Shell2, then Shell1 still contains intensity from the spot in Box0. The peak box dimensions are subsequently enlarged by a single pixel in each direction, and the procedure is executed again. If Shell2 contains higher intensities than Shell1, then it appears Shell2 contains intensity from a neighboring spot. The peak box dimensions are lowered by 1 pixel in all directions to try and exclude this influence, and the procedure is repeated. In this way, the peak box size is refined until the correct dimensions have been obtained.

The criterion used for determining whether or not $<I_{Sh1}(R)>$ and $<I_{Sh2}(R)>$ are 'approximately equal' is based on the normal approximation to the Poisson distribution, and follows a reasoning similar to that of the intensity threshold criterion outlined in subsection 2.5.3. It is assumed – as is common practice – that the background intensity follows a Poisson distribution [13]. In that case, using the reasoning of subsection 2.5.3, the background intensity of a pixel at a certain distance R from the beam center can be approximated by a normal distribution with mean and variance equal to the intensity's average. Remember, however, that the intensities of Shell1 and Shell2 are actually averages of multiple pixels that all lie at distance R. Averaging N observations from the same distribution has the effect of reducing the variance of the average by a factor of N as compared to the variance of a single observation. So, if Shell1 contains N_1 pixels at a distance R from the beam center, then the variance $(\sigma_{Sh1})^2$ of the average background intensity in Shell1 at R can be approximated by:

$$(\sigma_{Sh1})^2 \simeq \frac{\langle I_{Sh1}(R)\rangle}{N_1} \tag{2.9}$$

A similar expression can be written for the variance in the background intensity in Shell2 at a distance R from the beam center. By comparing the actual difference between the average intensities in the two shells, $<I_{Sh1}(R)>-<I_{Sh2}(R)>$, to the sum of their standard deviations, one can now determine whether or not the two differ significantly. The criterion for determining whether the peak box dimensions have been set correctly then becomes of the form:

$$\left|\langle I_{Sh1}(R)\rangle - \langle I_{Sh2}(R)\rangle\right| < \sqrt{\frac{\langle I_{Sh1}(R)\rangle}{N_1}} + \sqrt{\frac{\langle I_{Sh2}(R)\rangle}{N_2}} \tag{2.10}$$

in which the left-hand side of the criterion represents the absolute difference between the two experimentally observed average intensities, and the right-hand side represents the sum of the standard deviations of the distributions of the average intensities in the two shells at distance R. Shell1 and Shell2 contain N_1 and N_2 pixels with this specific distance to the beam center, respectively. If the criterion is not met, and $<I_{Sh1}(R)>$ is larger than $<I_{Sh2}(R)>$, then Shell1 contains intensity from the spot in Box0. If the criterion is not satisfied but $<I_{Sh2}(R)>$ is larger than $<I_{Sh1}(R)>$, then Shell2 contains intensity from a neighboring spot. In case the criterion is fulfilled, then the peak box dimensions are adequate. The peak box is taken as the final spot dimensions, and computation of features like the center of gravity of the spot can commence.

2.6.3. Initial settings

As with most of the routines, ShellCheck_4d starts by declaring some parameters that are required later on in the analysis (lines 18 through 22 of the routine's code). Among these is the beam center location on the detector in terms of its row and column index. At the end of the routine, the spot's detector coordinates are corrected for their spatial distortion. Clearly, this procedure should also be performed on the beam center coordinates, if one wishes to compute a reliable value for for instance R, the spot's distance to the beam center. This correction is carried out in lines 34-43. First of all, the borders of the rectangular area covered by the spatial distortion matrix are determined (see Figure 2.7). This is done by determining the coordinates of the first and of the last pixel covered by *SDMatrix*. These pixels form the upper left and lower right corner of the rectangular area and thus define the entire area. Knowing these values, the spatial distortion entries corresponding to the beam center can be retrieved by moving to the correct matrix entry. Since *SDMatrix* is nothing more than a list of all the pixels with their corresponding distortions and their corrected locations, and knowing that the pixels in *SDMatrix* are listed from left to right, top to bottom, the index of the entry corresponding to the beam center can be seen to equal

$$\begin{aligned}\text{index} = (RowBC - m\_min) * (n\_max - n\_min + 1) + \\ (ColumnBC - n\_min + 1)\end{aligned} \tag{2.11}$$

in which $(RowBC, ColumnBC)$ represents the uncorrected beam center location as declared in line 18, (m_min, n_min) represents the upper left corner of the area covered by *SDMatrix*, and (m_max, n_max) represents the lower right corner. Clearly, a similar reasoning applies to any other pixel within the area covered by the spatial distortion correction matrix.

2.6.4. Box and shell construction; background intensity computation

When the various parameters have been declared the routine continues with the refinement of the peak box dimensions. The idea behind this refinement was introduced in subsection 2.6.2. This subsection treats the manner in which this procedure has been translated into MATLAB code.

As mentioned, the spot refinement is performed by means of a loop that is exited when the correct spot dimensions have been achieved. To control the iterations of this loop, three variables are introduced that indicate the current state of the procedure. These so-called flags are introduced in lines 51-53: *box_too_large*, *box_too_small*, and *error_indicator*. The first two are set to 1, while the latter is set to 0. Furthermore, the fact that file number 0038 is missing (see subsection 2.5.5) also requires a flag to deal with potential problems introduced by this anomaly. This flag is called *file38_flag* and is set to 0 on line 55. Finally, the output *Total_Intensity* is already declared in line 57 (for now, just an empty matrix).

Line 59 shows the initiation of the `while`-loop. The loop can be exited only when both *box_too_large* and *box_too_small* both equal 0, or when the error flag is raised. Given the initial declarations of the flags, the loop will always run at least once. Within the loop, first of all, the lists of pixels in the peak box and in the surrounding shells, as well as their derivates *Box0def*, *Shell1def* and *Shell2def*, are set to empty matrices. This is necessary because generally these will still contain values from the previous iteration of the `while`-loop. After this, lines 75 through 93 determine the borders of the peak box and shells. This is done on the basis of the location of the peak in three dimensions as following from the peak list, (*row*,*column*,ω), together with the HWHM values. For subsequent iterations of the loop, these latter values will generally have evolved from the HWHM values to more accurate dimensions.

The ω-values of the images under consideration all lie between -30° and +61°. When a spot is located near these borders of the ω-region, the outermost shell Shell2 could extend beyond these borders. In this case, no accurate comparison can be made between the background intensities in the two shells, and therefore the estimated spot dimensions cannot be validated. Lines 95 through 104 identify the occurrence of this problem. When this problem arises, the error flag is raised and the loop is terminated. Since the criteria of line 59 are no longer all fulfilled, no new iteration of the loop is initiated.

Lines 106 through 125 correct for the afore-mentioned absence of file 0038. Since this file is located away from the border of the ω-region, many files are possibly affected. The loop is entered in case the peak box and/or one or both of the two shells contain pixels located in this image. If this is the case, then first of all an attempt is made to reduce *deltaomega*, to reduce the area in ω-space over which the routine runs. If this can be done the loop is reentered with the new *deltaomega* value. Furthermore the flag *file38_flag* is raised so that in future iterations the width in the ω direction will not be increased again. If *deltaomega* cannot be decreased because it already has its minimum value of 1, then the peak cannot be analyzed and the loop is exited with the error flag raised. Note that *deltaomega* indicates the location of Shell1, and not of the outermost image still belonging to the peak box. So, when *deltaomega* equals 1, this indicates that Shell1 is located only a single image to the 'left' and 'right' of the original image in ω-space; in other words, the neighboring images are part of Shell1. Similar reasonings go for *deltarow* and *deltacolumn*. From this, it also follows that the number of pixels belonging to the peak box, N_0, equals

$$N_0 = (2 * deltarow - 1)(2 * deltacolumn - 1)(2 * deltaomega - 1) \qquad (2.12)$$

Lines 131-213 construct the peak box and shells. A loop is made over the diffraction images from the start of Shell2 in ω-space to the end of Shell2. Each image is subsequently corrected for the dark current background. The loop over the peak box in (x,y)-space together with the loop over the images ensures that all pixels of interest are taken into account. For each of these pixels, it is determined whether the pixel is located in Box0, Shell1 or Shell2. This is expressed in the various `if-`, `elseif-` and `else`-statements within the loop. Each box/shell has a list associated with it that goes by the same name. If a pixel is found to belong to for instance the first shell, Shell1, then a new entry is created in the associated pixel list *Shell1*. This entry consists of the pixel's distance to the beam center as computed in lines 165 and 166, the pixel's intensity corrected for the dark current, and the pixel's uncorrected intensity. This last entry is needed for the evaluation of the criterion as described by equation (2.10). Furthermore, entries in the peak box list *Box0* also contain the detector coordinates and the ω value for each pixel. These are used later to compute the center of gravity coordinates of the spot.

When all pixels have been assigned to either Box0, Shell1 or Shell2, each of the corresponding lists will contain many entries which have the same rounded distance to beam center (termed *PixelPos* in the routine). Lines 218-288 take these lists and construct derivates of them (*Box0def*, *Shell1def*, *Shell2def*) in which the intensities of entries with equal distances to beam center have been summed and averaged. In these new lists, each entry consists of five values: the rounded pixel position, the total summed intensity of pixels with that radius, the number of pixels, the average corrected intensity, and the average uncorrected intensity (so before dark current subtraction). Regarding *Shell1def* and *Shell2def*, this latter entry is exactly the value required for evaluation of the right-hand side of the criterion of equation (2.10).

2.6.5. Criterion evaluation

Now that the average intensities per pixel in Shell1 and Shell2 are known as a function of the distance to the beam center, validation of the current spot dimensions can commence. If the peak box is set to the correct size, then for any distance to beam center the average intensity per pixel in Shell1 will not differ significantly from that of a pixel at the same distance but lying in Shell2. This is evaluated in lines 298 through 374. A loop is started over the length of *Shell1def*, so that each iteration corresponds to a specific entry in this list and therefore to a specific rounded distance to beam center (termed *PixelPos1*). First of all, the entry in *Shell2def* with the same distance to beam center is retrieved (301-305). By retrieving the values of the average uncorrected pixel intensity and the number of pixels for both of these entries, the right-hand side of equation (2.10) can be evaluated. This is done on line 313 of the routine's code, and the outcome is termed *criterion*. The value of *criterion* is then compared to the actual difference between the average corrected intensities per pixel in Shell1 and Shell2 at the distance to beam center under consideration.

In case the difference between the average intensities is larger than *criterion*, and the intensity in Shell1 is the larger of the two (line 318), then this indicates that the peak box dimensions are too small. Therefore, *deltarow* and *deltacolumn* are increased by a single pixel. The value of *deltaomega* is also increased by 1 image, but only in case the flag *file38_flag* hasn't been raised (see the corresponding paragraph on page 34). The state variable *box_too_small* is also increased by 1, to indicate that the peak box dimensions have been increased. While analyzing the datasets, however, it turned out that for some peaks the routine continued to increase the peak box size indefinitely. The routine contains a safety for this occurrence in the form of a maximum value for the flag *box_too_small*. If this flag exceeds a value of 20, the peak is discarded by setting the error indicator to 1 and exiting the loop.

If the difference between the two average intensities is larger than *criterion*, but the intensity in Shell2 is larger than that in Shell1 (line 340), then it appears that the outer shell contains some intensity from a neighboring spot. Therefore, to ensure that the total integrated intensity of the peak under consideration does not become contaminated with any intensity from the tail of this neighboring spot, the peak box dimensions are lowered by a single point. If this reduces any of the dimensions to zero (this is especially likely for *deltaomega*, since this has an initial value of 1 for almost all peaks), this dimension is reset to 1. The indicator *box_too_large* is also increased by a single point. If all of the dimensions become zero (line 345), then the peak box cannot shrink any further and hence the peak cannot be analyzed. The error indicator is adjusted and the loop is exited.

In case the difference between the average pixel intensities in Shell1 and Shell2 for a certain value of *PixelPos1* is small enough to fulfill the criterion (line 367), no action is required. The next iteration of the loop, and hence the next value of *PixelPos1*, is entered. If the routine has already arrived at the last iteration, the two state variables *box_too_large* and box_*too_small* are both set to zero, so that the condition for reentering the `while`-loop (line 59) is rejected and the routine knows that the current peak box settings are adequate. The routine can continue with the next part of the analysis.

The peak box refinement procedure contains one more test (376-385). In certain cases, it is conceivable that the routine ends up oscillating between increasing and decreasing the peak box size. For instance, if during the first iteration of the `while`-loop the box' dimensions are increased because there was still intensity in Shell1, but in the next iteration the dimensions are reduced again because of the threat of intensity contamination from a neighboring spot, then the routine would enter an endless process of increasing and reducing the box size. To prevent this from happening, a last check is performed on the values of *box_too_large* and *box_too_small*. In case they are both larger than 1, then this indicates that both increasing and decreasing of the peak dimensions has occurred. If this is the case, no correct peak box size can be determined, and the loop is exited.

2.6.6. Spot characterization

If the peak refinement procedure has been unable to produce acceptable results, then this is reflected in the value of *error_indicator*: it will have changed from 0 to 1. In this case, lines 389-392 prematurely end the entire ShellCheck_4d routine and return an empty matrix *Total_Intensity* to the calling program.

However, in case the routine did succeed in refining the dimensions to the correct values, the analysis continues by characterizing the spot in question. First of all, the total integrated corrected spot intensity is computed; 'corrected' in this case meaning that both the electronic and non-electronic background contributions have been excluded. As mentioned in subsection 2.6.2, the correction for the non-electronic background can be carried out by subtracting from each pixel within Box0 the average intensity of the pixels in Shell1 that have the same distance to beam center *PixelPos*. Lines 405-415 contain the code accounting for this. The variable *Total_Peak_Intensity*, introduced in line 402, will eventually contain the entire spot's corrected intensity. For each entry in *Box0def*, the entry in *Shell1def* with the same distance to the beam center is retrieved. Say that there are $N_{P.P.}$ pixels within Box0 that are located at a specific rounded distance to the beam center (*PixelPos*). These pixels add up to a total intensity of $I_{P.P.}$. This intensity has already been corrected for the electronic background contribution (the dark current intensities have been subtracted), but still contains some amount of non-electronic background intensity. For this specific distance to beam center, *Shell1def* lists an average intensity per pixel of $I_{Sh1,P.P.}$. In that case, the intensity of pixels within Box0 at that specific distance to beam center can be corrected for the non-electronic background by subtracting the average non-electronic background intensity per pixel times the number of pixels at that specific distance to beam center. Through this procedure, the total peak intensity corrected for both types of background, *Total_Peak_Intensiy*, can be written as

$$Total\_Peak\_Intensity = \sum_{P.P.} \left(I_{P.P.} - N_{P.P.} I_{Sh1,P.P.} \right) \tag{2.13}$$

in which the summation runs over all values of *PixelPos* for which there are pixels present in Box0. The loop contained by lines 405-414 does exactly this. The final value of *Total_Peak_Intensities* represents the total integrated spot intensity.

The next step in the characterization of the diffraction spot is the computation of the location of the center of gravity. This computation is based on a weighted average of the coordinates of all the pixels belonging to the spot, using the corresponding corrected intensities as weights. As a first step, the original list of pixels belonging to the spot, *Box0* (so before averaging pixels with the same distance to beam center), is sorted in ascending order on the basis of the distance to the beam center; the result is called *Box0_Sorted*. Then, the intensity of each entry in *Box0_Sorted* is corrected for the non-electronic background by subtracting the appropriate average background intensity from *Shell1def* (lines 428-436). So, *Box0_Sorted* now contains all pixels in the box, together with their corrected intensities and their locations in three-dimensional (*row*,*column*,ω)-space. Lines 442 up till 454 now compute the center of gravity coordinates of the diffraction spot in three-dimensional space.

2.6.7. Spatial distortion correction

The final part of ShellCheck_4d.m deals with the correction for the spatial distortion that all pixels have undergone during data collection. This correction is required since for reconstruction of the scattering vectors of all diffraction spots, the exact location of the spot on the detector is needed. First of all, however, a word on the timing of this spatial distortion correction is required.

As mentioned in subsection 2.2.1, the spatial distortion introduced by the system's optics is modeled using two bivariate splines of the third degree in both dimensions. This means that the function describing the spatial distortion as a function of a pixel's location on the detector is not linear, and the corrected average location of a specific spot will not be the same as the average corrected location of that spot:

$$\begin{aligned} f_m(\langle m\rangle,\langle n\rangle) &\neq \langle f_m(m,n)\rangle \\ f_n(\langle m\rangle,\langle n\rangle) &\neq \langle f_n(m,n)\rangle \end{aligned} \tag{2.14}$$

Here, $f_m(m,n)$ and $f_n(m,n)$ represent the spline function for the distortion of a pixel (m,n) in FIT2D's y and x direction, respectively. The angle brackets represent weighted averages over all pixels within a specific diffraction spot.

Because of this non-linearity of the spatial distortion spline functions, strictly speaking the spatial distortion correction should be applied to all pixels before the start of the analysis. In this way, the peak search commences on the corrected images, and the distortion effects have been cancelled before they can even play a role in the process. However, this procedure increases the computational load. Furthermore, it complicates the subsequent analysis because the grid points are no longer evenly spaced in the (x,y)-plane of the detector. Therefore, it would be beneficial if it were acceptable to apply the spatial distortion correction at a later stage of the analysis. To analyze the error introduced by such a delayed application of the spatial distortion, a single spot was chosen and analyzed using three separate scenarios:

A| First correct all pixels for their spatial distortion. Then start the analysis: search for peaks, determine the correct peak box size, and compute the location of the spot's center of gravity.

B| Perform the peak search, and determine the correct peak box dimensions. Correct all pixels within the box for their spatial distortion, and then compute the spot's center of gravity.

C| Perform the peak search, determine the peak box dimensions, and compute the center of gravity. Correct this center of gravity location using the spatial distortion spline.

Strictly speaking, scenario A is the correct way of applying the spatial distortion. Scenario B locates the peaks using the uncorrected pixel locations, and refines the peak box sizes using distorted HWHM values. This could influence the final peak box dimensions determined by the routine. Scenario C reduces the spot to a single set of center of gravity-coordinates $(row_{CoG}, column_{CoG})$, and applies the correction only to these two coordinates. Clearly this reduces computational load, but the accuracy of the resulting center of gravity coordinates of the peak might suffer considerably.

The spot chosen to be analyzed using these three separate scenarios was required to have two important characteristics. First of all, it was required to be a large spot. This will generally enlarge the differences between the outcomes of the three scenarios. Secondly, the spot was required to be located in the outer ring under consideration in this project, the *{220}*-ring. The general trend of the spatial distortion is to increase with increasing distance from the detector's center. Therefore, by picking a peak in the outer ring the influence of the distortion is likely to be largest, highlighting the differences between the scenarios even more.

The specific diffraction image in which the pixel was to be found was chosen at random. However, the choice was made *a priori* for a peak in the *e*-series, since the illuminated volume during these measurements was three times as high as during the *d*-series, and therefore broader peaks are expected. In this case, the picture chosen was file number 4e1711, corresponding to the following settings: *stripe* = 0, *layer* = 18, ω = -27°. Within this image, a visual search for a suitable spot was conducted. Eventually, a spot around pixel (868,383) was chosen. This spot was located in the outer ring, and was one of the larger spots with estimated HWHM values of 10 pixels in both detector dimensions. The spot was subsequently analyzed using the three scenarios as listed above. The results are listed in Table 2.2.

Table 2.2: Results of the analysis of the spot around pixel (868,383) of file 4e1711 using the three different scenarios as listed on page 39. It can be seen that the influence of postponing the spatial distortion correction only has a minor effect on the spot's computed center of gravity coordinates on the detector, (m_{CoG}, n_{CoG}), and in ω-space (ω_{CoG}), as well as on the spot's total integrated intensity.

	ω_{CoG} (°)	m_{CoG}	n_{CoG}	Total intensity (# counts)
Scenario A	-27.01	862.91	369.61	141×10^3
Scenario B	-27.03	863.00	369.61	146×10^3
Scenario C	-27.03	863.00	369.61	146×10^3

The table shows that the differences between the results of the peak analysis following the three scenarios are only minor. A minute deviation in the horizontal location of the spot's center of gravity is recorded, as well as a small increase in total integrated spot intensity (3.5%). Since these results have been obtained for a large spot in the outermost diffraction ring that is of interest during this project, the difference between the results of scenarios A and C can be regarded as a type of upper limit. It is not likely that the effect of postponing the spatial distortion correction will be much larger for any of the other spots. So, it appears acceptable to postpone the spatial distortion correction till after computation of the center of gravity coordinates of the spot, so that it needs to be applied only once per spot.

The results in Table 2.2 explain why the spatial distortion correction is only applied now, at the end of the peak characterization routine. Applying the distortion correction to the averaged peak coordinates does, however, introduce another difficulty. As outlined in subsection 2.2.1, the routine SDCorrection.m creates a look-up table of all the pixels in the detector area of interest, together with their distortions and their corrected intensities. The appropriate correction can then be carried out by looking up the required shift in this table. Clearly, this table only contains corrections for integer values of the pixel coordinates. However, the center of gravity coordinates (m_{CoG}, n_{CoG}) are a result of a weighting procedure, and will therefore in general not consist of integers. To account for this mismatch, an interpolation process is used. Figure 2.11 shows a schematic representation of this process. The figure displays the case for the shift in row coordinate of the pixel (or y coordinate in the FIT2D coordinate system), but obviously the same reasoning can be applied to the horizontal distortion.

