EMsoft
Dictionary-Based Indexing

Program Manual, v3.0, March 30, 2017*

This manual is under development; several sections still need to be written. updated versions will be made available as soon as they are completed.

Table of Contents

*This set of programs was developed with financial support from two agencies. The Office of Naval Research sponsored the development of the original version 2.0 f90 source code for computation of EBSD patterns in research grant N00014-12-1-0075 and some of the early work in developing the dictionary indexing approach. The IDL visualization interface, and the complete version 3.1 (including ECP) and beyond were developed with financial support from an AFOSR/MURI grant FA9550-12-1-0458.

1 Introduction

This manual describes a suite of programs for the dictionary-based indexing of experimental electron backscatter diffraction patterns (EBSD) and electron channeling patterns (ECP): EMEBSDDI, EMECPDI, and EMDPFit. The output generated by these programs is formatted in the open source HDF5 standard, in the .ang and/or .ctf formats, and can be analyzed by the DREAM.3D package.

In this manual, we hope to accomplish the following tasks:

1. Explain briefly the underlying approach behind the dictionary indexing programs (section 2);
2. Document the input files for the f90 programs;
3. Document the program output;
4. Explain the use of these programs by means of a few basic examples.

2 How Does Dictionary-Based Indexing Work?

2.1 Introductory comments

Consider a hypothetical experiment in which a total of N_e measurements is carried out. Each measurement produces several (N_v) numerical values that can be stored together in a data vector, $\mathbf{v}$. We represent the j-th data vector as $\mathbf{v}_j$ ($1 \dots j \dots N_e$); the components of this vector are represented by $v_{i,j}$, where $1 \dots i \dots N_v$. In the EBSD or ECP case, the components will be individual diffracted intensities, but that is not important for this introductory section. In general, one should make sure that all vector components have approximately the same order of magnitude, which may require scaling one or more components, i.e., pre-processing the data.

We know from elementary math that for any pair of vectors, $\mathbf{p}$ and $\mathbf{q}$, we can define the dot-product as

$$\mathbf{p} \cdot \mathbf{q} = |\mathbf{p}|\,|\mathbf{q}|\,\cos(\theta_{\mathbf{pq}}), \tag{1}$$

where the vertical bars indicate the norm of the vector (we will only use the Euclidean norm in this document), and $\theta_{\mathbf{pq}}$ is the angle between the two vectors. If we normalize each vector, then we have

$$\hat{\mathbf{p}} \cdot \hat{\mathbf{q}} = \cos(\theta_{\hat{\mathbf{p}}\hat{\mathbf{q}}}), \tag{2}$$

where the hat indicates a unit length vector, i.e., $|\hat{\mathbf{p}}| = 1$.

If we consider two measurement points $\hat{\mathbf{v}}_j$ and $\hat{\mathbf{v}}_k$ that originate from neighboring points in the sample, then it is likely that the components of these two vectors will be very similar. As an example, consider the case $N_v = 4$, with the two data vectors given by:

$$\mathbf{v}_j = (23, 18, 5, 31) \rightarrow \hat{\mathbf{v}}_j = (0.53634, 0.41974, 0.11659, 0.72289);$$
$$\mathbf{v}_k = (24, 16, 6, 30) \rightarrow \hat{\mathbf{v}}_k = (0.57078, 0.38052, 0.14270, 0.71348).$$

The dot product between these two normalized vectors is equal to $\hat{\mathbf{v}}_j \cdot \hat{\mathbf{v}}_k = 0.99825$, so that the angle is given by $\theta_{jk} = 3.38^\circ$.

If a third measurement, $\mathbf{v}_m$, is carried out, away from the first two points, then we may obtain the values:

$$\mathbf{v}_m = (18, 21, 11, 25) \rightarrow \hat{\mathbf{v}}_m = (0.46306, 0.54024, 0.28298, 0.64314);$$

the resulting angles are then $\theta_{jm} = 13.34^\circ$ and $\theta_{km} = 14.27^\circ$, significantly larger than θ_{jk}. Since the vectors $\hat{\mathbf{v}}_j$ and $\hat{\mathbf{v}}_k$ have a small angle between them, we say that they represent "similar" measurements; the angle between $\hat{\mathbf{v}}_m$ and $\hat{\mathbf{v}}_j$ is much larger, so that the measurements are considered to be "dissimilar". Where exactly the threshold between similar and dissimilar lies depends on the nature of the data and on the criteria imposed by the user. Perhaps the criterion for similarity is 20° in which case all three measurements would be considered to be similar; on the other hand, if the threshold value is 1°, then all three measurements are dissimilar. Whatever the criterion may be, it is clear that the dot-product between normalized vectors can be used to group data points into clusters, and this is precisely what we need to index EBSD or ECP patterns.