The center of gravity of a random spot is located at (m_{CoG}, n_{CoG}), with $m^* < m_{CoG} < (m^*+1)$ and $n^* < n_{CoG} < (n^*+1)$. The corrected locations of the four nearest pixels are known and can be found in the look-up table created by SDCorrection.m. The interpolation routine now constructs a curved surface through these four pixels according to the following relation:

$$m_{corr}(m_{CoG}, n_{CoG}) = M_{11} + M_{21}(m_{CoG} - m^*) + M_{12}(n_{CoG} - n^*) + M_{22}(m_{CoG} - m^*)(n_{CoG} - n^*) \tag{2.15}$$

Here, m_{corr} represents the corrected value of the first index as a function of both of the spot's center of gravity coordinates. The coefficients M_{ij} are related to the corrected values for the nearest pixels in the following manner:

$$M_{11} = m_{corr}(m^*, n^*) \tag{2.16}$$

$$M_{21} = m_{corr}(m^* + 1, n^*) - m_{corr}(m^*, n^*) \tag{2.17}$$

$$M_{12} = m_{corr}(m^*, n^* + 1) - m_{corr}(m^*, n^*) \tag{2.18}$$

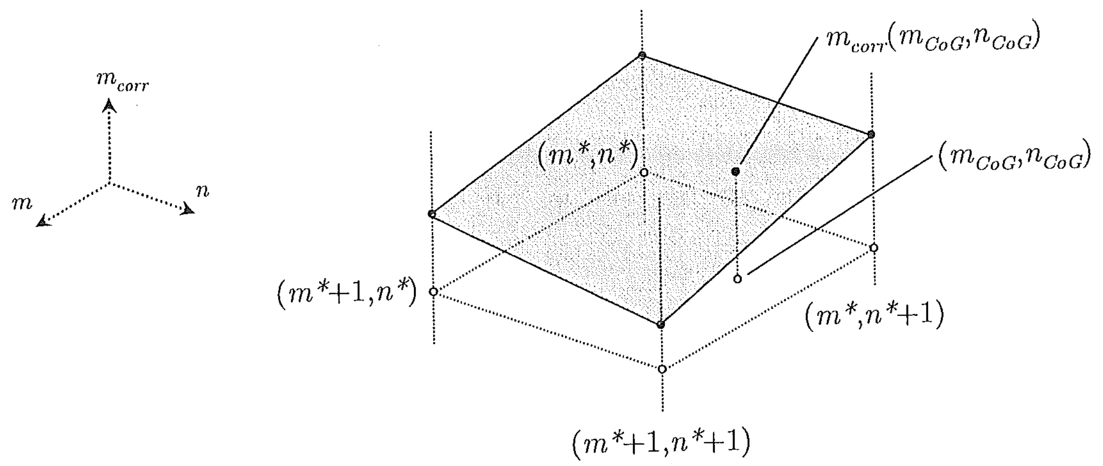


Figure 2.11: Schematic representation of the interpolation procedure used for computing the distortion correction in FIT2D's y direction for non-integer center of gravity coordinates. The correction for a spot located at (m_{CoG},n_{CoG}), with $m^*<m_{CoG}<(m^*+1)$ and $n^*<n_{CoG}<(n^*+1)$, is computed by constructing a (curved) surface through the four nearest pixels. The distortion correction is given by the surface value at the center of gravity coordinates. A similar procedure is used for the x direction.

$$M_{22} = m_{corr}(m^*,n^*) + m_{corr}(m*+1,n^*+1) - m_{corr}(m*+1,n^*) - m_{corr}(m^*,n^*+1) \tag{2.19}$$

These coefficients can directly be evaluated from inspection of the look-up table.

So, by using equations (2.15)-(2.19) the spatial distortion correction can be applied to non-integer coordinates. This interpolation procedure is included in the code of ShellCheck_4d.m in lines 466-493. The first four lines determine the indices of the four pixels nearest to the spot's center of gravity. Subsequently, the shifts in both indices of these four pixels are retrieved from the lookup table. Lines 488-493, finally, evaluate the interpolation equation. When the corrected center of gravity coordinates are known, the correct values of the other spot characteristics (R, 2θ, η) can also be computed. Note that here the corrected beam center location as computed at the start of ShellCheck_4d (lines 40-43) is used.

This concludes the peak characterization routine. *Total_Intensity*, the output variable already declared in line 57, is now filled with all the spot's characteristics. These include the details of the image in which the original peak was found, the corrected center of gravity coordinates, the total spot intensity, and the final dimensions of the peak box. *Total_Intensity* is returned to the calling program, which is generally the main program Analyze_4d.m. Here, the results are appended to the list of previous calls to ShellCheck_4d.m. In this manner, a new list emerges (*Total_Peak_Intensities*) which, when ShellCheck_4d has been called for all peaks, will contain the intensities and spatial information of all spots that have been identified.

However, a single spot on a detector does not necessarily constitute a reflection of its own. After all, due to the overlap between subsequent layers, a single grain is likely to produce intensity at multiple subsequent slit settings. So, spots coming from different parts of the same grain still need to be grouped together. This is where ArrangeSpots.m comes in.

2.7. Spot grouping – ArrangeSpots.m

2.7.1. Global outline

The tasks of the routine ArrangeSpots.m are twofold. Firstly, the routine takes the lists of spots as created by the repeated calling of ShellCheck_4d.m, and groups together those spots that come from different parts of the same grain and together form a single reflection. Secondly, when such a reflection has been identified, the routine computes the total intensity belonging to this reflection. Due to the overlap between subsequent slit positions, this procedure is more complicated than just a straightforward summing of the component intensities.

Figure 2.12 shows a flow chart of ArrangeSpots.m. When the routine is called, first of all the list of spots is retrieved and, if desired, cut so that only part of the entire collection of spots needs to be analyzed. When this is done, the first spot of the resulting list is taken. A search is conducted for spots within the list that lie on approximately the same location on the detector at the same value of ω; these spots are possible candidates for originating from (a different part of) the same grain as the initial spot. The resulting collection spots are then divided into spot groups, based on connectivity properties of the spots in (*stripe*,*layer*)-space (which translates one-on-one to the (z,y)-system in laboratory coordinates). From here on, each spot group is treated as a single reflection and hence as originating from a single, specific grain. Center of mass coordinates of the reflection on the detector, as well as within the sample volume are computed. The latter can be seen as a first estimate of the location of the diffracting grain in real space. Furthermore, the profile of reflection intensity versus position (in terms of layer number) is derived, giving a first rough indication of grain shape.

When these computations have been performed, the next spot is taken. If this spot has already been grouped in a previous iteration, the routine moves to the next spot on the list. If not, the analysis is performed again. When all spots have been grouped, the routine ends and returns two matrices to the calling program. One contains all the original spots, but grouped and numbered so that the individual spots making up a certain reflection can be inspected. The second contains, for each reflection, the center of mass coordinates and the total integrated intensity.

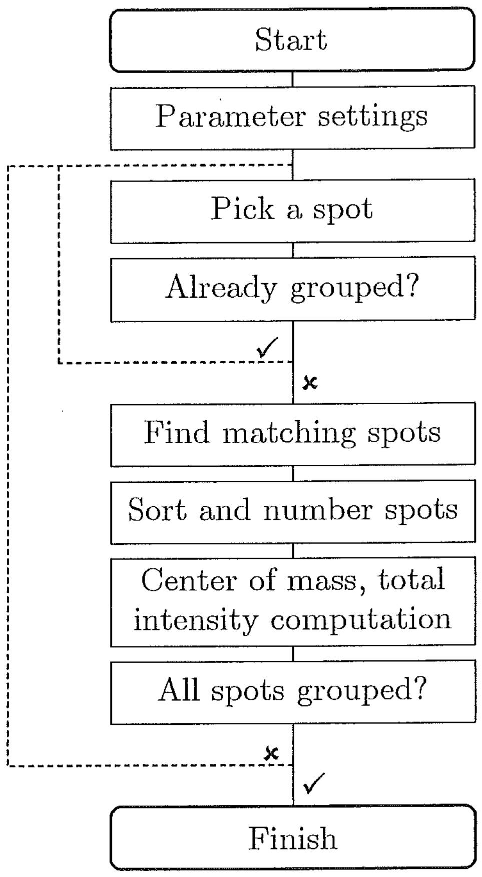


Figure 2.12: Flow chart of the spot grouping process. The routine finds spots that are likely to be part of a single specific reflection, and groups and numbers these spots. Subsequently, the center of mass and total intensity profile of the resulting reflection are computed.

Furthermore, during the routine files are created that list for each reflection the profile of corrected intensity versus layer position.

2.7.2. Initial settings

The code of ArrangeSpots.m starts on page 96. The input to the routine is the location of the list containing the spots (*spotlist*) and two integers that allow the user to only analyze a specific part of the list of spots (*startnr* and *endnr*). This provides the possibility of reduced computational load if one is only interested in the analysis of part of the set of spots. The routine starts of with some of the parameter settings that were already introduced in earlier routines. However, the tolerance level set in line 31 deserves some special attention. As explained, after picking a specific spot the routine searches for spots that lie on approximately the same location on the detector as the spot under consideration. The tolerance level *tol* (in units of pixels) quantifies this concept of 'approximately the same location'. *tol* sets the maximum difference between the row or column coordinate of the spot under consideration and of the spots that are designated as lying on 'approximately the same location'. The value for this tolerance level depends on such aspects as the mosaicity of the grain and the average grain size, and is determined through trial-and-error in combination with a visual inspection of the list of spots.

The input parameters *startnr* and *endnr* define which part of the spot list to analyze. The main program Analyze_4d.m uses the values 1 and *length_ TPI* for this, where the latter stands for the length of the spot list. In this manner, the entire list is analyzed in one run. Still, by allowing for the analysis of only a subset, the routine becomes more flexible to use. However, this does introduce an extra complication. Since spots originating from the same grain are grouped together into a single reflection by the routine, it is imperative the assortment of spots to be analyzed contains all spots that could come from the same grain. In other words, when analyzing a certain ring at a certain ω-setting, the routine needs to have access to all spots lying at those specific settings. Allowing the user to set the limits of the list of spots that are analyzed introduces the risk of the list being incomplete for certain ω-values.

This problem is overcome using lines 53-75. The list of spots is provided as input using the variable *spotlist* (note that, although generally the routine expects *spotlist* to be a matrix representing the list itself, it can also be entered as a string stating the location of the list on disk; in that case, line 53 should be activated and line 54 should be excluded from the routine). This list is first sorted in ascending order on the basis of the following spot characteristics (in order of importance, from most to least importance): diffraction ring – round(ω_{CoG}) – m_{CoG} – n_{CoG} – *stripe* – *layer*. Here, round(ω_{CoG}) represents the rounded value of the center of gravity ω-value of the peak. This sorting is the first step in arranging the spots in the correct manner. After all, spots that originate from different parts of the same grain and together form a single reflection must all lie in the same ring and at the same ω-value. Since the computation of the center of gravity ω-value in ShellCheck_4d might have introduced some small deviations in ω, these value are rounded before the sorting is performed.

Now that the list is sorted in the appropriate manner, the values of *startnr* and *endnr* are refined so that for each ring at a certain ω-value, either all spots or no spots are included in the list to be analyzed. This refinement takes place in lines 60-71. *startnr* (*endnr*) is decreased (increased) until it designates the first (last) entry for the (*ringnumber*,ω)-combination of the initial values of the parameter. The dataset defined by the final values of the two is taken from the spot list, and an extra column is added in front of it which will be used to tick off the individual spots that have been analyzed; the resulting matrix is termed A. Finally, some counting variables and the output variables M and N are declared (78-82), after which the sorting of the spots in A can commence.

2.7.3. Locating matching spots

Line 84 starts the actual spot sorting by starting a loop over all entries within the matrix A. The counting variable *nr_ start* indicates the entry of A under consideration. This parameter is incremented with a value of 1 during each iteration of the `while`-loop. For a certain spot, first a check is performed whether the spot hasn't already been indexed during a previous iteration of the loop. If this is the case, then the spot will have been ticked through a change of the entry in its first column from 0 to 1. In this case the spot can be skipped, and the routine advances to the next iteration of the loop by incrementing *nr_ start* and ending the current iteration using the `continue` command (lines 84-88).

When the spot hasn't been considered yet, it will be used as the basis of a new reflection. A matrix *GR* is introduced (line 95) that will be filled with all the spots that lie on approximately the same location on the detector and at the same ω-value as the spot under consideration, and therefore possibly originate from the same grain as the original spot. Remembering that A was sorted on the basis of *ringnumber*, then round(ω_{CoG}), then m_{CoG}, the `while`-loop of line 103 first identifies all spots for which these three parameters fulfill the requirements for the spots to originate from the same grain. For the value of m_{CoG}, a tolerance level of $2\times tol$ is used. This translates into a tolerance of *tol* in either direction, since the m_{CoG}-value of the original spot will always be the minimum value because this the matrix is sorted in ascending order based on this exact characteristic. The last entry that fulfills these criteria is indicated by the counter *nr_ end*.

The list A(*nr_ start*:*nr_ end*,:) consists of all spots that lie in the same ring, at the same ω-value, and in approximately the same row as the original spot. The last criterion that needs to be fulfilled by a spot for it to be included in the rest of the loop is that the column coordinate needs to be approximately the same as that of the original spot, n_{CoG}. This criterion is formulated in line 122 (together with the demand that the spot hasn't been considered in an earlier iteration). Each spot that fulfills this criterion is written into *GR*, and is ticked off in the spot list A. Finally, *GR* is sorted in ascending order on the basis of *stripe* – *layer* – m_{CoG} – n_{CoG} (in order of decreasing importance). This facilitates the subsequent grouping and numbering procedure.

2.7.4. Grouping and numbering of spots

GR eventually consists of all spots on the same ring, at the same ω-value, on (approximately) the same location on the detector. These spots are likely candidates to all originate from one single reflection from a specific grain. If this is the case, then the spots should form a single, connected group. The meaning of 'connected' in this case is illustrated by Figure 2.13. This figure depicts the desired results in terms of reflection identification for various groups of spots. The group is depicted in terms of the value for *stripe* and *layer* of the various individual spots within the group. When a spot with a certain (*stripe*,*layer*)-combination is present, this is represented by a black circle. The white circles represent missing spots. A| and B| show groups of which the spots all lie in the same stripe; C| and D| illustrate the principle in case the spots are distributed over two neighboring stripes. Similar illustrations could be drawn for the case of three stripes.

A| depicts the simplest case (barring a group containing just 1 entry). The layer numbers of the spots form an increasing sequence in which no entry is missing. This indicates that, as the beam scanned the sample, at a certain moment a specific grain generated a reflection. This reflection is spread out over multiple layer settings, since subsequent settings overlap (and since the grain size might very well exceed the layer width). So, the reflections is visible in multiple, subsequent diffraction images (at a specific value of ω), after which it disappears again. Since these spots all lie on

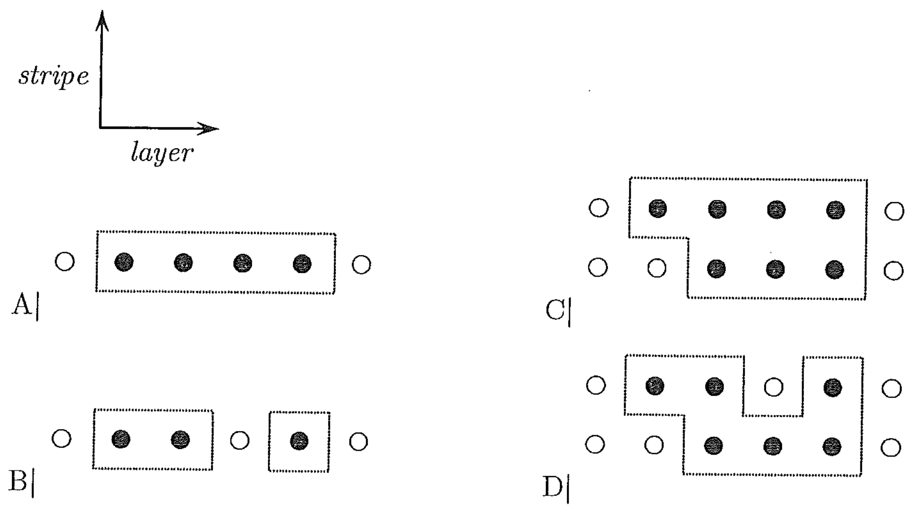


Figure 2.13: Schematic illustration of the connectivity property of a group of spots, showing the presence (black) or absence (white) of spots at a specific location on the detector for a specific ω-value as a function of the illuminated part of the sample in terms of *stripe* and *layer*. Connected spots are treated as belonging to the same reflection (A, C). Absent spots can result in the identification of two reflections (B). However, spots in neighboring stripes can also provide connectivity (D).

the same location and in subsequent images, it is very likely that they all belong to the same reflection. So, they are grouped together and are all assigned the same reflection number.

However, situation B| could also occur. In this case, the spots all lie on the same location on the detector and in the same stripe, but one or more spots are missing, and therefore the spots do not form a connected group. The spots are then divided into two (or more) groups, since it could be that they do indeed originate from different grains. The groups remain separated and each group receives its own reflection number. Still, it could also be that the only reason that one spot is missing is an unexpectedly high background intensity. In that case, a single reflection would incorrectly be split into two. It is therefore desirable to keep track of these groups, so that later on in the analysis the two groups can be added together again in case additional evidence is found that the two should indeed form a single group. This is done by numbering the groups not with integer numbers, but with decimal numbers of the same integer (for instance 2.01, 2.02 etc.).

C| depicts a scenario similar to A|, but here the spots are located in two stripes. This situation does, however, introduce an extra complication in determining whether a number of spots form a connected group or not, as shown in D|. Here, the spots in the upper stripe appear to form two distinct groups; in fact, their pattern is equal to that of B|. However, the spots in the second stripe 'bridge' the missing entry, resulting in a single connected group of spots. So, the spots are all numbered equally again. As mentioned, similar reasonings apply to the case where the spots are spread out over all three stripes.

The various scenarios of Figure 2.13 indicate some of the issues surrounding the grouping and numbering of the spots within the list *GR*. Allowing for all different possibilities requires quite some coding: lines 154 up to 542 of the MATLAB code are only about the grouping and numbering. *GR* is first classified based on the number of stripes in which the spots lie. Line 156 checks if *GR* contains more than 1 entry. If not, the single spot is numbered as a reflection (using the counting variable *reflnumber*) and lines 162-538 can be skipped. The situation also remains relatively straightforward in case all spots come from only a single stripe. Since *GR* is ordered on the basis of stripe number, this is easily tested by checking if the values of *stripe* for the first and last element of *GR* are equal (line 162). If so, then the scenario created by the spots resembles either A| or B| from Figure 2.13. Line 167 checks which of the two it is, by determining whether or not the layer numbers of the spots in the group form an incrementing sequence or not. If not, then the spots will be grouped using decimal numbers (line 168-178). If so, then all spots receive the same number *reflnumber*. Note that dataset *e* was recorded using only a single stripe, and therefore the analysis of this set is significantly less complicated than that of set *d*.

In case the spots come from more than one stripe, an extra step in the analysis is required. This step is related to the fact illustrated by scenario D| of Figure 2.13, namely that connectivity of the spots in a certain stripe can also be delivered by the spots in the neighboring stripe. In the code, the cases of spots from two stripes or from three stripes are treated separately (lines 182-322 and 323-538, respectively). In this treatment the case of three stripes will be discussed; the situation for two stripes is comparable.

The spots within *GR* are recognized as coming from three stripes if the difference between the stripe number of the first and last entry of *GR* is not equal to 0 (all spots from a single stripe) or 1 (all spots from 2 consecutive stripes). Note that, strictly speaking, the spots could come from only the two outer stripes, but this is quite unlikely since a grain exhibiting diffraction in stripe 0 and stripe 2 is also expected to diffract in stripe 1. When the spots are identified as coming from all three stripes, first of all the number of spots is compared to the value of $3(layer_{max} - layer_{min} + 1)$. In case the number of spots within *GR* equals this value or is only 1 or 2 spots smaller, then the spots will always form a single, connected group, and can therefore all be numbered equally. This principle can be understood by referring to Figure 2.14.

The expression $3(layer_{max} - layer_{min} + 1)$ describes the maximum number of

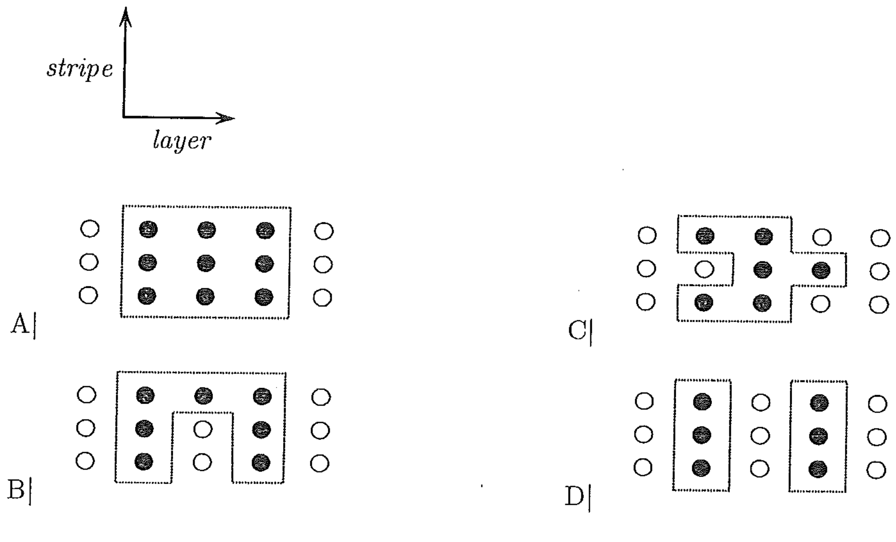


Figure 2.14: Illustration of how the number of spots within a group can indicate the spots' connectivity when they are located in all three stripes. For explanation of the symbols, see Figure 2.13. The number of stripes times the difference between the maximum and minimum layer number determines the possible number of spots (A). If two spots or less are missing, the result is always a single group (B). If more are absent, the result depends on the location of the missing spots (C, D).

spots expected in *GR* when the spots lie in three stripes, as a function of the maximum and minimum layer number of the spots, $layer_{max}$ and $layer_{min}$ respectively. In case all these spots have been identified, the situation at hand will resemble scenario A|. However, there might also be some missing spots. Still, as long as the number of absent spots is no more than 2, the spots in the group will still form a single connected group and can therefore be treated as belonging to the same reflection; see B|. When the number of missing spots exceeds this value, the number of resulting spot groups depends on the location of the missing spots in (*stripe*,*layer*)-space (C, D). A similar reasoning can be applied to the case of two stripes, only here the maximum number of missing spots for the remaining spots to automatically form a single group is only 1.

Returning to the code, lines 328-334 identify compositions of *GR* that resemble Figure 2.14, scenario B|. In this case, the spots are all assigned the same number, concluding part 1 of the routine. If the criterion is not satisfied, *GR* needs to be analyzed in a more thorough manner to determine the way in which the spots have been distributed over (*stripe*,*layer*)-space. This starts with determining how many spots are located in the various stripes. These numbers are assigned to the variables *k1*, *k2* and *k3* for the first, second and third stripe respectively (lines 340-348). Then, the spots in the three stripes are divided into groups by regarding each stripe separately; in other words, connectivity provided by spots in a neighboring stripe is ignored for now and numbering takes place according to scenario A| or B| of Figure 2.13. Lines 351 through 402 take care of this numbering; note the resemblance between these lines and lines 167-181, which perform the spot numbering in case *GR* contains spots from only a single stripe.

The spots in each stripe have now been grouped and numbered independently from the other stripes. The next step is linking the stripes together again and performing the renumbering procedure. The first part of this process is comparing the locations of spot groups in stripe 3 and 2, and determining if there are any connections between groups in these two different stripes. This starts on line 426 with a loop over all spot groups in the third stripe. For each group, a new loop is started over all spots within that group. The value of *layer* of such an individual spot is then compared to the layer numbers of all spots in the second stripe. If a match is found (determined by the `if`-criterion on line 436), then the group to which the spot in the third stripe belongs is renumbered with the *reflnumber* of the matching spot found in the second stripe (440-441). In this way, connectivity between the two spot groups is acknowledged. When connectivity has been established, an extra check is performed to determine whether or not the spot group in stripe 3 links up to even more groups in stripe 2 (443-465). If so, these are also renumbered using the *reflnumber* of the first matching group in stripe 2. The loop then moves on to the next peak group in the third stripe. Once all groups in stripe 3 have been checked for connections with groups in stripe 2, the process is repeated for stripes 2 and 1.

The last part in the numbering process is to renumber the spot groups to make sure the reflections numbers form an increasing sequence again. After all, as a result of the renumbering of the various spots of groups the reflection numbers might not be increasing with a steady pace anymore (for instance 2.01, 2.03, 2.04, 2.06 instead of 2.01-2.04). This final renumbering step is performed by means of lines 546 till 559. Finally, *GR* contains the individual spots, grouped together in groups which each represent an individual reflection. *GR* is appended to the matrix *M*, which is the final output matrix (564).

2.7.5. Center of mass, intensity profile computation

Now that all spots belonging to a specific reflection have been localized and grouped together, the center of mass coordinates of that reflection as well as the total integrated intensity can be computed. This is done in part 2 of ArrangeSpots.m. The main operation during this part of the routine is the reconstruction of the (intensity,position)-profile.

The idea behind this reconstruction can be understood by referring to Figure 2.15. This figure illustrates how the overlap in subsequent slit settings results in an increased resolution and how the intensity profile can be deconvoluted using this overlap. The figure depicts multiple layers (n through n+5). Since the shift between two successive layers is less than the layer width (b_{beam}), the intensity recorded at for layer n+1 will contain a lot of intensity that was also recorded in layer n. This implies that a simple summing of the intensities of the individual spots which make up a reflection is incorrect when one wants to determine the total integrated intensity of a reflection. To solve this issue, the volume from where the reflection originated is divided into smaller parts. This division is dictated by the layer overlap. The layer overlap is not constant but varies between 10 or 7.5 μm, and therefore the subvolumes are not of equal length either. The figure indicates the typical periodicity of the sizes of the subvolumes (A-F); most of them are 2.5 μm in width, but every fifth volume has a width of 5 μm. The intensity originating from a specific subvolume can now be calculated as a weighted average of fractions of the spot intensities that originate (partly) from the subvolume in question. For instance, the intensity coming from volume A, I_A, can be computed as an average of parts of the intensities I_n, I_{n+1} and I_{n+2}:

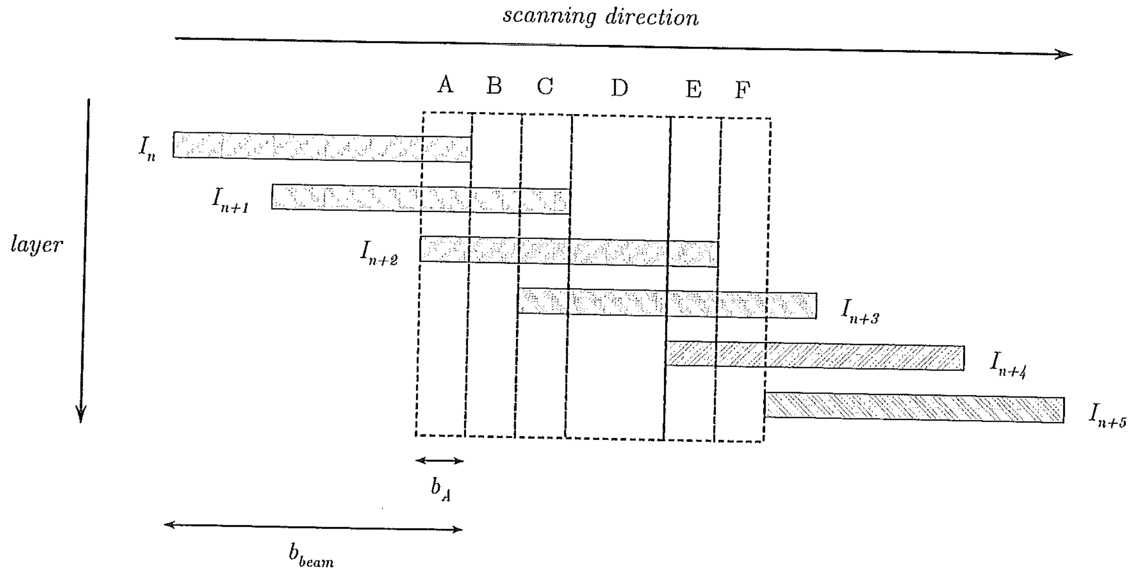


Figure 2.15: Schematic illustration of the reconstruction of an intensity profile from its component intensities using the layer overlap. The image shows layers overlapping in the scanning direction; for reasons of clarity they have been separated from each other vertically in the figure. The overlap creates an increased resolution in the scanning direction, allowing the intensity coming from for instance area A to be computed as a weighted average of fractions of I_n, I_{n+1} and I_{n+2}.

$$I_A = \frac{1}{3}\frac{b_A}{b_{beam}}\left(I_n + I_{n+1} + I_{n+2}\right) \tag{2.20}$$

Each intensity is weighed using by the fraction of the corresponding layer located within A, after which the average is computed. Layers n, n+1 and n+2 all cover volume A, so the averaging is carried out over the three corresponding intensities. Expressions similar to (2.20) can be constructed for the intensities of the other subvolumes. The result is a profile of intensity versus illuminated sample volume with a resolution of 2.5 μm (except for every fifth point, which has a resolution of only 5 μm). The profile can be used to compute center of mass coordinates of the reflection in terms of location on the detector as well as origin of diffraction within the sample.

Returning to the code, the second part of the routine starts on line 567. A loop is started over all reflections in *GR*. For each group, it is then determined in how many stripes the group's spots are located. For dataset *e*, clearly, this will always be 1, since this set was recorded using only a single stripe. The spots of dataset *d*, however, can be located in up to three stripes. For each amount of stripes, a separate block of code is written. The following will treat the code written for the case in which the spots are located in all three stripes, but the workings of the other parts of the code are comparable.

The code for the three-stripe scenario starts on line 980. Firstly, three matrices are declared (one for each stripe), and the amount of spots in each stripe is determined. Subsequently, the three matrices are filled with the starting and finishing coordinates of the layers associated with the spots in the stripe in question (lines 999-1085). Take for instance the filling of matrix *L1*. A loop is initiated over all spots of the current reflection that lie in the first stripe. The starting point of the first layer is defined as zero, and since the size of all layers in the overlapping direction is 15 μm, the finishing point of this layer lies at 15 μm (lines 1001 and 1002). Subsequent layers are translated with respect to this first layer. Their starting and finishing coordinates are computed in lines 1003-1016. The layer number of the spot under consideration is compared to that of its predecessor. As explained in subsection 3.1.2.1 of the thesis report, the periodicity of the layer shifts is 5 – 7.5 – 5 – 7.5 – 5 (μm). Because of this regularity, one can deduce from inspection of the layer number of a certain spot how far it has been shifted compared to the previous layer. This is performed using the `if/else`-block of lines 1006-1014, by evaluating the remainder of the layer number divided by 5. When the coordinates of the spot in question have been determined, they are written into matrix *L1*. Also included in this loop is the calculation of the center of mass coordinates of the reflection in terms of location on the diffraction image (*row*,*column*,ω) as well as in terms of the origin of diffraction (*stripe*). Note, however, that the computation of the center of mass layer coordinate is not included. This calculation is not performed until the intensity profile has been reconstructed, since this profile offers the opportunity to calculate the layer coordinate with more precision.

When the matrices *L1*, *L2* and *L3* have been filled, one additional operation is required. The starting and finishing coordinates contained in the matrices are all defined with respect to the starting point of the first layer in the corresponding stripe. However, in order to later perform the calculation of the weighted average layer coordinate, it is required that the coordinates in *L1*, *L2* and *L3* are all defined with respect to the same starting point. Therefore, generally the coordinates in two of the three matrices need to be shifted by a certain value. This operation is performed in lines 1108-1193. The obvious starting point is the start of the leftmost layer belonging to the reflection. Lines 1108-1109 first determine whether the reflection happens to start in the same layer for all three stripes. If so, then no shift is required for any of the three matrices, and only the variable *layer_ start*, which indicates the number of the leftmost layer of the reflection and which is required later on in the routine, needs to be declared. If there are differences between the starting points of the reflection in the various stripes, then a `switch/case`-statement is executed. Line 1113 determines which of the three stripes starts with the lowest-numbered layer.

When this has been determined, the differences between this layer number and the starting layers of the other two stripes are calculated. The entries in the two corresponding matrices are subsequently translated so that their starting points line up with that of the third matrix. By how much the coordinates need to be translated again is determined using the periodicity in the layers, like with the filling of the matrices itself.

So, the result of the operations above is a set of three matrices that contain the starting and finishing coordinate of each of the reflection's spots, together with the intensity of the spot. The next step is reconstructing the intensity profile from these matrices. As explained earlier (in relation to Figure 2.15), the appropriate step size for the intensity profile is 2.5 μm (in cases like the 5 μm wide area D in the figure, the area is divided into two parts of 2.5 μm in width which both receive an equal value for the corresponding intensity). For each volume defined in this manner, the intensity originating from this volume is computed by an averaging procedure similar to that of equation (2.20). These calculations are carried out for the three matrices in lines 1201-1248. The computational procedures can be clarified by taking a closer look at the code for matrix *L1*, for instance (lines 1201 till 1216). A loop is initiated over all 2.5 μm steps in this stripe, the amount of which (*stepnr1*) has been computed in line 1195. The first column of matrix *Int_corr1* is filled with the starting points of these 2.5 μm steps. Lines 1205-1212 then execute a loop which performs a search within matrix *L1* for all layers that cover the subvolume under consideration. If a layer does cover the subvolume, then this is indicated by the fact that the starting point of this layer is equal to or smaller than the starting point of the subvolume, and the finishing point is larger than the starting point of the subvolume. If both these criteria are fulfilled (this is tested in line 1206), then the intensity of the corresponding layer is added to the second column of *Int_corr1* and the counter *divider* is incremented by a single point. When the loop is finished, the summed intensities are divided by this counter, resulting in the average intensity for the 2.5 μm wide subvolume in question.

When these calculations have been performed for all three matrices, the results need to be written away to disk. One large matrix is constructed which contains the corrected intensity values for all the 2.5 μm wide subvolumes. This matrix is then written to disk as a file named grain****.txt, in which the **** represent the reflection number. In this way, the intensity profiles of all individual reflections can be reexamined after the analysis.

Apart from the creation of the output matrix *N*, one last important calculation still needs to be performed: the computation of the center of mass layer coordinate. This can now be done, using the intensity profile to arrive at a higher resolution.

First of all, for each 2.5 μm wide subvolume the coordinate of the middle of that volume is multiplied by the intensity attributed to it. By summing these values and dividing by the total intensity, the center of mass layer coordinate is computed in terms of the shift from the leftmost point of the reflecting volume. This shift (in μm) now needs to be converted to a layer number. First of all, every 30 μm shift corresponds to 5 layers (following from the layers' periodicity). This is accounted for in line 1306. The amount of μm in shift left is then taken into account in lines 1308-1317. Finally, line 1318 subtracts a value of 0.5 from the center of mass coordinate as computed so that an integer layer value corresponds to the middle of that layer and not to the edge. This preserves the consistency with the other center of mass coordinates computed earlier.

At the end of the routine, the output matrix N is created. For each reflection, N contains its number, its center of mass coordinates, and its total corrected intensity. The total output of ArrangeSpots.m thus consists of M (the matrix containing the reflections with their individual spots), N (containing the aggregate information on the individual reflections), and the set of grain****.txt-files (each containing the intensity profile for a specific reflection).

2.8. Reflection coupling – GrainSpotter

Having reconstructed all individual reflections, the next step of the microstructure reconstruction consists of linking the reflections to actual grains. Each grain within the illuminated sample volume will produce reflections at multiple values of ω. The positions of these reflections in terms of location on the detector and ω-value are determined by the crystallographic characteristics of the grain in question. Therefore, by scanning for groups of reflections that match the crystallographic criteria imposed by the material, reflections originating from the same grain can be grouped together. Furthermore, from as little as two independent reflections the grain's crystallographic orientation can be derived (in case the lattice parameters are unknown: three reflections) [15]. Linking together the individual reflections is therefore a crucial step in the reconstruction of the original microstructure.

This reflection matching, also called 'indexing', was performed using an alpha version of a program called GrainSpotter [3]. GrainSpotter, developed by dr. Søren Schmidt of Risø National Laboratory in Denmark, is based on earlier software called GRAINDEX [4]. GRAINDEX is a program designed for the image processing and indexing parts of the analysis of 3DXRD data. GrainSpotter is a stand-alone program performing only the latter of the two.

Theoretically, there are three different criteria on the basis of which one could index the individual reflections: the calculated coordinates of the diffracting volume, the crystallographic characteristics of the structure, and the total integrated intensities of the reflections [4]. Of these three, the latter is the least reliable. An important reason for this is the complications that arise with grains lying near the boundaries of the illuminated volume. When the sample is rotated about ω, these grains will oscillate in and out of the illuminated area. This can lead to these grains being only partly illuminated during diffraction, resulting in significantly lower intensities as compared to diffraction when these grains are fully illuminated. The criterion based on the calculated coordinates of the diffraction origin, on the other hand, is only applicable when the uncertainty with which these coordinates are determined is much smaller than the grain dimensions. The criterion most generally applicable is that of the crystallography of the material. Therefore, this is the criterion used by GrainSpotter when indexing the individual reflections.

GrainSpotter indexes the reflections by a stepwise scanning of Euler space, calculating the expected diffraction vectors as a function of the simulated crystallographic orientation of a diffracting grain and checking whether or not these vectors have been recorded. Grains are identified on the basis of two criteria: completeness and uniqueness. The completeness criterion states that the number of reflections found for a certain assumed crystallographic orientation, M_{exp}, should not be much smaller than the theoretically expected number of reflections for that orientation, M_0. This is quantified through the following expression:

$$M_{\exp} \geq (1-\alpha_c) M_0 \tag{2.21}$$

in which α_c is a dimensionless parameter determining the stringency of the criterion. The uniqueness criterion dictates that the set of matching reflections is not allowed to be a subset of the set of matching reflections for another simulated orientation.

When these two criteria are satisfied, the group of reflections is assigned to a grain with the crystallographic orientation under consideration. If no group of reflections can be constructed that fulfills the completeness and uniqueness criteria, it is inferred that no grain is present with the orientation under consideration. For a more detailed description of the indexing procedure, the reader is referred to [4, 16].

2.8.1. GrainSpotter input

Although the software for the reflection matching was available from the Danish Risø National Laboratory, an additional routine still needed to be written. This routine, InputGrainSpotter.m, was required to convert the output from ArrangeSpots into the format demanded by GrainSpotter.

The GrainSpotter input file can be divided into two parts. The first part contains information on the crystal structure of the material under investigation. Firstly, it describes the unit cell in terms of lattice parameters, angles and space-group symmetry. Subsequently, the wavelength of the x-rays used and the wedge angle are required. The latter is related to the angle between the incident beam and the ω rotation axis; in the current project, where the two are perpendicular, the wedge angle equals zero. These two values are followed by an enumeration of all reflections under consideration, together with their associated reciprocal lattice spacing. Because this part of the input file is well-defined, its creation is not included in the routine InputGrainSpotter. The routine only creates the second part of the file, which is subsequently appended to the first part which has been constructed manually at an earlier stage.

The second part of the input file consists of the details of the experimentally observed scattering vectors. The scattering vector **G** is defined as the difference between the scattered beam vector s and the incoming beam vector s_0 [17]:

$$\mathbf{G} = \frac{1}{\lambda}\left(\mathrm{s} - \mathrm{s}_0\right) \tag{2.22}$$

in which both s and s_0 are unit vectors, and λ represents the wavelength of the radiation used. It follows that the modulus of the scattering vector equals

$$G = \|\mathbf{G}\| = \frac{2\left|\sin\theta\right|}{\lambda} \tag{2.23}$$

in which θ equals half the weighted average diffraction angle of the reflection under consideration.

The value of **G** can be determined from the average characteristics of the reflection under consideration. For input into GrainSpotter, **G** is required to be expressed in the laboratory coordinate system that has been used before in this thesis. The system is shown in Figure 2.5. In this system, indicated with the subscript l, the normalized scattering vector can be seen to depend on θ and η in the following manner [16]:

$$\frac{\mathbf{G}_l}{\|\mathbf{G}_l\|} = \begin{bmatrix} G_{l,x} \\ G_{l,y} \\ G_{l,z} \end{bmatrix} = \cos\theta \begin{bmatrix} -\tan\theta \\ -\sin\eta \\ \cos\eta \end{bmatrix} \tag{2.24}$$

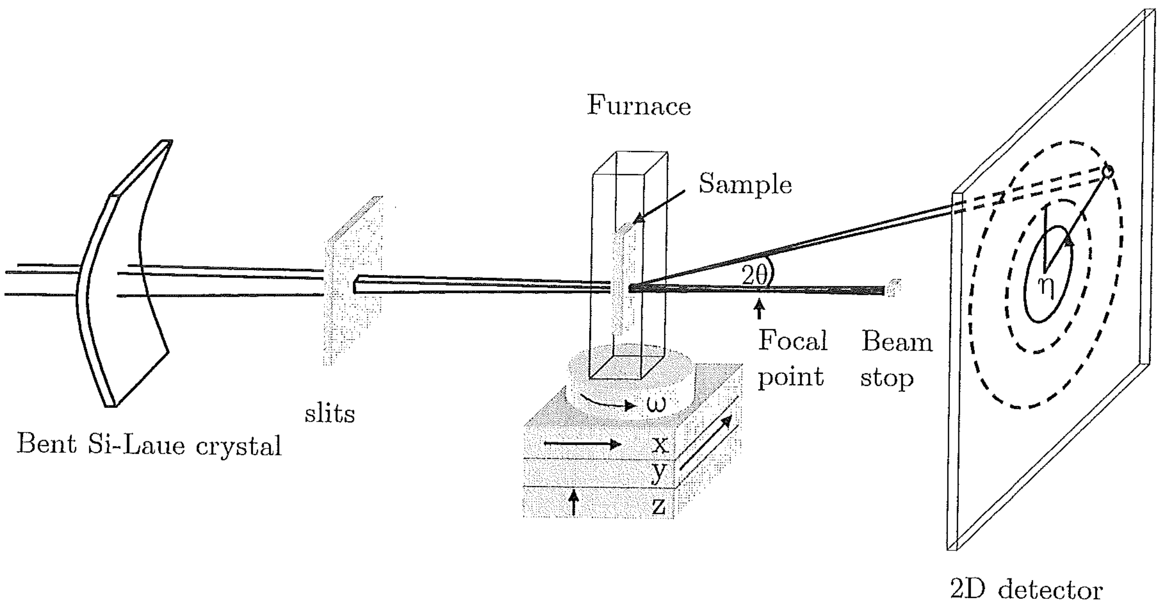


Figure 2.16: Schematic representation of the setup of the three-dimensional x-ray diffraction microscope at beamline ID11 of the ESRF used for the experiments under consideration. The setup consists of a bent Si-Laue crystal, slits, and a two-dimensional detector. The sample is positioned in a furnace which is mounted on a table, allowing the sample to be translated and rotated. Figure taken from [1].

After having obtained the normalized scattering vector from this equation, scaling to the desired length can be performed using equation (2.23). Apart from the scattering vector $\mathbf{G}_l$ itself, GrainSpotter also requires its modulus as well the reflection's weighted average location on the detector and the angles η and ω to be entered as input.

The code of InputGrainSpotter is included in chapter 3, starting on page 119. The routine starts with some parameter declarations, after which the output matrix *gvecs* is initialized. Starting on line 21, a loop is evaluated that computes the scattering vector and scattering vector modulus for each reflection. Lines 35-43 write the required output into *gvecs*. For more details of the exact input format required by GrainSpotter, the reader is referred to [3].

2.9. Grain characterization – CharacterizeGrains.m

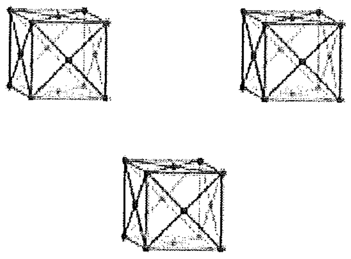
When GrainSpotter has produced the groups of reflections originating from the same grains, the final step of the microstructure reconstruction can be performed. This step entails using the characteristics of the individual reflections assigned to a certain grain to reconstruct that individual grain in terms of crystallographic orientation and location of center of mass.

The orientation of the grain is calculated and provided by GrainSpotter. It is expressed in the form of an orientation matrix based on the well-known Euler angles, a common way of describing a crystallographic orientation. For a concise description of Euler angles in relation to three-dimensional x-ray diffraction microscopy, see for instance [16], section 3.3. The location of the grain's center of mass, however, is not provided by GrainSpotter. It needs to be computed by the user, based on the origins of the individual reflections attributed to the grain in question.

However, the output format of the GrainSpotter results is not particularly suitable for further computational analysis. In fact, many of the data required for calculation of individual grains' characteristics need to be retrieved again from the lists created by ArrangeSpots.m. This process is carried out by CharacterizeGrains.m. The code of this routine is provided in chapter 3, starting on page 120.

The required input for the routine are the locations of both the output file from the GrainSpotter analysis, as well of the file containing the individual reflections created by ArrangeSpots.m. Since the GrainSpotter output file has a well-defined format (described in [3]) the locations of the desired data are known. This information is used in analyzing the file.

The file consists of a combination of text and numerical data. Line 15 defines *offset*, a constant representing the number of lines in the header of the file. These lines are skipped during the analysis. The first line of the file, though, contains the number of identified grains. This number, *nrgrains*, is retrieved in line 18 of the code. When this value is known, some variables are initialized, after which a loop is started over all the grains identified (line 28). By constantly reading and skipping specific lines of the file, the required data can be retrieved. For instance, the numbers of theoretically expected and experimentally observed reflections, *M0* and *Mexp*, can be retrieved, as well as the grain's orientation matrix *UGrain*. However, for information on the grain's center of mass, the reflections need to be traced back to the reflection list used as input to the routine. This is done within the second `for`-loop. Each of the reflections attributed to the grain in question is retrieved from the list of reflections produced by ArrangeSpots. The required computations can then be performed on the resulting list of reflections. As an example, CharacterizeGrains produces the list of reflections called *GrainCharacteristics* and saves this file, naming it using the grain number retrieved from the GrainSpotter output. In this manner, the link can be made between the individual reflections and the grain's attributes.



3. MATLAB code

A considerable part of the results of this thesis consists of the actual MATLAB routines. The individual routines are discussed in chapter 2. The current chapter provides the exact code of the routines treated there. Each section introduces an individual routine, starting on a new page. Line numbers have been added to allow for simple comparison with the descriptions given in chapter 2.

3.1. SDCorrection.m

```
%-------------------------------------------------------------------------
%Start of: SDCorrection.m
%-------------------------------------------------------------------------
%
%SDCorrection computes a spatial correction as described by splinefile to
%all of the points on a detector, and subsequently trims the resulting
%matrix to contain the distortion values only for pixels lying in the
%Region of Interest (ROI) on the detector, defined by the outermost mask.
%
%Note: to facilitate processing of the splinefile, I have manually added
%some blanks in the file so that all of the B-spline coefficients are
%separated from the following and preceding number. Though this a small
%effort, one could also incorporate this in the code.
%
%TvdZ. Last edits: February 2007.

function [SDMatrix] = SDCorrect(splinefile);

%Declaration of the degree of the splines that have been computed (e.g.
%for k=3, the polynomial on each knot interval is of the third power at
%most).

kx = 3;
ky = 3;

%First, the splinefile is read in

fid = fopen(splinefile,'r');
if fid == 2
    disp('Error: file specified by splinefile cannot be opened!')
    return
end

%Because a fit2d splinefile has a well-defined format, splinefile can be
%read quite easily by simply running over all lines and picking up the
%necessary data

line = fgetl(fid); %'SPATIAL DISTORTION SPLINE INTERPOLATION COEFFICIENTS'
line = fgetl(fid); %BLANK LINE
line = fgetl(fid); %'VALID REGION'
line = fgetl(fid); %The actual valid region: xmin ymin xmax ymax
A = str2num(line);
xmin = A(1);
ymin = A(2);
xmax = A(3);
ymax = A(4);
line = fgetl(fid); %BLANK LINE
line = fgetl(fid); %'GRID SPACING, X-PIXEL SIZE, Y-PIXEL SIZE'
line = fgetl(fid); %The actual values for the spacing and pixel sizes
A = str2num(line);
spacing = A(1);
xsize = A(2);
ysize = A(3);
line = fgetl(fid); %BLANK LINE
line = fgetl(fid); %'X-DISTORTION'
line = fgetl(fid); %nx1 and ny1: # of knots in x and y direction, resp.,
                    %for the X distortion spline
A = str2num(line);
nx1 = A(1);
ny1 = A(2);
%Now follow three blocks of lines, each line containing 5 values. First,
%the nx1 values of the x positions of the knots are given. Then, ny1
%values giving the y positions of the knots. Finally,
%(nx1-kx-1)*(ny1-ky-1) (=(nx1-4)*(ny1-4)) values for the b-spline
%coefficients
tx1 = [];
tx1l = ceil(nx1./5); %The number of lines containing x position values
for i = 1:tx1l
    line = fgetl(fid);
    A = str2num(line);
    tx1 = [tx1 A];
end
ty1 = [];
ty1l = ceil(ny1./5); %The number of lines containing y position values
for i = 1:ty1l
    line = fgetl(fid);
    A = str2num(line);
    ty1 = [ty1 A];
end
c1 = [];
nc1 = (nx1-kx-1).*(ny1-ky-1);
c1l = ceil(nc1./5); %The number of lines containing b-spline coeff.
for i = 1:c1l
    line = fgetl(fid);
    A = str2num(line);
    c1 = [c1 A];
end
line = fgetl(fid); %BLANK LINE
line = fgetl(fid); %'Y-DISTORTION'
line = fgetl(fid); %nx2 and ny2: # of knots in x and y direction, resp.,
                    %for the Y distortion spline
A = str2num(line);
nx2 = A(1);
ny2 = A(2);
tx2 = [];
tx2l = ceil(nx2./5); %The number of lines containing x position values
for i = 1:tx2l
    line = fgetl(fid);
    A = str2num(line);
    tx2 = [tx2 A];
end
ty2 = [];
ty2l = ceil(ny2./5); %The number of lines containing y position values
for i = 1:ty2l
    line = fgetl(fid);
    A = str2num(line);
    ty2 = [ty2 A];
end
c2 = [];
nc2 = (nx2-kx-1).*(ny2-ky-1);
c2l = ceil(nc2./5); %The number of lines containing b-spline coeff.
for i = 1:c2l
    line = fgetl(fid);
    A = str2num(line);
    c2 = [c2 A];
end
```

```
%Create the two arrays x and y whose cross product describe all points on
%the detector

x = [];
for i = 1:2048
    x(i) = i;
end
y = x;

%All the required data have been retrieved from the fit2d splinefile.
%Compute the spatial distortions of all individual points.

deltaj = bispev(tx1,ty1,c1,kx,ky,x,y);
deltai = bispev(tx2,ty2,c2,kx,ky,x,y);

%Now correct all points on the detector for their distortion.

SDMatrix11 = ones(2048,1);
SDMatrix2 = [];
SDMatrix2 = [];
SDMatrix3 = [];
SDMatrix4 = [];
for i = 1:2048
    SDMatrix1 = [SDMatrix1; i.*SDMatrix11];
    SDMatrix2 = [SDMatrix2; y];
    SDMatrix3 = [SDMatrix3; deltai(i,:)'];
    SDMatrix4 = [SDMatrix4; deltaj(i,:)'];
end
SDMatrix5 = SDMatrix1+SDMatrix3;
SDMatrix6 = SDMatrix2+SDMatrix4;

%Construct SDMatrix_full.txt, which is a matrix of dimensions (2048^2,6)
%containing: [i_orig j_orig delta_i delta_j i_corr j_corr], where i and j
%are the fast and slow index of the matrix representing the edf images,
%respectively. The i index corresponds to the Y direction in fit2d, and
%the j index to the X direction.

SDMatrix_full = [SDMatrix1 SDMatrix2 SDMatrix3 SDMatrix4 SDMatrix5...
    SDMatrix6];
dlmwrite('E:\4d_4e_Analysis\SDMatrix_full.txt',SDMatrix_full,...
    'delimiter','\t');

%SDMatrix_full = dlmread('E:\4d_4e_Analysis\SDMatrix_full.txt','\t');

%Define the borders of the rectangle that encloses the outermost mask that
%is used in the analysis of the diffraction patterns, and hence defines
%the ROI on the detector. These values have been determined by running
%CreateMask2DTilt.m and inspecting the resulting masks. The borders are
%placed multiple pixels beyond the outermost pixel of the outer mask at
%each side, to create a margin for unexpectancies in the data.

m_min = 340;
m_max = 1759;
n_min = 280;
n_max = 1687;

%First, trim off the parts with m < m_min or m > m_max

sdstart = (m_min-1)*2048+1;
sdend = m_max*2048;
```

```
SDMatrix_full = SDMatrix_full(sdstart:sdend,:);

%Then collect all lines with n >= n_min and n =< n_max

SDMatrix = [];
for i = 1:(m_max-m_min+1)
    SDMatrix = [SDMatrix ; SDMatrix_full((((i-1)*2048+n_min):...
        ((i-1)*2048+n_max)),:)];
end

dlmwrite('E:\4d_4e_Analysis\SDMatrix.txt',SDMatrix,'delimiter','\t');

%-----------------------------------------------------------------------------
%End of: SDCorrection.m
%-----------------------------------------------------------------------------
```

3.2. bispev.m

```
%------------------------------------------------------------------------
%Start of: bispev.m
%------------------------------------------------------------------------
%
%Translated from Fortran into Matlab using the f2matlab package
%
%The following are the comments from the original Fortran file
%
%----------
%
%  subroutine bispev evaluates on a grid(x(i),y(j)),i=1,...,mx; j=1,...
%  ,my a bivariate spline s(x,y) of degrees kx and ky, given in the
%  b-spline representation.
%
%  calling sequence:
%     call bispev(tx,nx,ty,ny,c,kx,ky,x,mx,y,my,z,wrk,lwrk,
%    iwrk,kwrk,ier)
%
%  input parameters:
%   tx    : real array, length nx, which contains the position of the
%           knots in the x-direction.
%   ty    : real array, length ny, which contains the position of the
%           knots in the y-direction.
%   c     : real array, length(nx-kx-1)*(ny-ky-1), which contains the
%           b-spline coefficients.
%   kx,ky : integer values, giving the degrees of the spline.
%   x     : real array of dimension(mx).
%           before entry x(i) must be set to the x co-ordinate of the
%           i-th grid point along the x-axis.
%           tx(kx+1)<=x(i-1)<=x(i)<=tx(nx-kx), i=2,...,mx.
%   y     : real array of dimension(my).
%           before entry y(j) must be set to the y co-ordinate of the
%           j-th grid point along the y-axis.
%           ty(ky+1)<=y(j-1)<=y(j)<=ty(ny-ky), j=2,...,my.
%   wrk   : real array of dimension lwrk. used as workspace.
%   lwrk  : integer, specifying the dimension of wrk.
%           lwrk >= mx*(kx+1)+my*(ky+1)
%   iwrk  : integer array of dimension kwrk. used as workspace.
%   kwrk  : integer, specifying the dimension of iwrk. kwrk >= mx+my.
%
%  output parameters:
%   z     : real array of dimension(mx*my).
%           on succesful exit z(my*(i-1)+j) contains the value of s(x,y)
%           at the point(x(i),y(j)),i=1,...,mx;j=1,...,my.
%   ier   : integer error flag
%   ier=0 : normal return
%   ier=10: invalid input data(see restrictions)
%
%  restrictions:
%   mx >=1, my >=1, lwrk>=mx*(kx+1)+my*(ky+1), kwrk>=mx+my
%   tx(kx+1) <= x(i-1) <= x(i) <= tx(nx-kx), i=2,...,mx
%   ty(ky+1) <= y(j-1) <= y(j) <= ty(ny-ky), j=2,...,my
%
%  references :
%    de boor c : on calculating with b-splines, j. approximation theory
%                6(1972) 50-62.
%    cox m.g.   : the numerical evaluation of b-splines, j. inst. maths
%                 applics 10(1972) 134-149.
%    dierckx p. : curve and surface fitting with splines, monographs on
%                  numerical analysis, oxford university press, 1993.
%
%  author :
%    p.dierckx
%    dept. computer science, k.u.leuven
%    celestijnenlaan 200a, b-3001 heverlee, belgium.
%    e-mail : Paul.Dierckx@cs.kuleuven.ac.be
%
%  latest update : march 1987
%
%----------
%
%TvdZ. Last edits: November 2006


function [z,ier] = bispev(tx,ty,c,kx,ky,x,y);

ier = 10; %error flag

nx = size(tx,2); %Total number of knots in the x direction
ny = size(ty,2); %Total number of knots in the y direction
mx = size(x,2); %Total number of x grid points on which to evaluate spline
my = size(y,2); %Total number of y grid points on which to evaluate spline

lwest = (kx+1).*mx+(ky+1).*my;

%Allocating two workspaces wrk and iwrk

lwrk = mx*(kx+1)+my*(ky+1);
wrk = zeros(1,lwrk);
kwrk = mx+my;
iwrk = zeros(1,kwrk);

% Before starting computations a data check is made. If the input data
% are invalid control is immediately repassed to the calling program.

if (mx-1 < 0)
    return
elseif (mx-1 == 0)
else
    for i = 2:mx
        if (x(i) < x(i-1))
            return
        end
    end
end

if (my-1 < 0)
    return
elseif (my-1 == 0)
else
    for i = 2:my
        if (y(i) < y(i-1))
            return
        end
    end
end
```

```
ier = 0;
iw = mx.*(kx+1)+1;
z = fpbisp(tx,nx,ty,ny,c,kx,ky,x,mx,y,my,wrk(1),wrk(iw),iwrk(1),...
    iwrk(mx+1));

%-------------------------------------------------------------------------
%End of: bispev.m
%-------------------------------------------------------------------------
```

3.3. fpbisp.m

```
%-----------------------------------------------------------------------
%Start of: fpbisp.m
%-----------------------------------------------------------------------
%
%Translated from Fortran into Matlab using the f2matlab package.
%
%TvdZ. Last edits: November 2006

function z = fpbisp(tx,nx,ty,ny,c,kx,ky,x,mx,y,my,wx,wy,lx,ly);

h = zeros(1,6);

%x dimension

kx1 = kx+1;
nkx1 = nx-kx1;
tb = tx(kx1);
te = tx(nkx1+1);
l = kx1;
l1 = l+1;

for i = 1:mx
    arg = x(i);
    if (arg < tb)
        arg = tb;
    end
    if (arg > te)
        arg = te;
    end

    while (arg >= tx(l1)) && (l ~= nkx1)
        l = l1;
        l1 = l+1;
    end

    h = fpbspl(tx,nx,kx,arg,l,h);

    lx(i) = l-kx1;
    for j = 1:kx1
        wx(i,j) = h(j);
    end
end

%y dimension

ky1 = ky+1;
nky1 = ny-ky1;
tb = ty(ky1);
te = ty(nky1+1);
l = ky1;
l1 = l+1;

for i = 1:my
    arg = y(i);
    if (arg < tb)
        arg = tb;
        end
        if (arg > te)
            arg = te;
        end
        while (arg >= ty(l1)) && (l ~= nky1)
            l = l1;
            l1 = l+1;
        end

        h = fpbspl(ty,ny,ky,arg,l,h);

        ly(i) = l-ky1;
        for j = 1:ky1
            wy(i,j) = h(j);
        end
end

%meshing together

m = 0;
z = zeros(mx,my);
for i = 1:mx
    l = lx(i).*nky1;
    for i1 = 1:kx1
        h(i1) = wx(i,i1);
    end
    for j = 1:my
        l1 = l+ly(j);
        sp = 0.;
        for i1 = 1:kx1
            l2 = l1;
            for j1 = 1:ky1
                l2 = l2+1;
                sp = sp+c(l2).*h(i1).*wy(j,j1);
            end
            l1 = l1+nky1;
        end
        m = m+1;
        z(m) = sp;
    end
end

%---------------------------------------------------------------------------
%End of: fpbisp.m
%---------------------------------------------------------------------------
```

3.4. fpbspl.m

```
%-----------------------------------------------------------------------
%Start of: fpbspl.m
%-----------------------------------------------------------------------
%
%Translated from Fortran into Matlab using f2matlab
%
%Subroutine fpbspl evaluates the (k+1) non-zero b-splines of degree k at
%t(l) <= x < t(l+1) using the stable recurrence relation of De Boor and
%Cox.
%
%TvdZ. Last edits: November 2006

function h = fpbspl(t,n,k,x,l,h);

hh = zeros(1,5);

h(1) = 1;

for j = 1:k

    for i = 1:j
        hh(i) = h(i);
    end

    h(1) = 0.;

    for i = 1:j
        li = l+i;
        lj = li-j;
        f = hh(i)./(t(li)-t(lj));
        h(i) = h(i)+f.*(t(li)-x);
        h(i+1) = f.*(x-t(lj));
    end

end

%-----------------------------------------------------------------------
%End of: fpbspl.m
%-----------------------------------------------------------------------
```

3.5. DetermineBC.m

```
%-----------------------------------------------------------------------
%Start of: DetermineBC.m
%-----------------------------------------------------------------------
%
%This routine determines the location of the beam center using
%the LaB6 measurement by means of a weighted average of the pixel
%positions of the pixels in and just around the direct beam mark, using
%the corresponding intensities as weights.
%
%TvdZ, November 2006

function [mBC,nBC,rowprof,colprof] = DetermineBC(image)

%xest and yest, first estimates of the location of the beam center, are
%entered. These values have been determined by visual inspection using the
%ESRF visualization program FIT2D, and are entered in FIT2D coordinates.
%Furthermore, half the width and height of the box over which the
%computation will run are entered (deltax en deltay), which have also been
%determined by visual inspection. All values are then transformed into
%matrix indices m and n.

xest = 981;
yest = 1011;
deltax = 20;
deltay = 10;

mest = 2049-yest;
nest = xest;
mmin = mest-deltay;
mmax = mest+deltay;
nmin = nest-deltax;
nmax = nest+deltax;

%Compute the dark current intensities

DarkCurrent = double(zeros(2048,2048));
for nr = 26:47
    filename = ['E:\Data\Dark_Current\darkcurrent00' num2str(nr) '.edf'];
    AddMatrix = double(readfrelon2k(filename));
    DarkCurrent = DarkCurrent+AddMatrix;
end
DarkCurrent = double(DarkCurrent./22);

%The LaB6-file is read

A = readfrelon2k(image);
A = double(A);
A = A-DarkCurrent;

%Computation over the submatrix A(imin:imax,jmin:jmax)

mBC = 0;
nBC = 0;
rowprof = zeros(mmax-mmin+1,1);
```

```
colprof = zeros(nmax-nmin+1,1);
sumsum = sum(sum(A(mmin:mmax,nmin:nmax)));

for i = mmin:mmax
    for j = nmin:nmax
        weight = A(i,j);
        rowprof(i-mmin+1) = rowprof(i-mmin+1)+weight;
        colprof(j-nmin+1) = colprof)j-nmin+1)+weight;
        mBC = double(mBC+double(i*weight/sumsum));
        nBC = double(nBC+double(j*weight/sumsum));
    end
end
rowprof = rowprof./(nmax-nmin+1);
colprof = colprof./(mmax-mmin+1);
mBC = double(mBC);
nBC = double(nBC);

%-----------------------------------------------------------------------------
%End of: DetermineBC.m
%-----------------------------------------------------------------------------
```

3.6. readfrelon2k.m

```
%--------------------------------------------------------------------------
%Start of: readfrelon2k.m
%--------------------------------------------------------------------------
%
function [A] = readfrelon2k(filename)
% edf reader for the 2K Frelon camera
% function [A]= readfrelon2k(filename)


fid = fopen(filename,'r');

if (fid~=-1)
    fseek(fid,-(2*2048*2048),'eof');

    A = uint16((fread(fid,[2048,2048],'uint16'))');

    fclose(fid);
else
    ['edf reader: unable to open file: ',filename]
end

%--------------------------------------------------------------------------
%End of: readfrelon2k.m
%--------------------------------------------------------------------------
```

3.7. Analyze_4d.m

```
%---------------------------------------------------------------------------
%Start of: Analyze_4d.m
%---------------------------------------------------------------------------
%
%This is the main routine for reconstruction of the original austenite
%microstructure from the diffraction images of the 4d-dataset. This routine
%calls the various subroutines that perform the different steps required
%for the reconstruction. It is possible to exclude some of the code from
%being executed, e.g. when one has already performed a specific part of the
%analysis earlier. In this case, exclude the lines in question using '%'
%and include code reading in the result of the earlier analysis.
%
%TvdZ. Last edits: February 2007

function u = Analyze_4d;

%Create a list of all peaks of the 4d measurement using FindPeaks_4d, or,
%if this has already been done previously, access this file using the
%dlmread command.

PeakList = FindPeaks_4d;
%PeakList = dlmread('E:\4d_4e_Analysis\4d_Peak_List.txt','\t');

[nr_peaks width_peaklist] = size(PeakList);

%Compute the dark current intensities

DarkCurrent = double(zeros(2048,2048));
for nr = 26:47
    filename = ['E:\Data\Dark_Current\darkcurrent00' num2str(nr) '.edf'];
    AddMatrix = double(readfrelon2k(filename));
    DarkCurrent = DarkCurrent+AddMatrix;
end
DarkCurrent = double(DarkCurrent./22);

%Open file that contains the spatial distortion correction

SDMatrix = dlmread('E:\4d_4e_Analysis\SDMatrix.txt','\t');

%Determine if the estimates for the peak width in all three dimensions are
%correct, by comparing background intensities from the two enveloping
%shells.

Total_Peak_Intensities = [];

ind = 1;

for n = 1:nr_peaks

    stripe = PeakList(n,1);
    layer = PeakList(n,2);
    omega = PeakList(n,3);
    ringnumber = PeakList(n,4);
    row = PeakList(n,5);
```

```
        column = PeakList(n,6);
        R = PeakList(n,7);
        twotheta = PeakList(n,8);
        eta = PeakList(n,9);
        intensity = PeakList(n,10);
        deltarow = PeakList(n,11);
        deltacolumn = PeakList(n,12);
        deltaomega = PeakList(n,13);

        [Total_Intensity] = ShellCheck_4d(stripe,layer,omega,ringnumber,...
            row,column,deltarow,deltacolumn,deltaomega,DarkCurrent,SDMatrix);

        if isempty(Total_Intensity) == 0
            Total_Peak_Intensities(ind,:) = Total_Intensity;
            ind = ind+1;
        end

    end

    dlmwrite('E:\4d_4e_Analysis\4d_Total_Peak_Intensities.txt',...
        Total_Peak_Intensities,'-append','delimiter','\t');

    %If the peak analysis has been performed separately at an earlier stage,
    %use the line below to read in the results of that analysis.

    %Total_Peak_Intensities = dlmread('E:\4d_4e_Analysis\'...
    %    '4d_Total_Peak_Intensities.txt','\t');
    [length_TPI width_TPI] = size(Total_Peak_Intensities);

    [Arranged_Spots,Total_Grain_Intensities] = ArrangeSpots(...
        Total_Peak_Intensities,1,length_TPI);

    %---------------------------------------------------------------------------
    %End of: Analyze_4d.m
    %---------------------------------------------------------------------------
```

3.8. FindPeaks_4d.m

```
%-----------------------------------------------------------------------
%Start of: FindPeaks_4d.m
%-----------------------------------------------------------------------
%
%This function creates a list of all peaks in the 4d series. The output is
%in the form of an (nx19)-matrix, where n is the total number of peaks
%found. Each row contains peak information in the following manner:
%[Stripe Layer Omega RingNumber Row Column R TwoTheta Eta Intensity
%DeltaRow DeltaColumn DeltaOmega RowMinus RowPlus ColumnMinus ColumnPlus
%OmegaMinus OmegaPlus]. The output is created by subsequent calls of the
%subroutine AnalyzeLayer_4d.m.
%
%TvdZ. Last edits: February 2007.

function Peaks = FindPeaks_4d;

%Compute dark current image for electronic background correction

DarkCurrent = double(zeros(2048,2048));
for nr = 26:47
    filename = ['E:\Data\Dark_Current\darkcurrent00' num2str(nr) '.edf'];
    AddMatrix = double(readfrelon2k(filename));
    DarkCurrent = DarkCurrent+AddMatrix;
end
DarkCurrent = double(DarkCurrent./22);

%Start peaksearching

Peaks = [];

for stripe = 0:2

    for layer = 0:49

        u = AnalyzeLayer_4d(stripe,layer,DarkCurrent);
        Peaks = [Peaks;u];

    end

end

dlmwrite('E:\4d_4e_Analysis\4d_PeakList.txt',Peaks,'delimiter','\t');

%-----------------------------------------------------------------------
%End of: FindPeaks_4d.m
%-----------------------------------------------------------------------
```

3.9. AnalyzeLayer_4d.m

```
%----------------------------------------------------------------------------
%Start of: AnalyzeLayer_4d.m
%----------------------------------------------------------------------------
%
%This program analyzes a single layer from a 4d measurement, corresponding
%to 92 files with different omega orientations. The files are read,
%together with their previous and subsequent file in omega space. Masks are
%then applied so that the data of interest remains. Within these remaining
%images, searches are conducted for peak maxima in (x,y,omega)-space.
%
%TvdZ. Last edits: February 2007.

function u = AnalyzeLayer_4d(stripe,layer,DarkCurrent);

if stripe == 0 || stripe == 1 || stripe == 2
else
    error('Invalid stripe input. Valid input are: 0, 1, 2.')
end
if rem(layer,1) == 0 & layer >= 0 & layer <= 49
else
    error('Invalid layer input. Valid input is an integer between 0 and'...
        '49.')
end

stripestr = num2str(stripe);
layerstr = num2str(layer);

MinTwoTheta_A200 = 4.75; %degrees
MaxTwoTheta_A200 = 5.2;
MinTwoTheta_A220 = 6.8;
MaxTwoTheta_A220 = 7.3;

PixelSize = 47.4; %micrometer
Lsd = 241; %millimeter
Lsd_Pixels = Lsd*1000/PixelSize; %#pixels
BoxHeight = 100; %micrometer
BoxWidth = 15; %micrometer
RowBeamCenter = 1038; %row index
ColumnBeamCenter = 981; %column index

%Specification of detector tilt using definitions of CreateMask2DTilt.m.
%Tilt_Psi and Tilt_EtaT follow from the TILT algorithm in fit2d on the LaB6
%image before spatial correction.
%Note: when computing detector tilt on the corrected image, the value found
%is much smaller (less than 1 degree).

Tilt_Psi = 8.1; %degrees
Tilt_EtaT = 11.9; %degrees

k = 0;
u = [];

%Compute average dark current intensity and the threshold intensity
```

```
avg_dc = mean(mean(DarkCurrent));

IMin = 2*sqrt(avg_dc); %The minimum intensity that is considered as coming
                       %from a grain equals twice the square root of the
                       %average D.C. intensity; for a Poisson distribution,
                       %this equals twice the standard deviation.

%Compute the masks for the austenite and background rings, and determine
%the borders of the rectangles that enclose these masks.

Mask_A200 = CreateMask2DTilt(Lsd,PixelSize,Tilt_Psi,Tilt_EtaT,...
            ColumnBeamCenter,RowBeamCenter, MinTwoTheta_A200,...
            MaxTwoTheta_A200);
Mask_A220 = CreateMask2DTilt(Lsd,PixelSize,Tilt_Psi,Tilt_EtaT,...
            ColumnBeamCenter,RowBeamCenter, MinTwoTheta_A220,...
            MaxTwoTheta_A220);

m_A200_min = 2;
m_A200_max = 2047;
n_A200_min = 2;
n_A200_max = 2047;
A200_m = sum(Mask_A200,2);
A200_n = sum(Mask_A200,1);
while A200_m(m_A200_min) == 0
    m_A200_min = m_A200_min+1;
end
while A200_m(m_A200_max) == 0;
    m_A200_max = m_A200_max-1;
end
while A200_n(n_A200_min) == 0
    n_A200_min = n_A200_min+1;
end
while A200_n(n_A200_max) == 0;
    n_A200_max = n_A200_max-1;
end

m_A220_min = 2;
m_A220_max = 2047;
n_A220_min = 2;
n_A220_max = 2047;
A220_m = sum(Mask_A220,2);
A220_n = sum(Mask_A220,1);
while A220_m(m_A220_min) == 0
    m_A220_min = m_A220_min+1;
end
while A220_m(m_A220_max) == 0;
    m_A220_max = m_A220_max-1;
end
while A220_n(n_A220_min) == 0
    n_A220_min = n_A220_min+1;
end
while A220_n(n_A220_max) == 0;
    n_A220_max = n_A220_max-1;
end

%Read the file to be examined, and create the masked images.

for omega = -29:60

    omegastr = num2str(omega);
    if omega == -29

        filenr = stripe*4600+layer*92+(omega+30);

        if filenr < 10
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d000'...
                num2str(filenr) '.edf'];
        elseif filenr < 100
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00'...
                num2str(filenr) '.edf'];
        elseif filenr < 1000
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d0'...
                num2str(filenr) '.edf'];
        else
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d'...
                num2str(filenr) '.edf'];
        end
        ImageRaw = double(readfrelon2k(filename));
        ImageRaw = ImageRaw-DarkCurrent;
        Image_A200 = ImageRaw.*Mask_A200;
        Image_A220 = ImageRaw.*Mask_A220;

        filenr = filenr-1;

        if filenr < 10
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d000'...
                num2str(filenr) '.edf'];
        elseif filenr < 100
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00'...
                num2str(filenr) '.edf'];
        elseif filenr < 1000
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d0'...
                num2str(filenr) '.edf'];
        else
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d'...
                num2str(filenr) '.edf'];
        end
        ImageRaw = double(readfrelon2k(filename));
        ImageRaw = ImageRaw-DarkCurrent;
        Image_A200_prev = ImageRaw.*Mask_A200;
        Image_A220_prev = ImageRaw.*Mask_A220;

        filenr = filenr+2;

        if filenr < 10
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d000'...
                num2str(filenr) '.edf'];
        elseif filenr < 100
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00'...
                num2str(filenr) '.edf'];
        elseif filenr < 1000
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d0'...
                num2str(filenr) '.edf'];
        else
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d'...
                num2str(filenr) '.edf'];
        end
        ImageRaw = double(readfrelon2k(filename));
        ImageRaw = ImageRaw-DarkCurrent;
        Image_A200_next = ImageRaw.*Mask_A200;
        Image_A220_next = ImageRaw.*Mask_A220;
```

```
elseif stripe == 0 && layer == 0 && ...
        (omega == 7 || omega == 8 || omega == 9)

    %File 0038, corresponding to stripe = 0, layer = 0, omega = 8 is
    %missing. Hence this file and the two surrounding it in omega space
    %cannot be analyzed properly

    continue

elseif stripe == 0 && layer == 0 && omega == 10

    filenr = stripe*4600+layer*92+(omega+30);

    filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00' num2str(filenr)...
        '.edf'];
    ImageRaw = double(readfrelon2k(filename));
    ImageRaw = ImageRaw-DarkCurrent;
    Image_A200 = ImageRaw.*Mask_A200;
    Image_A220 = ImageRaw.*Mask_A220;

    filenr = filenr-1;

    filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00' num2str(filenr)...
        '.edf'];
    ImageRaw = double(readfrelon2k(filename));
    ImageRaw = ImageRaw-DarkCurrent;
    Image_A200_prev = ImageRaw.*Mask_A200;
    Image_A220_prev = ImageRaw.*Mask_A220;

    filenr = filenr+2;

    filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00' num2str(filenr)...
        '.edf'];
    ImageRaw = double(readfrelon2k(filename));
    ImageRaw = ImageRaw-DarkCurrent;
    Image_A200_next = ImageRaw.*Mask_A200;
    Image_A220_next = ImageRaw.*Mask_A220;

elseif stripe == 0 && layer == 34 && omega == -18

    %File 3141, corresponding to stripe 0, layer 34, omega -17 has
    %dimensions 2047x2048 (probably due to a glitch during data
    %collection); therefore the first row of ImageRaw is discarded,
    %since this row contains nonsense (numerical representations of
    %ASCII characters), after which an extra row of pixels with
    %intensity 1000 is added to the bottom of the matrix (visual
    %inspection of the location of the direct beam mark in this picture
    %suggested that the missing row was probably the bottom one)

    Image_A200_prev = Image_A200;
    Image_A220_prev = Image_A220;
    Image_A200 = Image_A200_next;
    Image_A220 = Image_A220_next;
    filenr = filenr+1;
    if filenr < 10
        filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d000'...
            num2str(filenr) '.edf'];
    elseif filenr < 100
        filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00'...
            num2str(filenr) '.edf'];
        elseif filenr < 1000
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d0'...
                num2str(filenr) '.edf'];
        else
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d'...
                num2str(filenr) '.edf'];
        end
        ImageRaw = double(readfrelon2k(filename));
        ImageRaw = ImageRaw(2:2048,:);
        Addition = double(1000*ones(1,2048));
        ImageRaw = [ImageRaw;Addition];
        ImageRaw = ImageRaw-DarkCurrent;
        Image_A200_next = ImageRaw.*Mask_A200;
        Image_A220_next = ImageRaw.*Mask_A220;

    else

        Image_A200_prev = Image_A200;
        Image_A220_prev = Image_A220;
        Image_A200 = Image_A200_next;
        Image_A220 = Image_A220_next;
        filenr = filenr+1;
        if filenr < 10
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d000'...
                num2str(filenr) '.edf'];
        elseif filenr < 100
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00'...
                num2str(filenr) '.edf'];
        elseif filenr < 1000
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d0'...
                num2str(filenr) '.edf'];
        else
            filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d'...
                num2str(filenr) '.edf'];
        end
        ImageRaw = double(readfrelon2k(filename));
        ImageRaw = ImageRaw-DarkCurrent;
        Image_A200_next = ImageRaw.*Mask_A200;
        Image_A220_next = ImageRaw.*Mask_A220;

    end

    for ring = 1:2

        switch ring
            case 1
                hkl = 'A200';
                Image = Image_A200;
                Image_prev = Image_A200_prev;
                Image_next = Image_A200_next;
                m_min = m_A200_min;
                m_max = m_A200_max;
                n_min = n_A200_min;
                n_max = n_A200_max;
            case 2
                hkl = 'A220';
                Image = Image_A220;
                Image_prev = Image_A220_prev;
                Image_next = Image_A220_next;
                m_min = m_A220_min;
                m_max = m_A220_max;
            n_min = n_A220_min;
            n_max = n_A220_max;
    end

    %Find peaks in (x,y,omega)-space

    for m = m_min:m_max
        for n = n_min:n_max
            if Image(m,n) > IMin
                if isequal(Image(m,n),max(max(Image(m-1:m+1,n-1:n+1))))
                    if Image(m,n) >= max(max(Image_prev(m-1:m+1,...
                            n-1:n+1)))
                        if Image(m,n) >= max(max(Image_next(m-1:m+1,...
                                n-1:n+1)))
                            k = k+1;
                            StripePeak(k,1) = stripe;
                            LayerPeak(k,1) = layer;
                            OmegaPeak(k,1) = omega;
                            RingNumberPeak(k,1) = ring;
                            RowPeak(k,1) = m;
                            ColumnPeak(k,1) = n;
                            RPeak(k,1) = sqrt((m-RowBeamCenter)^2+...
                                (n-ColumnBeamCenter)^2);
                            TwoThetaPeak(k,1) = (180/pi)*...
                                atan2(RPeak(k,1),Lsd_Pixels);
                            EtaPeak(k,1) = (180/pi)*atan2(n-...
                                ColumnBeamCenter,RowBeamCenter-m);
                            %In this way, both arguments to atan2 are
                            %positive when the spot is in the first
                            %quadrant with respect to the beam centre.
                            if EtaPeak(k,1) < 0
                                EtaPeak(k,1) = EtaPeak(k)+360;
                            end
                            IntensityPeak(k,1) = Image(m,n);
                            %Calculate peak width in row direction
                            %(half width half maximum)
                            inc = 1;
                            while Image(m-inc,n) > IntensityPeak(k,1)/2
                                inc = inc+1;
                            end
                            RowMinus(k,1) = RowPeak(k,1)-inc;
                            inc = 1;
                            while Image(m+inc,n) > IntensityPeak(k,1)/2
                                inc = inc+1;
                            end
                            RowPlus(k,1) = RowPeak(k,1)+inc;
                            DeltaRow(k,1) = ceil((RowPlus(k,1)-...
                                RowMinus(k,1))/2);
                            %Calculate peak width in colummn direction
                            %(half width half maximum)
                            inc = 1;
                            while Image(m,n-inc) > IntensityPeak(k,1)/2
                                inc = inc+1;
                            end
                            ColumnMinus(k,1) = ColumnPeak(k,1)-inc;
                            inc = 1;
                            while Image(m,n+inc) > IntensityPeak(k,1)/2
                                inc = inc+1;
                            end
                            ColumnPlus(k,1) = ColumnPeak(k,1)+inc;
                            DeltaColumn(k,1) = ceil(...
                                (ColumnPlus(k,1)-ColumnMinus(k,1))/2);
                            %Calculate width in omega direction. It is
                            %assumed this is never more than 2 in a
                            %single direction.
                            inc = 1;
                            if Image_prev(m,n) > IntensityPeak(k,1)/2
                                inc = 2;
                            end
                            OmegaMinus(k,1) = OmegaPeak(k,1)-inc;
                            inc = 1;
                            if Image_next(m,n) > IntensityPeak(k,1)/2
                                inc = 2;
                            end
                            OmegaPlus(k,1) = OmegaPeak(k,1)+inc;
                            DeltaOmega(k,1) = (OmegaPlus(k,1)-...
                                OmegaMinus(k,1))/2;
                            %Account for DeltaOmega-dependence on Eta.
                            DeltaOmega(k,1) = ceil(DeltaOmega(k,1)*...
                                abs(sin(EtaPeak(k,1)*pi/180)));
                            if DeltaOmega(k,1) == 0
                                %If Eta = 0, change from 0 to 1.
                                DeltaOmega(k,1) = 1;
                                OmegaMinus(k,1) = OmegaPeak(k,1)-1;
                                OmegaPlus(k,1) = OmegaPeak(k,1)+1;
                            end
                        end
                    end
                end
            end
        end
    end
    if k == 0
        v = [];
    else
        v = [StripePeak LayerPeak OmegaPeak RingNumberPeak RowPeak...
            ColumnPeak RPeak TwoThetaPeak EtaPeak IntensityPeak...
            DeltaRow DeltaColumn DeltaOmega RowMinus RowPlus...
            ColumnMinus ColumnPlus OmegaMinus OmegaPlus];
    end
    u = [u;v];

    k = 0;
    StripePeak = [];
    LayerPeak = [];
    OmegaPeak = [];
    RingNumberPeak = [];
    RowPeak = [];
    ColumnPeak = [];
    RPeak = [];
    TwoThetaPeak = [];
    EtaPeak = [];
    IntensityPeak = [];
    DeltaRow = [];
    DeltaColumn = [];
    DeltaOmega = [];
    RowMinus = [];
    RowPlus = [];
    ColumnMinus = [];
    ColumnPlus = [];
    OmegaMinus = [];
    OmegaPlus = [];
    end
end

u;

%-----------------------------------------------------------------------------
%End of: AnalyzeLayer_4d.m
%-----------------------------------------------------------------------------
```

3.10. CreateMask2DTilt.m

```
%-----------------------------------------------------------------------
%Start of: CreateMask2DTilt.m
%-----------------------------------------------------------------------
%
%function CreateMask2DTilt.m********************************************
%An ellipsoidal mask is created to the 2D diffraction patterns
%to remove the data below a scattering angle SAmin and
%above a scattering angle SAmax. The beam center is located
%at (CenterX,CenterY). The sample to detector distance Lsd and
%the pixel size are used. Psi is the tilt of the detector and EtaT
%describes the direction of the rotation axis in the XY-plane for
%tilt of the 2D detector. The rotation axis is oriented along
%(X-XC,Y-YC) = (sin(EtaT),cos(EtaT)). The direction of the longest
%axis of the ellipse from the beam center is perpendicular to
%this. Note that the center of the ellipse is not equal to the beam
%center; the latter is located in one of the focal points of the ellipse.
%From the rotation angles PsiX and PsiY the total rotation angle
%amounts to: Psi = (PsiX^2+PsiY^2)^(1/2). The rotation angle amounts
%to EtaT = atan2(PsiX,PsiY).
%Lsd is required in millimeters, pixel in micrometers.
%
%Niels van Dijk, June 2006.

function u = CreateMask2DTilt(Lsd,pixel,Psi,EtaT,CX,CY,SAmin,SAmax)

%Begin *****************************************************************

%Initialisation ********************************************************

Eta0 = EtaT+sign(Psi)*90.; % direction for the long axis

PsiRad  = (pi/180.)*Psi;
Eta0Rad = (pi/180.)*EtaT;
SAminRad = (pi/180.)*SAmin;
SAmaxRad = (pi/180.)*SAmax;
Lsd = Lsd*1000; %transform Lsd from milli- to micrometers

FactorMax = 1.-(tan(SAmaxRad)*tan(PsiRad))^2;
FactorMin = 1.-(tan(SAminRad)*tan(PsiRad))^2;

R0Min = (Lsd/pixel)*abs(tan(SAminRad));
R0Max = (Lsd/pixel)*abs(tan(SAmaxRad));

RlMin = R0Min/abs(cos(PsiRad)*FactorMin); %long axis ellipse for 2ThetaMin
RlMax = R0Max/abs(cos(PsiRad)*FactorMax); %long axis ellipse for 2ThetaMax

RsMin = R0Min/(FactorMin)^(1/2);          %short axis ellipse for 2ThetaMin
RsMax = R0Max/(FactorMax)^(1/2);          %short axis ellipse for 2ThetaMax

dRMin = RlMin*abs(tan(SAminRad)*tan(PsiRad));
dRMax = RlMax*abs(tan(SAmaxRad)*tan(PsiRad));

CEXMin = CX+dRMin*sin(Eta0Rad);        %X value center ellipse for 2ThetaMin
CEYMin = CY+dRMin*cos(Eta0Rad);        %Y value center ellipse for 2ThetaMin
```

```
CEXMax = CX+dRMax*sin(Eta0Rad);          %X value center ellipse for 2ThetaMax
CEYMax = CY+dRMax*cos(Eta0Rad);          %Y value center ellipse for 2ThetaMax

%Masking ******************************************************************

Mask = zeros(2048,2048);

[rows cols] = size(Mask);
for m = 1:rows
    dyMax = m-CEYMax;
    dyMin = m-CEYMin;
    for n = 1:cols
        dxMax = n-CEXMax;
        dxMin = n-CEXMin;
        duMax = (cos(Eta0Rad)*dxMax-sin(Eta0Rad)*dyMax)/RsMax;
        duMin = (cos(Eta0Rad)*dxMin-sin(Eta0Rad)*dyMin)/RsMin;
        dvMax = (sin(Eta0Rad)*dxMax+cos(Eta0Rad)*dyMax)/RlMax;
        dvMin = (sin(Eta0Rad)*dxMin+cos(Eta0Rad)*dyMin)/RlMin;
        if (duMin)^2+(dvMin)^2 >= 1. && (duMax)^2+(dvMax)^2 <= 1.
            Mask(m,n) = 1;
        else
            Mask(m,n) = 0;
        end
    end
end

u = Mask;

%End **********************************************************************

%--------------------------------------------------------------------------
%End of: CreateMask2DTilt.m
%--------------------------------------------------------------------------
```

3.11. ShellCheck_4d.m

```
%-----------------------------------------------------------------------
%Start of: ShellCheck_4d.m
%-----------------------------------------------------------------------
%
%ShellCheck_4d.m determines if the size of the three-dimensional box
%surrounding a peak as estimated using the half-width-half-maximum values
%is sufficient to envelop all the intensity belonging to the peak in
%question. If not, the size is adjusted till it suffices. Average peak
%characteristics are computed, and spatial distortion correction is
%applied.
%
%TvdZ. Last edits: September 2006.

function [Total_Intensity] = ShellCheck_4d(stripe,layer,omega,...
    ringnumber,row,column,deltarow,deltacolumn,deltaomega,DarkCurrent,...
    SDMatrix);

RowBC = 1038; %Distorted beam center position - row
ColumnBC = 981; %Distorted beam center position - column
PixelSize = 47.4; %micrometer
Lsd = 241; %millimeter
Lsd_Pixels = Lsd*1000/PixelSize; %#pixels

%Retrieve the corrected beam center position from SDMatrix, which contains
%the spatial distortion corrections for the pixels in the ROI. The location
%of this position within matrix SDMatrix depends on m_min, m_max, n_min and
%n_max - the borders of the outermost (i.e. A220) mask - in the following
%manner:
%
% index = (rowBC-m_min)*(n_max-n_min+1)+(columnBC-n_min+1)
%
%The values of m_min, m_max, n_min and n_max can be retrieved from SDMatrix

[lengthSD widthSD] = size(SDMatrix);
m_min = SDMatrix(1,1);
m_max = SDMatrix(lengthSD,1);
n_min = SDMatrix(1,2);
n_max = SDMatrix(lengthSD,2);

RowBC_corr = SDMatrix(((RowBC-m_min)*(n_max-n_min+1)+...
    (ColumnBC-n_min+1)),5);
ColumnBC_corr = SDMatrix(((RowBC-m_min)*(n_max-n_min+1)+...
    (ColumnBC-n_min+1)),6);

%Compute the correct peak box size

%Define two parameters which are used to directly indicate whether or not
%the correct box size is reached, and a parameter indicating an error
%occurence

box_too_large = 1;
box_too_small = 1;
error_indicator = 0;
```

```
     file38_flag = 0; %See line 106

     Total_Intensity = [];

     while (box_too_large ~= 0 || box_too_small ~= 0) && (error_indicator == 0)
60
         Box0 = [];
         Shell1 = [];
         Shell2 = [];
         Box0def = [];
         Shell1def = [];
         Shell2def = [];

         %Obtain a list of pixel positions around the selected peak for the
         %current peak box and the two validation shells.
70       %deltarow, deltacolumn and deltaomega should represent the position of
         %the first surrounding shell. E.g., if the peak maximum is located at
         %(1000,1000) and deltarow = 5, this means pixel (995,1000) belongs to
         %shell1 but (996,1000) belongs to the peak box.

         rowmin2 = row-deltarow-1;
         rowmax2 = row+deltarow+1;
         columnmin2 = column-deltacolumn-1;
         columnmax2 = column+deltacolumn+1;
         rowmin1 = row-deltarow;
80       rowmax1 = row+deltarow;
         columnmin1 = column-deltacolumn;
         columnmax1 = column+deltacolumn;
         rowmin0 = row-deltarow+1;
         rowmax0 = row+deltarow-1;
         columnmin0 = column-deltacolumn+1;
         columnmax0 = column+deltacolumn-1;

         omegamin2 = omega-deltaomega-1;
         omegamax2 = omega+deltaomega+1;
90       omegamin1 = omega-deltaomega;
         omegamax1 = omega+deltaomega;
         omegamin0 = omega-deltaomega+1;
         omegamax0 = omega+deltaomega-1;

         if omegamin2 < -30 || omegamax2 > 61
             message = ['The peak at stripe ' num2str(stripe) ', layer '...
                        num2str(layer) ', omega ' num2str(omega) ', row '...
                        num2str(row) ', column ' num2str(column) ' cannot '...
                        'be used because the peak exceeds the limit of the'...
100                     ' omega scan'];
             %disp(message)
             error_indicator = 1;
             break
         end

         if (stripe == 0) && (layer == 0) && (omegamin2 <= 8) &&...
                 (omegamax2 >= 8)
             %File 38, corresponding to stripe = 0, layer = 0, omega = 8, is
             %missing
110          if deltaomega > 1
                 deltaomega = deltaomega-1;
                 file38_flag = 1;
                 continue
             else
                 %No correct box size can be defined, because the algorithm
```

```
            %oscillates between either increasing or decreasing the box
            message = ['No correct box size can be defined for the peak'...
                ' at stripe ' num2str(stripe) ', layer ' num2str(layer)...
                ', omega ' num2str(omega) ', row ' num2str(row)...
                ', column ' num2str(column)];
            %disp(message)
            error_indicator = 1;
            break
        end
    end

    j = 0;
    k = 0;
    r = 0;

    for omeganew = omegamin2:omegamax2

        filenr = stripe*4600+layer*92+(omeganew+30);

        if filenr < 10
                filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d000'...
                    num2str(filenr) '.edf'];
            elseif filenr < 100
                filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d00'...
                    num2str(filenr) '.edf'];
            elseif filenr < 1000
                filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d0'...
                    num2str(filenr) '.edf'];
            else
                filename = ['E:\Data\Fe1C3MnA_4d\Fe1C3MnA_4d'...
                    num2str(filenr) '.edf'];
        end
        Image = double(readfrelon2k(filename));
        %From the localizing of the peaks, earlier, it is known that file
        %3141 has dimensions 2047x2048 instead of 2048x2048; in this case,
        %an extra row of background intensity is added.
        if size(Image) == [2047 2048]
            Addition = double(1000*ones(1,2048));
            Image = [Image;Addition];
        elseif size(Image) == [2048 2048]
        else
            error('Error: unexpected dimensions for diffraction image')
        end
        Image_corr = Image-DarkCurrent;

        for m = rowmin2:rowmax2

            for n = columnmin2:columnmax2

                PixelPos = sqrt((n-ColumnBC)^2+(m-RowBC)^2);
                PixelPos = round(PixelPos);
                if omeganew == omegamin2 || omeganew == omegamax2
                    k = k+1;
                    Shell2(k,1) = PixelPos;
                    Shell2(k,2) = Image_corr(m,n);
                    Shell2(k,3) = Image(m,n);
                elseif omeganew == omegamin1 || omeganew == omegamax1
                    if m == rowmin2 || m == rowmax2 ||...
                            n == columnmin2 || n == columnmax2
                        k = k+1;
                        Shell2(k,1) = PixelPos;
                    Shell2(k,2) = Image_corr(m,n);
                    Shell2(k,3) = Image(m,n);
                else
                    j = j+1;
                    Shell1(j,1) = PixelPos;
                    Shell1(j,2) = Image_corr(m,n);
                    Shell1(j,3) = Image(m,n);
                end
            else
                if m == rowmin2 || m == rowmax2 ||...
                        n == columnmin2 || n == columnmax2
                    k = k+1;
                    Shell2(k,1) = PixelPos;
                    Shell2(k,2) = Image_corr(m,n);
                    Shell2(k,3) = Image(m,n);
                elseif m == rowmin1 || m == rowmax1 ||...
                        n == columnmin1 || n == columnmax1
                    j = j+1;
                    Shell1(j,1) = PixelPos;
                    Shell1(j,2) = Image_corr(m,n);
                    Shell1(j,3) = Image(m,n);
                else
                    r = r+1;
                    Box0(r,1) = PixelPos;
                    Box0(r,2) = Image_corr(m,n);
                    Box0(r,3) = Image(m,n);
                    %Also save the row and column coordinates and the
                    %omega value, to later determine the center of
                    %gravity of the peak
                    Box0(r,4) = m;
                    Box0(r,5) = n;
                    Box0(r,6) = omeganew;
                end
            end
        end
    end
end

%For the peak box, Shell1 and Shell2, average the intensities of pixels
%with the same radius from the beam center.

Shell1min = min(Shell1(:,1));
Shell1max = max(Shell1(:,1));
Dif1 = Shell1max-Shell1min;

for i = 0:Dif1
    PixelPosRef = Shell1min+i;
    SumInt1 = 0;
    Count1 = 0;
    SumOrigInt1 = 0;
    for p = 1:j
        PixelPos = Shell1(p,1);
        if PixelPos == PixelPosRef
            Count1 = Count1+1;
            SumInt1 = SumInt1+Shell1(p,2);
            SumOrigInt1 = SumOrigInt1+Shell1(p,3);
        end
    end
    Shell1def(i+1,1) = PixelPosRef;
    Shell1def(i+1,2) = SumInt1;
    Shell1def(i+1,3) = Count1;
```

```
        Shell1def(i+1,4) = SumInt1/Count1;
        Shell1def(i+1,5) = SumOrigInt1/Count1;
    end

    Shell2min = min(Shell2(:,1));
    Shell2max = max(Shell2(:,1));
    Dif2 = Shell2max-Shell2min;

    for i = 0:Dif2
        PixelPosRef = Shell2min+i;
        SumInt2 = 0;
        Count2 = 0;
        SumOrigInt2 = 0;
        for p = 1:k
            PixelPos = Shell2(p,1);
            if PixelPos == PixelPosRef
                Count2 = Count2+1;
                SumInt2 = SumInt2+Shell2(p,2);
                SumOrigInt2 = SumOrigInt2+Shell2(p,3);
            end
        end
        Shell2def(i+1,1) = PixelPosRef;
        Shell2def(i+1,2) = SumInt2;
        Shell2def(i+1,3) = Count2;
        Shell2def(i+1,4) = SumInt2/Count2;
        Shell2def(i+1,5) = SumOrigInt2/Count2;
    end

    Box0min = min(Box0(:,1));
    Box0max = max(Box0(:,1));
    Dif0 = Box0max-Box0min;

    for i = 0:Dif0
        PixelPosRef = Box0min+i;
        SumInt0 = 0;
        Count0 = 0;
        SumOrigInt0 = 0;
        for p = 1:r
            PixelPos = Box0(p,1);
            if PixelPos == PixelPosRef
                Count0 = Count0+1;
                SumInt0 = SumInt0+Box0(p,2);
                SumOrigInt0 = SumOrigInt0+Box0(p,3);
            end
        end
        Box0def(i+1,1) = PixelPosRef;
        Box0def(i+1,2) = SumInt0;
        Box0def(i+1,3) = Count0;
        Box0def(i+1,4) = SumInt0/Count0;
        Box0def(i+1,5) = SumOrigInt0/Count0;
    end

    %For each entry in Shell1, check if the corresponding average intensity
    %is approximately equal to the average intensity in Shell2
    %corresponding to the same distance from the beam center.

    [length_shell1def width_shell1def] = size(Shell1def);

    j = 1;

    for i = 1:length_shell1def
```

```
        PixelPos1 = Shell1def(i,1);
        PixelPos2 = Shell2def(j,1);
        while PixelPos1 ~= PixelPos2
            j = j+1;
            PixelPos2 = Shell2def(j,1);
        end
        I1 = Shell1def(i,4);
        I1_orig = Shell1def(i,5);
        I2 = Shell2def(j,4);
        I2_orig = Shell2def(j,5);
        Count1 = Shell1def(i,3);
        Count2 = Shell2def(j,3);

        criterion = sqrt(I1_orig)/sqrt(Count1)+sqrt(I2_orig)/sqrt(Count2);
            %This is the criterion that determines whether or not the
            %average intensities in Shell1 and Shell2 differ significantly;
            %this value corresponds to two standard deviations

        if I1-I2 > criterion %Shell1 contains intensity from the peak in
                             %Box0
            deltarow = deltarow+1;
            deltacolumn = deltacolumn+1;
            if file38_flag == 0
                deltaomega = deltaomega+1;
                if deltaomega > 2
                    deltaomega = 2; %deltaomega > 2 is highly unlikely
                end
            end
            box_too_small = box_too_small+1;
            if box_too_small > 20
                %No correct box size can be defined, because the algorithm
                %keeps increasing the box size 'indefinitely'
                message = ['No correct box size can be defined for the '...
                    'peak at stripe ' num2str(stripe) ', layer '...
                    num2str(layer) ', omega ' num2str(omega) ', row '...
                    num2str(row) ', column ' num2str(column)];
                %disp(message)
                error_indicator = 1;
            end
            break %The for-loop is ended
        elseif I2-I1 > criterion %Shell2 contains intensity from a
                                 %neighbouring peak
            deltarow = deltarow-1;
            deltacolumn = deltacolumn-1;
            deltaomega = deltaomega-1;
            if deltarow < 1 && deltacolumn < 1 && deltaomega < 1
                message = ['The peak at stripe ' num2str(stripe)...
                    ', layer ' num2str(layer) ', omega ' num2str(omega)...
                    ', row ' num2str(row) ', column ' num2str(column)...
                    ' cannot be used due to overlap from a neighbouring'...
                    ' peak'];
                %disp(message)
                error_indicator = 1;
                break
            else
                if deltarow < 1
                    deltarow = 1;
                end
                if deltacolumn < 1
                    deltacolumn = 1;
                end
                if deltaomega < 1
                    deltaomega = 1;
                end
            end
            box_too_large = box_too_large+1;
            break %The for-loop is ended
        else %For this specific distance to beam center R, Box0 is defined
             %correctly
            if i == length_shell1def
                box_too_large = 0;
                box_too_small = 0;
            end
        end
    end

    if box_too_large > 1 && box_too_small > 1
        %No correct box size can be defined, because the algorithm
        %oscillates between either increasing or decreasing the box
        message = ['No correct box size can be defined for the peak at '...
            'stripe ' num2str(stripe) ', layer ' num2str(layer)...
            ', omega ' num2str(omega) ', row ' num2str(row) ', column '...
            num2str(column)];
        %disp(message)
        error_indicator = 1;
    end

end

if error_indicator == 1
    Total_Intensity = [];
    return
end

%Now that the box is defined correctly, compute total peak intensity

rowsize_peak = 2*deltarow-1;
columnsize_peak = 2*deltacolumn-1;
omegasize_peak = 2*deltaomega-1;

[length_box0def width_box0def] = size(Box0def);
s = 1;
Total_Peak_Intensity = 0;
N0 = sum(Box0def(:,3));

for r = 1:length_box0def
    Pos_Box0def = Box0def(r,1);
    Pos_Shell1def = Shell1def(s,1);
    while Pos_Box0def ~= Pos_Shell1def
        s = s+1;
        Pos_Shell1def = Shell1def(s,1);
    end
    Total_Peak_Intensity = Total_Peak_Intensity+(Box0def(r,2)-...
        Box0def(r,3).*Shell1def(s,4));
end
Total_Peak_Intensity;

%Compute the center of gravity of the peak

%First, sort Box0 on the basis of distance from beam center R
```

```
Box0_Sorted = sortrows(Box0,1);

%Then, subtract the non-electronic background from each pixel intensity

[length_box0 width_box0] = size(Box0);
s = 1;

for r = 1:length_box0
    Pos_Box0 = Box0_Sorted(r,1);
    Pos_Shell1def = Shell1def(s,1);
    while Pos_Box0 ~= Pos_Shell1def
        s = s+1;
        Pos_Shell1def = Shell1def(s,1);
    end
    Box0_Sorted(r,2) = Box0_Sorted(r,2)-Shell1def(s,4);
end

%Now weigh m, n and omega for each entry in Box0_Sorted using the
%intensities as weights, resulting in the position of the center of gravity
%of the peak in terms of m, n and omega

Row_CoG = 0;
Column_CoG = 0;
Omega_CoG = 0;
Peak_Intensity = sum(Box0_Sorted(:,2)); %By definition equal to
                                        %Total_Peak_Intensity
for r = 1:length_box0
    Row_CoG = Row_CoG+Box0_Sorted(r,2).*Box0_Sorted(r,4);
    Column_CoG = Column_CoG+Box0_Sorted(r,2).*Box0_Sorted(r,5);
    Omega_CoG = Omega_CoG+Box0_Sorted(r,2).*Box0_Sorted(r,6);
end
Row_CoG = Row_CoG./Peak_Intensity;
Column_CoG = Column_CoG./Peak_Intensity;
Omega_CoG = Omega_CoG./Peak_Intensity;

%The Center of Gravity of the peak has now been computed without any
%correction for the spatial distortion having been applied. This correction
%is now applied to the CoG coordinates. Non-integer coordinates are treated
%by applying a linear interpolation to the coordinate between the two
%neighbouring integers.
%Note: a test of the impact of applying the spatial distortion correction
%after the computation of the Center of Gravity instead of before (which
%would strictly be preferable) showed that this impact is negligible (less
%than 0.05 pixel).

Row_floor = floor(Row_CoG);
Row_ceil = ceil(Row_CoG);
Column_floor = floor(Column_CoG);
Column_ceil = ceil(Column_CoG);

rff = SDMatrix(((Row_floor-m_min)*(n_max-n_min+1)+...
    (Column_floor-n_min+1)),5);
rcf = SDMatrix(((Row_ceil-m_min)*(n_max-n_min+1)+...
    (Column_floor-n_min+1)),5);
rfc = SDMatrix(((Row_floor-m_min)*(n_max-n_min+1)+...
    (Column_ceil-n_min+1)),5);
rcc = SDMatrix(((Row_ceil-m_min)*(n_max-n_min+1)+...
    (Column_ceil-n_min+1)),5);
cff = SDMatrix(((Row_floor-m_min)*(n_max-n_min+1)+...
    (Column_floor-n_min+1)),6);
ccf = SDMatrix(((Row_ceil-m_min)*(n_max-n_min+1)+...
    (Column_floor-n_min+1)),6);
cfc = SDMatrix(((Row_floor-m_min)*(n_max-n_min+1)+...
    (Column_ceil-n_min+1)),6);
ccc = SDMatrix(((Row_ceil-m_min)*(n_max-n_min+1)+...
    (Column_ceil-n_min+1)),6);

Row_CoG_corr = (rcf-rff)*(Row_CoG-Row_floor)+(rfc-rff)*...
    (Column_CoG-Column_floor)+(rcc+rff-rcf-rfc)*(Row_CoG-Row_floor)*...
    (Column_CoG-Column_floor)+rff;
Column_CoG_corr = (ccf-cff)*(Row_CoG-Row_floor)+(cfc-cff)*...
    (Column_CoG-Column_floor)+(ccc+cff-ccf-cfc)*(Row_CoG-Row_floor)*...
    (Column_CoG-Column_floor)+cff;

R_CoG = sqrt((Column_CoG_corr-ColumnBC_corr)^2+...
    (Row_CoG_corr-RowBC_corr)^2);
TwoTheta_CoG = (180/pi)*atan2(R_CoG,Lsd_Pixels);
Eta_CoG = (180/pi)*atan2(Column_CoG_corr-ColumnBC_corr,RowBC_corr-...
    Row_CoG_corr);
if Eta_CoG < 0
    Eta_CoG = Eta_CoG+360;
end


%Output

Row_CoG_corr; %Row position of peak, center-of-gravity basis
Column_CoG_corr; %Column position of peak, center-of-gravity basis
Omega_CoG; %Omega position of peak, center-of-gravity basis
Total_Peak_Intensity; %Total corrected intensity from peak
N0; %Total number of pixels attributed to the peak

b0 = Box0def; %Positions and average intensities of pixels in Box0
b1 = Shell1def; %Positions and average intensities of pixels in Shell1
b2 = Shell2def; %Positions and average intensities of pixels in Shell2

Total_Intensity = [stripe layer omega Omega_CoG ringnumber Row_CoG_corr...
    Column_CoG_corr R_CoG TwoTheta_CoG Eta_CoG Total_Peak_Intensity...
    rowsize_peak columnsize_peak omegasize_peak N0];

%------------------------------------------------------------------------
%End of: ShellCheck_4d.m
%------------------------------------------------------------------------
```

3.12. ArrangeSpots.m

```
%---------------------------------------------------------------------------
%Start of: ArrangeSpots.m
%---------------------------------------------------------------------------
%
%Program that arranges spots coming from different parts of the same grain
%together. Since expected grain size is several times larger than the size
%of the beam, it is expected that many grains will, for a certain value of
%omega, produce a reflection at multiple slit setttings, as different parts
%of the grain are illuminated when the sample is translated.
%
%TvdZ. Last edits: February 2007.

function [M,N] = ArrangeSpots(spotlist,startnr,endnr);

%Input arguments:
%spotlist = location of file (or name of matrix) containing the spots
%startnr = first spot to include in analysis
%endnr = last spot to include in analysis
%
%Output arguments:
%M = matrix containing the spots from spotlist, with spots from different
%parts of the same grain grouped together. All groups of spots are
%numbered. Length is equal to that of spotlist; width is that of spotlist
%plus 1 (column 1, containing the numbering)
%N = matrix containing, for each group of spots, the center of mass
%coordinates and the total integrated intensity

%Define the tolerance for the deviation in spot location between different
%layers

tol = 5;

RowBC = 1038;
ColumnBC = 981;
RowBC_corr = 1038.52; %From file containing the spatial distortions
ColumnBC_corr = 981.37; %From file containing the spatial distortions
PixelSize = 47.4; %micrometer
Lsd = 241; %millimeter
Lsd_Pixels = Lsd*1000/PixelSize; %#pixels
layerwidth = 15; %microns

%After opening matrix A containing the information on the spots, refine
%startnr and endnr in such a way that the submatrix A(startnr:endnr,:)
%contains either all or none of the spots for each combination of a certain
%omega and ring number.
%The structure of matrix A will be: A = [stripe layer omega Omega_CoG...
%ringnumber Row_CoG Column_CoG R_CoG TwoTheta_CoG Eta_CoG...
%Total_Peak_Intensity rowsize_peak columnsize_peak omegasize_peak NO];
%Furthermore, A is ordered in ascending order based on the columns (from
%most to least important): ring number - round(omega_CoG) - row_CoG -
%column_CoG - stripe - layer

%A = dlmread(spotlist,'\t');
A = spotlist;
```

```
[length_spotlist width_spotlist] = size(A);
A(:,width_spotlist+1) = round(A(:,4));
A = sortrows(A,[5,(width_spotlist+1),6,7,1,2]);
A = A(:,1:width_spotlist);

while (startnr ~= 1) && (A(startnr,3) == A(startnr-1,3)) &&...
        (A(startnr,5) == A(startnr-1,5))
    startnr = startnr-1;
end
if endnr > length_spotlist
    endnr = length_spotlist;
else
    while (endnr ~= length_spotlist) && (A(endnr,3) == A(endnr+1,3)) &&...
            (A(endnr,5) == A(endnr+1,5))
        endnr = endnr+1;
    end
end
A = A(startnr:endnr,:);
[length_A width_A] = size(A);
add = zeros(length_A,1);
A = [add,A]; %This first row will be used for ticking off the spots


nr_start = 1;
reflnumber = 0;
M = [];
N = [];
ncount = 0;

while nr_start <= length_A

    if A(nr_start,1) == 1 %The spot has already been grouped
        nr_start = nr_start+1;
        continue

    else

        [length_M width_M] = size(M);
        ncount = ncount+1;

        GR = []; %This matrix will be filled with the spots grouped
                 %together
        layer = A(nr_start,3);
        omega = round(A(nr_start,5));
        ring_number = A(nr_start,6);
        row = A(nr_start,7);
        column = A(nr_start,8);
        nr_end = nr_start;
        while (round(A(nr_end,5)) == omega) &&...
            (A(nr_end,6) == ring_number) && (abs(A(nr_end,7)-row) <= 2*tol)
                %Twice the tolerance level, because since the matrix is
                %sorted in ascending order based on the row index, 'row'
                %will always be the minimum value, and therefore allowing
                %for a variation of 'tol' from the center of the spot in
                %BOTH direction requires setting the limit to 2*tol
            nr_end = nr_end+1;
            if nr_end == length_A+1
                break
            end
        end
        nr_end = nr_end-1;
        %Find all spots that appear to belong to the same grain as the spot
        %at nr_start

        j = 0;
        for i = nr_start:nr_end
            if (abs(A(i,8)-column) <= tol) && (A(i,1) == 0)
                j = j+1;
                GR(j,:) = A(i,:);
                A(i,1) = 1; %Ticking off this specific spot
            end
        end
        GR = sortrows(GR,[2,3,7,8]);

        %We now have a matrix GR containing j spots (all with the same ring
        %and omega value) that all fall on approximately the same
        %location on the detector. Hence, these spots most likely originate
        %from the same grain. The matrix is sorted in ascending
        %order on the basis of (in order of importance): stripe - layer -
        %row - column (though row and column sorting should not have any
        %effect in this case).
        %
        %Now we test whether the spots in GR form a complete set (i.e.,
        %whether there are no spots 'missing' for certain stripe or layer
        %values). If this is the case, the spots in GR all belong to the
        %same grain, and they are all numbered equally. If not, it could be
        %GR contains spots from multiple grain; therefore, the subsets are
        %numbered differently. However, by numbering the spots using non-
        %integer numbers (e.g. 2.01, 2.02, 2.03 etc.) it it still clear
        %that although strictly these have to be identified as different
        %grains, it could still be the case that in reality they do
        %originate from one and the same grain - especially if there are
        %only 1 or 2 spots missing

%----------------------------------------------------------------------
%Part 1: grouping and ordering the individual spots
%----------------------------------------------------------------------

        reflnumber = reflnumber+1;

        if j == 1 %GR contains only a single spot
            nr_stripes = 1;
            stripe1 = GR(1,2);
            stripe2 = [];
            stripe3 = [];
            GR(1,1) = reflnumber;
        elseif GR(j,2) == GR(1,2) %All spots come from the same stripe
            nr_stripes = 1;
            stripe1 = GR(1,2);
            stripe2 = [];
            stripe3 = [];
            if (GR(j,3)-GR(1,3)+1) > j %There is at least one spot missing
                reflnumber = reflnumber+0.01; %So non-integer numbering
                                              %is used
                for i = 1:(j-1)
                    GR(i,1) = reflnumber;
                    if GR(i,3) ~= GR(i+1,3)-1 %There is (at least) one spot
                                              %missing
                        reflnumber = reflnumber+0.01;
                    end
                end
                GR(j,1) = reflnumber;
                reflnumber = floor(reflnumber);
            else %No spots are missing
                GR(:,1) = reflnumber;
            end
        elseif GR(j,2)-GR(1,2) == 1 %The spots come from exactly 2 stripes
            nr_stripes = 2;
            stripe1 = GR(1,2);
            stripe2 = GR(j,2);
            stripe3 = [];
            if (j == 2*(max(GR(:,3))-min(GR(:,3))+1)) ||...
                    (j == 2*(max(GR(:,3))-min(GR(:,3))+1)-1)
                %No more than one spot is missing; it follows that the
                %spots form a single connected group in (stripe,layer)-
                %space, and therefore come from one grain
                GR(:,1) = reflnumber;
            else
                k1 = 0; %The number of spots in stripe1
                while GR(k1+1,2) == stripe1
                    k1 = k1+1;
                end
                k2 = j-k1; %The number of spots in stripe2
                %Number the groups of spots in the two stripes separately
                %(i.e. ignoring the other stripe for now)
                reflnumber = reflnumber+0.01;
                nr1 = 1; %The number of distinct spot groups in the first
                         %stripe
                if k1 == 1;
                    GR(1,1) = reflnumber;
                else
                    for i = 1:(k1-1)
                        GR(i,1) = reflnumber;
                        if GR(i,3) ~= GR(i+1,3)-1 %There is (at least) one
                                                  %spot missing
                            reflnumber = reflnumber+0.01;
                            nr1 = nr1+1;
                        end
                    end
                    GR(k1,1) = reflnumber;
                end
                reflnumber = reflnumber+0.01;
                nr2 = 1; %The number of distinct spot groups in the second
                         %stripe
                nr2ind(1,1) = k1+1; %Vector containing the locations of the
                                    %starts of the new groups
                if k2 == 1;
                    GR(j,1) = reflnumber;
                else
                    for i = (k1+1):(j-1)
                        GR(i,1) = reflnumber;
                        if GR(i,3) ~= GR(i+1,3)-1 %There is (at least) one
                                                  %spot missing
                            reflnumber = reflnumber+0.01;
                            nr2 = nr2+1;
                            nr2ind(nr2,1) = i+1;
                        end
                    end
                    GR(j,1) = reflnumber;
                end
                GRstr1 = GR((1:k1),:); %The spots in stripe1; size =
                                       %[k1 width_A]
```

```
GRstr2 = GR((k1+1:j),:); %The spots in stripe2; size =
                         %[k2 width_A]
%The groups of spots in each stripe are numbered while
%ignoring the other stripe. However, groups that appear
%distinct in one stripe can be connected by a third group
%in the other stripe, thus forming one large group
%representing a single grain. This complication is dealt
%with in the following.
GRstr1lay = GRstr1(:,3);
GRstr2lay = GRstr2(:,3);
nr2ind = nr2ind-k1; %Vector nr2ind, of length nr2, now
    %refers to GRstr2 and GRstr2lay instead of to GR
nr2ind(nr2+1,1) = k2+1;
for groupnr = 1:nr2 %Looping over all spot groups in the
                    %second stripe
    ind1 = [];
    for i1 = nr2ind(groupnr):(nr2ind(groupnr+1)-1)
                %Looping over all spots in a group
        str2lay = GRstr2lay(i1);
                %Find the layer of the spot in question
        ind1 = find(GRstr1lay == str2lay);
                %Does this layer match the layer of one of
                %the spots in the first stripe?
        if isempty(ind1) == 0
                %If so, the entire group is renumbered to
                %match the number of the group in the first
                %stripe
            numind1 = GRstr1(ind1(1),1);
            GRstr2(nr2ind(groupnr):...
                (nr2ind(groupnr+1)-1),1) = numind1;
            ind2 = [];
            if i1 ~= nr2ind(groupnr+1)-1
                for i2 = (i1+1):nr2ind(groupnr+1)-1
                    str2lay = GRstr2lay(i2);
                    ind2 = find(GRstr1lay == str2lay);
                        %Check if the group in stripe2
                        %connects to even more groups in
                        %stripe1
                    if isempty(ind2) == 0
                        %If so, these other groups are also
                        %renumbered to match the first
                        %group
                        numind2 = GRstr1(ind2(1),1);
                        if numind1 ~= numind2
                            for i3 = ind2(1):k1
                                if GRstr1(i3,1) == numind2
                                    GRstr1(i3,1) = numind1;
                                end
                            end
                        end
                        ind2 = [];
                    end
                end
            end
            break %A match with stripe1 is found, so the
                  %loop over i1 is terminated
        end
    end
end
%All spots in the two layers have been numbered. However,
%due to the renumbering process it is possible the numbers
```

```
        %no longer form an equidistant set (e.g. 2.01, 2.02, 2.04,
        %2.05 and 2.07 instead of 2.01 through 2.05). So, an extra
        %renumbering step is required.
        GR((1:k1),:) = GRstr1;
        GR((k1+1:j),:) = GRstr2;
        GR = sortrows(GR,[1,2,3,7,8]);
        %Sort on the basis of reflnumber (then stripe, layer, row,
        %column)
        for i = 1:(j-1)
            diff = round(100*(GR(i+1,1)-GR(i,1)));
            switch diff
                case 0 %Spots i and (i+1) belong to the same group,
                       %so no renumbering required
                    continue
                case 1 %Spots i and (i+1) belong to subsequent
                       %groups, so no renumbering required
                    continue
                otherwise %Spots i and (i+1) belong to different
                          %groups; renumbering required
                    GR((i+1):j,1) = GR((i+1):j,1)-(diff/100-0.01);
            end
        end
        reflnumber = floor(reflnumber);
    end
else %The spots come from three (or only the two outer) stripes
    nr_stripes = 3;
    stripe1 = 0;
    stripe2 = 1;
    stripe3 = 2;
    if (j == 3*(max(GR(:,3))-min(GR(:,3))+1)) ||...
       (j == 3*(max(GR(:,3))-min(GR(:,3))+1)-1) ||...
       (j == 3*(max(GR(:,3))-min(GR(:,3))+1)-2)
        %No more than two spots are missing; it follows that the
        %spots form a single connected group in (stripe,layer)-
        %space, and therefore come from one grain
        GR(:,1) = reflnumber;
    else
        nr2ind = []; %Vector containing the locations of the starts
                     %of the groups in stripe2
        nr3ind = []; %Vector containing the locations of the starts
                     %of the groups in stripe3
        k1 = 0; %The number of spots in the first stripe
        while GR(k1+1,2) == stripe1
            k1 = k1+1;
        end
        k2 = 0; %The number of spots in the second stripe
        while GR(k1+k2+1,2) == stripe2
            k2 = k2+1;
        end
        k3 = j-k1-k2; %The number of spots in the third stripe
        %Number the groups of spots in the three stripes separately
        %(i.e. ignoring the other stripes for now)
        reflnumber = reflnumber+0.01;
        nr1 = 1; %The number of distinct spot groups in the first
                 %stripe
        if k1 == 1;
            GR(1,1) = reflnumber;
        else
            for i = 1:(k1-1)
                GR(i,1) = reflnumber;
                if GR(i,3) ~= GR(i+1,3)-1 %There is (at least) one
```

```
                                    %spot missing
                reflnumber = reflnumber+0.01;
                nr1 = nr1+1;
            end
        end
        GR(k1,1) = reflnumber;
    end
    reflnumber = reflnumber+0.01;
    nr2 = 1; %The number of distinct spot groups in the second
             %stripe
    nr2ind(1,1) = k1+1;
    if k2 == 1;
        GR(k1+k2,1) = reflnumber;
    else
        for i = (k1+1):(k1+k2-1)
            GR(i,1) = reflnumber;
            if GR(i,3) ~= GR(i+1,3)-1 %There is (at least) one
                                      %spot missing
                reflnumber = reflnumber+0.01;
                nr2 = nr2+1;
                nr2ind(nr2,1) = i+1;
            end
        end
        GR(k1+k2,1) = reflnumber;
    end
    reflnumber = reflnumber+0.01;
    nr3 = 1; %The number of distinct spot groups in the third
             %stripe
    nr3ind(1,1) = k1+k2+1;
    if k2 == 1;
        GR(j,1) = reflnumber;
    else
        for i = (k1+k2+1):(k1+k2+k3-1)
            GR(i,1) = reflnumber;
            if GR(i,3) ~= GR(i+1,3)-1 %There is (at least) one
                                      %spot missing
                reflnumber = reflnumber+0.01;
                nr3 = nr3+1;
                nr3ind(nr3,1) = i+1;
            end
        end
        GR(j,1) = reflnumber;
    end
    GRstr1 = GR((1:k1),:); %The spots in stripe1; size =
                           %[k1 width_A]
    GRstr2 = GR((k1+1:k1+k2),:); %The spots in stripe2; size =
                                 %[k2 width_A]
    GRstr3 = GR((k1+k2+1:j),:); %The spots in stripe3; size =
                                %[k3 width_A]
    %The groups of spots in each stripe are numbered while
    %ignoring the other stripes. However, groups that appear
    %distinct in one stripe can be connected by a third group
    %in another stripe, thus forming one large group
    %representing a single grain. This complication is dealt
    %with in the following.
    GRstr1lay = GRstr1(:,3);
    GRstr2lay = GRstr2(:,3);
    GRstr3lay = GRstr3(:,3);
    nr2ind = nr2ind-k1; %Vector nr2ind, of length nr2, now
        %refers to GRstr2 and GRstr2lay instead of to GR
    nr2ind(nr2+1,1) = k2+1;
```

```
nr3ind = nr3ind-k1-k2; %Vector nr3ind, of length nr3, now
    %refers to GRstr3 and GRstr3lay instead of to GR
nr3ind(nr3+1,1) = k3+1;
%We first check whether any of the spot groups in the third
%stripe link up to groups in the second stripe.
for groupnr = 1:nr3 %Looping over all spot groups in the
                    %third stripe
    ind1 = [];
    for i1 = nr3ind(groupnr):(nr3ind(groupnr+1)-1)
            %Looping over all spots in a group
        str3lay = GRstr3lay(i1);
            %Find the layer of the spot in question
        ind1 = find(GRstr2lay == str3lay);
            %Does this layer match with the layer of one of
            %the spots in the second stripe?
        if isempty(ind1) == 0
            %If so, the entire group is renumbered to match
            %the number of the group in the second stripe
            numind1 = GRstr2(ind1(1),1);
            GRstr3(nr3ind(groupnr):...
                (nr3ind(groupnr+1)-1),1) = numind1;
            ind2 = [];
            if i1 ~= nr3ind(groupnr+1)-1
                for i2 = (i1+1):nr3ind(groupnr+1)-1
                    str3lay = GRstr3lay(i2);
                    ind2 = find(GRstr2lay == str3lay);
                        %Check if the group in stripe3
                        %connects to even more groups in
                        %stripe2
                    if isempty(ind2) == 0
                        %If so, these other groups are also
                        %renumbered to match the first
                        %group
                        numind2 = GRstr2(ind2(1),1);
                        if numind1 ~= numind2
                            for i3 = ind2(1):k2
                                if GRstr2(i3,1) == numind2
                                    GRstr2(i3,1) = numind1;
                                end
                            end
                        end
                        ind2 = [];
                    end
                end
            end
            break %A match with stripe2 is found, so the
                  %loop over i1 is terminated
        end
    end
end
%We then check for links between groups in the second and
%first stripe
for groupnr = 1:nr2 %Looping over all spot groups in the
                    %second stripe
    ind1 = [];
    for i1 = nr2ind(groupnr):(nr2ind(groupnr+1)-1)
            %Looping over all spots in a group
        str2lay = GRstr2lay(i1);
            %Find the layer of the spot in question
        ind1 = find(GRstr1lay == str2lay);
            %Does this layer match with the layer of one of
```

```
                %the spots in the first stripe?
            if isempty(ind1) == 0
                %If so, the entire group is renumbered to match
                %the number of the group in the first stripe
                numind1 = GRstr1(ind1(1),1);
                if str2lay > numind1
                    for i3 = 1:k2
                        if GRstr2(i3,1) == str2lay
                            GRstr2(i3,1) = numind1;
                            %Rename the part of the group in
                            %stripe2
                        end
                    end
                    for i4 = 1:k3
                        if GRstr3(i4,1) == str2lay
                            GRstr3(i4,1) = numind1;
                            %And rename the part of the group
                            %in stripe3, if any
                        end
                    end
                    ind2 = [];
                    if i1 ~= nr2ind(groupnr+1)-1
                        for i2 = (i1+1):nr2ind(groupnr+1)-1
                            str2lay = GRstr2lay(i2);
                            ind2 = find(GRstr1lay == str2lay);
                                %Check if the group in stripe2
                                %connects to even more groups
                                %in stripe1
                            if isempty(ind2) == 0
                                %If so, these other groups are
                                %also renumbered to match the
                                %first group
                                numind2 = GRstr1(ind2(1),1);
                                if numind1 ~= numind2
                                    for i3 = ind2(1):k1
                                        if GRstr1(i3,1) ==...
                                                numind2
                                              GRstr1(i3,1) =...
                                                numind1;
                                        end
                                    end
                                end
                                ind2 = [];
                            end
                        end
                    end
                else
                    for i3 = 1:k1
                        if GRstr1(i3,1) == numind1
                            GRstr(i3,1) = str2lay;
                        end
                    end
                end
                break %A match with stripe1 is found, so the
                        %loop over i1 is terminated
            end
        end
    end
    GR((1:k1),:) = GRstr1;
    GR((k1+1:k1+k2),:) = GRstr2;
    GR((k1+k2+1:j),:) = GRstr3;
```

```
                GR = sortrows(GR,[1,2,3,7,8]);
                %Sort on the basis of reflnumber (then stripe, layer, row,
                %column)
                for i = 1:(j-1)
                    diff = round(100*(GR(i+1,1)-GR(i,1)));
                    switch diff
                        case 0 %Spots i and (i+1) belong to the same group,
                               %so no renumbering required
                            continue
                        case 1 %Spots i and (i+1) belong to subsequent
                               %groups, so no renumbering required
                            continue
                        otherwise %Spots i and (i+1) belong to different
                                  %groups - renumbering required
                            GR((i+1):j,1) = GR((i+1):j,1)-(diff/100-0.01);
                    end
                end
                reflnumber = floor(reflnumber);
            end
        end

        M((length_M+1:length_M+j),:) = GR;
        nr_start = nr_start+1;

%-----------------------------------------------------------------------------
%Part 2: computing average coordinates and total intensity
%-----------------------------------------------------------------------------

        nr_groups = round(((GR(j,1)-GR(1,1))/0.01)+1);
        for grp = 1:nr_groups
            [length_N width_N] = size(N);
            if grp == 1
                ind1 = 1; %First entry belonging to this group of spots
                ind2 = 1; %First entry belonging to the next group of spots
                          %(to be refined)
            end
            ind1 = ind2;
            if grp == nr_groups
                ind2 = j+1;
            else
                while (GR(ind2,1) == GR(ind1,1))
                    ind2 = ind2+1;
                end
            end
            Int_total = 0;
            stripe_CoM = 0;
            layer_CoM = 0;
            omega_CoM = 0;
            row_CoM = 0;
            column_CoM = 0;
            NRofPixels = 0;

            if GR(ind1,2) == GR(ind2-1,2)
                %All spots in the group lie within the same stripe
                L = []; %Matrix that will be filled with the starting and
                        %finishing coordinates of all layers in the group
                for i = ind1:(ind2-1)
                    if i == ind1
                        l1 = 0; %Left border of layer GR(i,:)
                        l2 = 15; %Right border of layer GR(i,:)
                    else
                for i2 = (layerprev+1):GR(i,3)
                    rem1 = rem(i2,5);
                    if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
                        %The next layer is shifted 5 mu
                        l1 = l1+5;
                        l2 = l2+5;
                    else
                        %The next layer is shifted 7.5 mu
                        l1 = l1+7.5;
                        l2 = l2+7.5;
                    end
                end
            end
            layerprev = GR(i,3);
            L(i-ind1+1,1) = l1;
            L(i-ind1+1,2) = l2;
            L(i-ind1+1,3) = GR(i,12);
            %Now compute weighted coordinates
            omega_CoM = omega_CoM+GR(i,5)*GR(i,12);
            row_CoM = row_CoM+GR(i,7)*GR(i,12);
            column_CoM = column_CoM+GR(i,8)*GR(i,12);
            NRofPixels = NRofPixels+GR(i,16);
        end
        Raw_Int = sum(GR(ind1:(ind2-1),12));
        omega_CoM = omega_CoM/Raw_Int;
        row_CoM = row_CoM/Raw_Int;
        column_CoM = column_CoM/Raw_Int;
        R_CoM = sqrt((row_CoM-RowBC_corr)^2+...
                (column_CoM-ColumnBC_corr)^2);
        TwoTheta_CoM = (180/pi)*atan2(R_CoM,Lsd_Pixels);
        Eta_CoM = (180/pi)*atan2(column_CoM-ColumnBC_corr,...
                    RowBC_corr-row_CoM);
        if Eta_CoM < 0
            Eta_CoM = Eta_CoM+360;
        end

        %Because of the overlap of the individual layers, the
        %resolution in the layer dimension becomes larger than the
        %layer width. Therefore the Center of Mass layer coordinate
        %is recomputed using refined intensities as weighting
        %factors.
        stepnr = L(ind2-ind1,2)/2.5;
            %Width of the layer region (# of 2.5 micron steps)
        Int_corr = zeros(stepnr,2);
            %Matrix that will be filled with the corrected
            %intensities of all of the 2.5 micron steps
        for i = 1:stepnr
            l3 = (i-1)*2.5;
            Int_corr(i,1) = l3;
            divider = 0;
            for ind3 = 1:(ind2-ind1) %Loop over matrix L
                if (l3 >= L(ind3,1)) && (l3 < L(ind3,2))
                    Int_corr(i,2) = Int_corr(i,2)+L(ind3,3);
                    divider = divider+1;
                elseif l3 < L(ind3,1)
                    break
                end
            end
            if divider ~= 0
                Int_corr(i,2) = Int_corr(i,2)*(2.5/15)/divider;
            end
```

```
end
Int_corr_save = ones(stepnr,3);
Int_corr_save(:,1) = GR(ind1,2)*Int_corr_save(:,1);
Int_corr_save(:,2:3) = Int_corr(:,:);
if nr_groups > 1
    if reflnumber < 10
        reflnrstring = ['000' num2str(reflnumber) '.0'...
            num2str(grp)];
    elseif reflnumber < 100
        reflnrstring = ['00' num2str(reflnumber) '.0'...
            num2str(grp)];
    elseif reflnumber < 1000
        reflnrstring = ['0' num2str(reflnumber) '.0'...
            num2str(grp)];
    else
        reflnrstring = [num2str(reflnumber) '.0'...
            num2str(grp)];
    end
else
    if reflnumber < 10
        reflnrstring = ['000' num2str(reflnumber)];
    elseif reflnumber < 100
        reflnrstring = ['00' num2str(reflnumber)];
    elseif reflnumber < 1000
        reflnrstring = ['0' num2str(reflnumber)];
    else
        reflnrstring = [num2str(reflnumber)];
    end
end
writestring = ['dlmwrite(''E:\4d_4e_Analysis\4d_'...
    'Intensity_Profiles\grain' reflnrstring '.txt'','...
    'Int_corr_save,''delimiter'',''\t'',''precision'','...
    '''%.2f'')'];
eval(writestring);
Int_total = sum(Int_corr(:,2));
    %Total grain intensity, corrected for layer overlap
%Now compute the layer_CoM coordinate, using the refined
%intensities as weights.
layer_ref = sum((Int_corr(:,1)+1.25).*Int_corr(:,2));
layer_ref = layer_ref/Int_total;
dum1 = floor(layer_ref/30);
dum2 = rem(layer_ref,30);
layer_CoM = GR(ind1,3)+5*dum1;
    %30 microns correspond to 5 layers
while dum2 >= 15
    rem2 = rem(layer_CoM,5);
    layer_CoM = layer_CoM+1;
    if (rem2 == 0) || (rem2 == 2) || (rem2 == 4)
        dum2 = dum2-5;
    else
        dum2 = dum2-7.5;
    end
end
layer_CoM = layer_CoM+dum2/15;
layer_CoM = layer_CoM-0.5;
    %So that an integer value for layer_CoM corresponds to
    %the middle of the layer instead of the beginning

N(length_N+1,1) = GR(ind1,1);
N(length_N+1,2) = GR(ind1,2);
N(length_N+1,3) = layer_CoM;
```

```
                N(length_N+1,4)  = omega_CoM;
                N(length_N+1,5)  = GR(1,6);
                N(length_N+1,6)  = row_CoM;
                N(length_N+1,7)  = column_CoM;
                N(length_N+1,8)  = R_CoM;
                N(length_N+1,9)  = TwoTheta_CoM;
                N(length_N+1,10)  = Eta_CoM;
                N(length_N+1,11)  = NRofPixels;
                N(length_N+1,12)  = Int_total;

        elseif GR(ind1,2) == GR(ind2-1,2)-1
                %The spots in the group come from two stripes

            L1 = [];%Matrix that will be filled with the starting and
                    %finishing coordinates of all layers of the spots
                    %in the first stripe of the group
            L2 = [];%Matrix that will be filled with the starting and
                    %finishing coordinates of all layers of the spots
                    %in the second stripe of the group
            ind12 = ind1;
            while GR(ind12+1,2) == GR(ind1,2)
                ind12 = ind12+1;
            end
            for i = ind1:(ind12)
                if i == ind1
                    l1 = 0; %Left border of layer GR(i,:)
                    l2 = 15; %Right border of layer GR(i,:)
                else
                    for i2 = (layerprev+1):GR(i,3)
                        rem1 = rem(i2,5);
                        if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
                            %The next layer is shifted 5 mu
                            l1 = l1+5;
                            l2 = l2+5;
                        else
                            %The next layer is shifted 7.5 mu
                            l1 = l1+7.5;
                            l2 = l2+7.5;
                        end
                    end
                end
                layerprev = GR(i,3);
                L1(i-ind1+1,1) = l1;
                L1(i-ind1+1,2) = l2;
                L1(i-ind1+1,3) = GR(i,12);
                %Compute weighted coordinates
                stripe_CoM = stripe_CoM+GR(i,2)*GR(i,12);
                omega_CoM = omega_CoM+GR(i,5)*GR(i,12);
                row_CoM = row_CoM+GR(i,7)*GR(i,12);
                column_CoM = column_CoM+GR(i,8)*GR(i,12);
                NRofPixels = NRofPixels+GR(i,16);
            end
            for i = (ind12+1):(ind2-1)
                if i == (ind12+1)
                    l1 = 0; %Left border of layer GR(i,:)
                    l2 = 15; %Right border of layer GR(i,:)
                else
                    for i2 = (layerprev+1):GR(i,3)
                        rem1 = rem(i2,5);
                        if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
                            %The next layer is shifted 5 mu
                l1 = l1+5;
                l2 = l2+5;
            else
                %The next layer is shifted 7.5 mu
                l1 = l1+7.5;
                l2 = l2+7.5;
            end
        end
    end
    layerprev = GR(i,3);
    L2(i-ind12,1) = l1;
    L2(i-ind12,2) = l2;
    L2(i-ind12,3) = GR(i,12);
    %Compute weighted coordinates
    stripe_CoM = stripe_CoM+GR(i,2)*GR(i,12);
    omega_CoM = omega_CoM+GR(i,5)*GR(i,12);
    row_CoM = row_CoM+GR(i,7)*GR(i,12);
    column_CoM = column_CoM+GR(i,8)*GR(i,12);
    NRofPixels = NRofPixels+GR(i,16);
end
Raw_Int = sum(GR(ind1:(ind2-1),12));
stripe_CoM = stripe_CoM/Raw_Int;
omega_CoM = omega_CoM/Raw_Int;
row_CoM = row_CoM/Raw_Int;
column_CoM = column_CoM/Raw_Int;
R_CoM = sqrt((row_CoM-RowBC_corr)^2+...
        (column_CoM-ColumnBC_corr)^2);
TwoTheta_CoM = (180/pi)*atan2(R_CoM,Lsd_Pixels);
Eta_CoM = (180/pi)*atan2(column_CoM-ColumnBC_corr,...
           RowBC_corr-row_CoM);
if Eta_CoM < 0
    Eta_CoM = Eta_CoM+360;
end

%Within matrices L1 and L2, columns 1 and 2 (containing the
%left and right border of the slit positions) are defined
%with respect to the first slit position for which
%intensity is found in that specific stripe. However, we
%want to define them both w.r.t. the SAME first slit
%position, so that later on weighting of the slit position
%based on the intensities can be performed. Hence the
%following:
diff = GR(ind1,3)-GR(ind12+1,3);
if diff < 0 %Starting slit in stripe1 is smaller than in
            %stripe2; redefine L2(:,1:2) to match the
            %stripe1 starting slit
    layer_start = GR(ind1,3);
    for i = 0:(abs(diff)-1)
        rem1 = rem(GR(ind12+1,3)-i,5);
        if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
            L2(:,1:2) = L2(:,1:2)+5;
        else
            L2(:,1:2) = L2(:,1:2)+7.5;
        end
    end
elseif diff > 0
    %Starting slit in stripe2 is smaller than in stripe1
    layer_start = GR(ind12+1,3);
    for i = 0:(abs(diff)-1)
        rem1 = rem(GR(ind1,3)-i,5);
        if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
```

```
                L1(:,1:2) = L1(:,1:2)+5;
            else
                L1(:,1:2) = L1(:,1:2)+7.5;
            end
        end
    else
        %Both stripes start at the same layer; no action
        %required
        layer_start = GR(ind1,3);
    end

    stepnr1 = L1(ind12+1-ind1,2)/2.5;
        %Width of the layer region of stripe1 in 2.5 mu steps
    stepnr2 = L2(ind2-ind12-1,2)/2.5;
        %Width of the layer region of stripe2 in 2.5 mu steps
    Int_corr1 = zeros(stepnr1,2);
        %Matrix that will be filled with the corrected
        %intensities of all of the 2.5 micron steps in stripe1
    Int_corr2 = zeros(stepnr2,2);
        %Matrix that will be filled with the corrected
        %intensities of all of the 2.5 micron steps in stripe2
    for i = 1:stepnr1
        l3 = (i-1)*2.5;
        Int_corr1(i,1) = l3;
        divider = 0;
        for ind3 = 1:(ind12+1-ind1) %Loop over matrix L1
            if (l3 >= L1(ind3,1)) && (l3 < L1(ind3,2))
                Int_corr1(i,2) = Int_corr1(i,2)+L1(ind3,3);
                divider = divider+1;
            elseif l3 < L1(ind3,1)
                break
            end
        end
        if divider ~= 0
            Int_corr1(i,2) = Int_corr1(i,2)*(2.5/15)/divider;
        end
    end
    for i = 1:stepnr2
        l3 = (i-1)*2.5;
        Int_corr2(i,1) = l3;
        divider = 0;
        for ind3 = 1:(ind2-ind12-1) %Loop over matrix L2
            if (l3 >= L2(ind3,1)) && (l3 < L2(ind3,2))
                Int_corr2(i,2) = Int_corr2(i,2)+L2(ind3,3);
                divider = divider+1;
            elseif l3 < L2(ind3,1)
                break
            end
        end
        if divider ~= 0
            Int_corr2(i,2) = Int_corr2(i,2)*(2.5/15)/divider;
        end
    end
    Int_corr1_save = ones(stepnr1,3);
    Int_corr1_save(:,1) = GR(ind1,2)*Int_corr1_save(:,1);
    Int_corr1_save(:,2:3) = Int_corr1(:,:);
    Int_corr2_save = ones(stepnr2,3);
    Int_corr2_save(:,1) = GR(ind2-1,2)*Int_corr2_save(:,1);
    Int_corr2_save(:,2:3) = Int_corr2(:,:);
    Int_corr_save = zeros((stepnr1+stepnr2),3);
    Int_corr_save(1:stepnr1,:) = Int_corr1_save;
```

```
Int_corr_save(stepnr1+1:stepnr1+stepnr2,:) =...
    Int_corr2_save;
if nr_groups > 1
    if reflnumber < 10
        reflnrstring = ['000' num2str(reflnumber) '.0'...
            num2str(grp)];
    elseif reflnumber < 100
        reflnrstring = ['00' num2str(reflnumber) '.0'...
            num2str(grp)];
    elseif reflnumber < 1000
        reflnrstring = ['0' num2str(reflnumber) '.0'...
            num2str(grp)];
    else
        reflnrstring = [num2str(reflnumber) '.0'...
            num2str(grp)];
    end
else
    if reflnumber < 10
        reflnrstring = ['000' num2str(reflnumber)];
    elseif reflnumber < 100
        reflnrstring = ['00' num2str(reflnumber)];
    elseif reflnumber < 1000
        reflnrstring = ['0' num2str(reflnumber)];
    else
        reflnrstring = [num2str(reflnumber)];
    end
end
writestring = ['dlmwrite(''E:\4d_4e_Analysis\4d_'...
    'Intensity_Profiles\grain' reflnrstring '.txt'','...
    'Int_corr_save,''delimiter'',''\t'',''precision'','...
    '''%.2f'')'];
eval(writestring);
Int_total = sum(Int_corr1(:,2))+sum(Int_corr2(:,2));
%Total grain intensity, corrected for layer overlap

%Now compute the layer_CoM coordinate, using the refined
%intensities as weights.
layer_ref = sum((Int_corr1(:,1)+1.25).*Int_corr1(:,2))+...
    sum((Int_corr2(:,1)+1.25).*Int_corr2(:,2));
layer_ref = layer_ref/Int_total;
dum1 = floor(layer_ref/30);
dum2 = rem(layer_ref,30);
layer_CoM = layer_start+5*dum1;
    %30 microns correspond to 5 layers
while dum2 >= 15
    rem2 = rem(layer_CoM,5);
    layer_CoM = layer_CoM+1;
    if (rem2 == 0) || (rem2 == 2) || (rem2 == 4)
        dum2 = dum2-5;
    else
        dum2 = dum2-7.5;
    end
end
layer_CoM = layer_CoM+dum2/15;
layer_CoM = layer_CoM-0.5;
    %So that an integer value for layer_CoM corresponds to
    %the middle of the layer instead of the beginning

N(length_N+1,1) = GR(ind1,1);
N(length_N+1,2) = stripe_CoM;
N(length_N+1,3) = layer_CoM;
```

```
                N(length_N+1,4)  = omega_CoM;
                N(length_N+1,5)  = GR(1,6);
                N(length_N+1,6)  = row_CoM;
                N(length_N+1,7)  = column_CoM;
                N(length_N+1,8)  = R_CoM;
                N(length_N+1,9)  = TwoTheta_CoM;
                N(length_N+1,10)  = Eta_CoM;
                N(length_N+1,11)  = NRofPixels;
                N(length_N+1,12)  = Int_total;

            else %The spots in the group come from all three stripes

                L1 = [];%Matrix that will be filled with the starting and
                        %finishing coordinates of all layers of the spots
                        %in the first stripe of the group
                L2 = [];%Matrix that will be filled with the starting and
                        %finishing coordinates of all layers of the spots
                        %in the second stripe of the group
                L3 = [];%Matrix that will be filled with the starting and
                        %finishing coordinates of all layers of the spots
                        %in the third stripe of the group
                ind12 = ind1;
                while GR(ind12+1,2) == GR(ind1,2)
                    ind12 = ind12+1;
                end
                ind23 = ind12+1;
                while GR(ind23+1,2) == GR(ind12+1,2)
                    ind23 = ind23+1;
                end
                for i = ind1:(ind12)
                    if i == ind1
                        l1 = 0; %Left border of layer GR(i,:)
                        l2 = 15; %Right border of layer GR(i,:)
                    else
                        for i2 = (layerprev+1):GR(i,3)
                            rem1 = rem(i2,5);
                            if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
                                %The next layer is shifted 5 mu
                                l1 = l1+5;
                                l2 = l2+5;
                            else
                                %The next layer is shifted 7.5 mu
                                l1 = l1+7.5;
                                l2 = l2+7.5;
                            end
                        end
                    end
                    layerprev = GR(i,3);
                    L1(i-ind1+1,1) = l1;
                    L1(i-ind1+1,2) = l2;
                    L1(i-ind1+1,3) = GR(i,12);
                    %Compute weighted coordinates
                    stripe_CoM = stripe_CoM+GR(i,2)*GR(i,12);
                    omega_CoM = omega_CoM+GR(i,5)*GR(i,12);
                    row_CoM = row_CoM+GR(i,7)*GR(i,12);
                    column_CoM = column_CoM+GR(i,8)*GR(i,12);
                    NRofPixels = NRofPixels+GR(i,16);
                end
                for i = (ind12+1):(ind23)
                    if i == (ind12+1)
                        l1 = 0; %Left border of layer GR(i,:)
            12 = 15; %Right border of layer GR(i,:)
        else
            for i2 = (layerprev+1):GR(i,3)
                rem1 = rem((i2),5);
                if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
                    %The next layer is shifted 5 mu
                    11 = 11+5;
                    12 = 12+5;
                else
                    %The next layer is shifted 7.5 mu
                    11 = 11+7.5;
                    12 = 12+7.5;
                end
            end
        end
        layerprev = GR(i,3);
        L2(i-ind12,1) = 11;
        L2(i-ind12,2) = 12;
        L2(i-ind12,3) = GR(i,12);
        %Compute weighted coordinates
        stripe_CoM = stripe_CoM+GR(i,2)*GR(i,12);
        omega_CoM = omega_CoM+GR(i,5)*GR(i,12);
        row_CoM = row_CoM+GR(i,7)*GR(i,12);
        column_CoM = column_CoM+GR(i,8)*GR(i,12);
        NRofPixels = NRofPixels+GR(i,16);
    end
    for i = (ind23+1):(ind2-1)
        if i == (ind23+1)
            11 = 0; %Left border of layer GR(i,:)
            12 = 15; %Right border of layer GR(i,:)
        else
            for i2 = (layerprev+1):GR(i,3)
            rem1 = rem(i2,5);
                if (rem1 == 0) || (rem1 == 1) || (rem1 == 3)
                    %The next layer is shifted 5 mu
                    11 = 11+5;
                    12 = 12+5;
                else
                    %The next layer is shifted 7.5 mu
                    11 = 11+7.5;
                    12 = 12+7.5;
                end
            end
        end
        layerprev = GR(i,3);
        L3(i-ind23,1) = 11;
        L3(i-ind23,2) = 12;
        L3(i-ind23,3) = GR(i,12);
        %Compute weighted coordinates
        stripe_CoM = stripe_CoM+GR(i,2)*GR(i,12);
        omega_CoM = omega_CoM+GR(i,5)*GR(i,12);
        row_CoM = row_CoM+GR(i,7)*GR(i,12);
        column_CoM = column_CoM+GR(i,8)*GR(i,12);
        NRofPixels = NRofPixels+GR(i,16);
    end
    Raw_Int = sum(GR(ind1:(ind2-1),12));
    stripe_CoM = stripe_CoM/Raw_Int;
    omega_CoM = omega_CoM/Raw_Int;
    row_CoM = row_CoM/Raw_Int;
    column_CoM = column_CoM/Raw_Int;
    R_CoM = sqrt((row_CoM-RowBC_corr)^2+...
        (column_CoM-ColumnBC_corr)^2);
TwoTheta_CoM = (180/pi)*atan2(R_CoM,Lsd_Pixels);
Eta_CoM = (180/pi)*atan2(column_CoM-ColumnBC_corr,...
           RowBC_corr-row_CoM);
if Eta_CoM < 0
    Eta_CoM = Eta_CoM+360;
end

%Within matrices L1, L2 and L3, columns 1 and 2 (containing
%the left and right border of the slit positions) are
%defined with respect to the first slit position for which
%intensity is found in that specific stripe. However, we
%want to define them all w.r.t. the SAME first slit
%position, so that later on weighting of the slit position
%based on the intensities can be performed. Hence the
%following:
if (GR(ind1,3) == GR(ind12+1,3)) &&...
    (GR(ind1,3) == GR(ind23+1,3))
    %Easiest case
    layer_start = GR(ind1,3);
else
    [layer_start lindex] = min([GR(ind1,3) GR(ind12+1,3)...
                                GR(ind23+1,3)]);
    switch lindex
        case 1
            diff1 = GR(ind12+1,3)-layer_start;
            diff2 = GR(ind23+1,3)-layer_start;
            if diff1 > 0
                for i = 0:(diff1-1)
                    rem1 = rem(layer_start+diff1-i,5);
                    if (rem1 == 0) || (rem1 == 1) ||...
                       (rem1 == 3)
                        L2(:,1:2) = L2(:,1:2)+5;
                    else
                        L2(:,1:2) = L2(:,1:2)+7.5;
                    end
                end
            end
            if diff2 > 0
                for i = 0:(diff2-1)
                    rem1 = rem(layer_start+diff2-i,5);
                    if (rem1 == 0) || (rem1 == 1) ||...
                       (rem1 == 3)
                        L3(:,1:2) = L3(:,1:2)+5;
                    else
                        L3(:,1:2) = L3(:,1:2)+7.5;
                    end
                end
            end
        case 2
            diff1 = GR(ind1,3)-layer_start;
            diff2 = GR(ind23+1,3)-layer_start;
            if diff1 > 0
                for i = 0:(diff1-1)
                    rem1 = rem(layer_start+diff1-i,5);
                    if (rem1 == 0) || (rem1 == 1) ||...
                       (rem1 == 3)
                        L1(:,1:2) = L1(:,1:2)+5;
                    else
                        L1(:,1:2) = L1(:,1:2)+7.5;
                    end
                end
            end
            if diff2 > 0
                for i = 0:(diff2-1)
                    rem1 = rem(layer_start+diff2-i,5);
                    if (rem1 == 0) || (rem1 == 1) ||...
                       (rem1 == 3)
                        L3(:,1:2) = L3(:,1:2)+5;
                    else
                        L3(:,1:2) = L3(:,1:2)+7.5;
                    end
                end
            end
        case 3
            diff1 = GR(ind1,3)-layer_start;
            diff2 = GR(ind12+1,3)-layer_start;
            if diff1 > 0
                for i = 0:(diff1-1)
                    rem1 = rem(layer_start+diff1-i,5);
                    if (rem1 == 0) || (rem1 == 1) ||....
                       (rem1 == 3)
                        L1(:,1:2) = L1(:,1:2)+5;
                    else
                        L1(:,1:2) = L1(:,1:2)+7.5;
                    end
                end
            end
            if diff2 > 0
                for i = 0:(diff2-1)
                    rem1 = rem(layer_start+diff2-i,5);
                    if (rem1 == 0) || (rem1 == 1) ||...
                       (rem1 == 3)
                        L2(:,1:2) = L2(:,1:2)+5;
                    else
                        L2(:,1:2) = L2(:,1:2)+7.5;
                    end
                end
            end
        otherwise
    end
end

stepnr1 = L1(ind12+1-ind1,2)/2.5;
stepnr2 = L2(ind23-ind12,2)/2.5;
stepnr3 = L3(ind2-ind23-1,2)/2.5;
Int_corr1 = zeros(stepnr1,2);
Int_corr2 = zeros(stepnr2,2);
Int_corr3 = zeros(stepnr3,2);
for i = 1:stepnr1
    l3 = (i-1)*2.5;
    Int_corr1(i,1) = l3;
    divider = 0;
    for ind3 = 1:(ind12+1-ind1) %Loop over matrix L1
        if (l3 >= L1(ind3,1)) && (l3 < L1(ind3,2))
            Int_corr1(i,2) = Int_corr1(i,2)+L1(ind3,3);
            divider = divider+1;
        elseif l3 < L1(ind3,1)
            break
        end
    end
    if divider ~= 0
```

```
                reflnrstring = [num2str(reflnumber) '.0'...
                    num2str(grp)];
            end
        else
            if reflnumber < 10
                reflnrstring = ['000' num2str(reflnumber)];
            elseif reflnumber < 100
                reflnrstring = ['00' num2str(reflnumber)];
            elseif reflnumber < 1000
                reflnrstring = ['0' num2str(reflnumber)];
            else
                reflnrstring = [num2str(reflnumber)];
            end
        end
        writestring = ['dlmwrite(''E:\4d_4e_Analysis\4d_'...
            'Intensity_Profiles\grain' reflnrstring '.txt'','...
            'Int_corr_save,''delimiter'',''\t'',''precision'','...
            '''%.2f'')'];
        eval(writestring);
        Int_total = sum(Int_corr1(:,2))+sum(Int_corr2(:,2))+...
            sum(Int_corr3(:,2));
        %Total grain intensity, corrected for layer overlap

        %Now compute the layer_CoM coordinate, using the refined
        %intensities as weights.
        layer_ref = sum((Int_corr1(:,1)+1.25).*Int_corr1(:,2))+...
                    sum((Int_corr2(:,1)+1.25).*Int_corr2(:,2))+...
                    sum((Int_corr3(:,1)+1.25).*Int_corr3(:,2));
        layer_ref = layer_ref/Int_total;
        dum1 = floor(layer_ref/30);
        dum2 = rem(layer_ref,30);
        layer_CoM = layer_start+5*dum1;
            %30 microns correspond to 5 layers
        while dum2 >= 15
            rem2 = rem(layer_CoM,5);
            layer_CoM = layer_CoM+1;
            if (rem2 == 0) || (rem2 == 2) || (rem2 == 4)
                dum2 = dum2-5;
            else
                dum2 = dum2-7.5;
            end
        end
        layer_CoM = layer_CoM+dum2/15;
        layer_CoM = layer_CoM-0.5;
            %So that an integer value for layer_CoM corresponds to
            %the middle of the layer instead of the beginning

        N(length_N+1,1)  = GR(ind1,1);
        N(length_N+1,2)  = stripe_CoM;
        N(length_N+1,3)  = layer_CoM;
        N(length_N+1,4)  = omega_CoM;
        N(length_N+1,5)  = GR(1,6);
        N(length_N+1,6)  = row_CoM;
        N(length_N+1,7)  = column_CoM;
        N(length_N+1,8)  = R_CoM;
        N(length_N+1,9)  = TwoTheta_CoM;
        N(length_N+1,10)  = Eta_CoM;
        N(length_N+1,11)  = NRofPixels;
        N(length_N+1,12)  = Int_total;

    end
```

```
        end

    end

end

M;
N;

%--------------------------------------------------------------------------
%End of: ArrangeSpots.m
%--------------------------------------------------------------------------
```

3.13. InputGrainSpotter.m

```
%-----------------------------------------------------------------------
%Start of: InputGrainSpotter.m
%-----------------------------------------------------------------------
%
%Program that converts the output from ArrangeSpots into the desired format
%for reading by GrainSpotter.
%
%TvdZ. Last edits: March 2007.

function gvecs = InputGrainSpotter(ReflList);

RowBC = 1038.52;
ColBC = 981.37;
pixel = 47.4;
Lsd = 241;
Lsd_pixels = Lsd*1000/pixel;
wavelength = 0.155;

gvecs = zeros(length(ReflList),8);

for n = 1:length(ReflList)

    Row_CoM = ReflList(n,6);
    Col_CoM = ReflList(n,7);
    omega = ReflList(n,4);
    eta = ReflList(n,10);
    etaRad = pi/180*eta;
    twotheta = ReflList(n,9);
    thetaRad = pi/180*0.5*twotheta;

    ds = 2*abs(sin(thetaRad))/wavelength;

    %Determination of the scattering vector G, using equation (3.6) from
    %the book by Henning Poulsen on 3DXRD theory
    gvecs(n,1) = -ds*cos(thetaRad)*tan(thetaRad);
    gvecs(n,2) = -ds*cos(thetaRad)*sin(etaRad);
    gvecs(n,3) = ds*cos(thetaRad)*cos(etaRad);

    gvecs(n,4) = Row_CoM;
    gvecs(n,5) = Col_CoM;
    gvecs(n,6) = ds;
    gvecs(n,7) = eta;
    gvecs(n,8) = omega;

end

%-----------------------------------------------------------------------
%End of: InputGrainSpotter.m
%-----------------------------------------------------------------------
```

3.14. CharacterizeGrains.m

```
%------------------------------------------------------------------------
%Start of: CharacterizeGrains.m
%------------------------------------------------------------------------
%
%Routine written to read the file containing the groups of reflections
%identified by GrainSpotter as coming from the same grain. The required
%input are the locations of both the output list created by GrainSpotter as
%well as the list of reflections created by the routine ArrangeSpots.
%
%TvdZ. Last edits: March 2007

function [TotalRefl,TotalExpected] = ...
    CharacterizeGrains(GrainSpotterList,ReflList);

offset = 11; %Number of lines in the header of GrainSpotterList

%Retrieve the number of identified grains, nrgrains
[str1 nrgrains str2] = textread(GrainSpotterList,'%s %u %s',1);
%Initialize matrix U, which will be filled with the orientation matrices of
%the individual grains
U = zeros(3*nrgrains,3);
TotalRefl = 0; %The total number of indexed reflections
TotalExpected = 0; %The total number of theoretically expected reflections

linenr = offset;
refllist = dlmread(ReflList,'\t');

for n = 1:nrgrains

    %Retrieve the number of expected (M0) and found (Mexp) reflections for
    %the grain in question
    [str1 grainnr] = textread(GrainSpotterList,'%s %u',1,'headerlines',...
        linenr);
    [M0 Mexp] = textread(GrainSpotterList,'%u %u',1,'headerlines',...
        linenr+1);

    GrainCharacteristics = zeros(Mexp+3,15);
    %This matrix will be filled with the grain's characteristics. The first
    %three elements of the first three rows will contain the orientation
    %matrix U. The rest of the rows will be filled with the individual
    %reflections.

    UGrain = textread(GrainSpotterList,'',3,'headerlines',linenr+2);
    GrainCharacteristics(1:3,1:3) = UGrain;
    GSMatrix = textread(GrainSpotterList,'',Mexp,'headerlines',linenr+5);
    %GSMatrix now contains GrainSpotter's output information for the grain
    %in question. For further grain characterization, refllist is required

    TotalRefl = TotalRefl+Mexp;
    TotalExpected = TotalExpected+M0;

    for refl = 1:Mexp

        reflnr = GSMatrix(refl,2)+1; %reflnr is the row number of the
            %reflection under consideration within refllist. The output
            %from GrainSpotter is zero-based, so 1 is added.
            %Here, the user can specify the required computations. As an
            %example, for each grain a file is created containing the
            %individual reflections' characteristics. GrainSpotter has already
            %created such a file, but the information contained in this file is
            %too limited for use in further analysis.

            reflection = refllist(reflnr,:);
            GrainCharacteristics(refl+3,1) = reflection(1,1);
            GrainCharacteristics(refl+3,2:4) = GSMatrix(refl,3:5);
            GrainCharacteristics(refl+3,5:15) = reflection(1,2:12);

        end

        if grainnr < 10
            grainnrstring = ['000' num2str(grainnr)];
        elseif grainnr < 100
            grainnrstring = ['00' num2str(grainnr)];
        elseif grainnr < 1000
            grainnrstring = ['0' num2str(grainnr)];
        else
            grainnrstring = [num2str(grainnr)];
        end
        writestring = ['dlmwrite(''E:\4d_4e_Analysis\4d_Grain_'...
                        'Reflections\grain' grainnrstring '.txt'','...
                        'GrainCharacteristics,''delimiter'',''\t'','...
                        '''precision'',''%.4f'')'];
        eval(writestring);

        linenr = linenr+Mexp+6;

    end

    %--------------------------------------------------------------------------
    %End of: CharacterizeGrains.m
    %--------------------------------------------------------------------------
```