In general, the data vectors $\mathbf{v}_j$ belong to a high-dimensional vector space, usually $\mathbb{R}^n$ with n some large integer. As an example, we will consider EBSD patterns of 60×60 pixels in the next section; for this case we would have $n = 3,600$. The high dimensionality of the space should not deter the user; the concepts of normalized vectors and dot-products are the same, regardless of the value of n. Similarity between high-dimensional vectors can thus be analyzed by means of the dot-product concept.

2.2 Things we can do with dot-products

The basic math behind the dictionary approach (at least, our implementation of it) is no more difficult that the dot-product analysis shown above. On the experimental side, we have a large number N_e of stored EBSD patterns; each pattern has $n_x \times n_y$ pixels in it, with intensity levels typically in the range $[0 \dots 255]$. We can convert each of these patterns into a vector $\mathbf{v}_j$, with $1 \dots j \dots N_e$; each vector has $n_x \times n_y$ elements, and we can easily normalize this vector as before. So, for EBSD patterns recorded on a 480×480 camera with $8\times$ binning, each normalized pattern can be represented by a vector $\hat{\mathbf{v}}_j$ with $60 \times 60 = 3,600$ components, and each component has a value in the interval $[-1, 1]$ such that $\sum_{i=1}^{3,600} v_{i,j}^2 = 1$.

We can already apply the dot-product concept just to the experimental data, and compute for each data point the average dot-product with its four nearest neighbors. Since an EBSD pattern changes upon crossing a grain boundary, the two vectors on either side of the boundary will in general have a larger angle between them than for points inside either grain, so that the dot product will be lower for pixels at or next to grain boundaries. A map of the average dot product will therefore show a lower intensity at all the grain boundaries.

> *Throughout this document we will use an experimental data set acquired by Stuart Wright (EDAX/TSL) on a Ni sample. The experimental data file contains* 10 *data sets of* $N_e = 186 \times 151 = 28,086$ *EBSD patterns each for nominally the same region of interest on the sample. Each pattern was acquired on a camera with* 480×480 *pixels and* $8\times$ *binning, resulting in patterns of* 60×60 *pixels or, equivalently, data vectors* $\mathbf{v}_j$ $(1 \dots j \dots N_e)$ *with* $N_v = 3,600$ *components each. The data file also contains the Euler angle triplets, confidence index, image quality, etc. for each pattern, as determined by the EDAX software (using the Hough transform indexing approach). The microscope was operated at* 20 *kV and the sample was inclined at an angle of* $\sigma = 75.7^\circ$ *with respect to horizontal. Most of the examples in this document will make use of one of the* 10 *data sets in this file; the data sets differ only in the camera settings. We will refer to a data set as* Scan 1, Scan 2, *etc.*

Using Scan 1, we compute for each pixel of the region of interest (ROI) the average dot product with its 4 nearest neighbors (fewer for pixels at the edges or corners). This requires normalizing all patterns to the interval $[0, 1]$, followed by the dot product averaging process. The resulting average dot product map is shown in Fig. 1(a); it is clear that at grain boundaries the value of the average neighbor dot product decreases slightly.

A histogram of the dot products is shown in Fig. 1(b); note that the main peak has a shoulder to the left of the maximum, corresponding to the grain boundary pixels. When the average intensity is subtracted from the individual EBSD patterns before normalization, then the histogram becomes stretched out horizontally (note the different horizontal scales in Fig. 1(b) and (c)); the dot product map, however, looks basically the same since the map is obtained by putting the smallest dot product value at intensity level 0, and the highest at intensity level 255. Other than a rescaling of the histogram, nothing new is obtained by subtracting the average pattern from each EBSD pattern. In principle, one could use this dot product map to obtain a segmentation of the microstructure.

2.3 Uniform Sampling of Orientation Space

In the dictionary indexing approach, we compute dot products between experimental patterns and simulated patterns. Using the forward model described in the EMEBSD manual, simulated patterns are obtained for the experimental detector geometry (the parameters for this geometry are

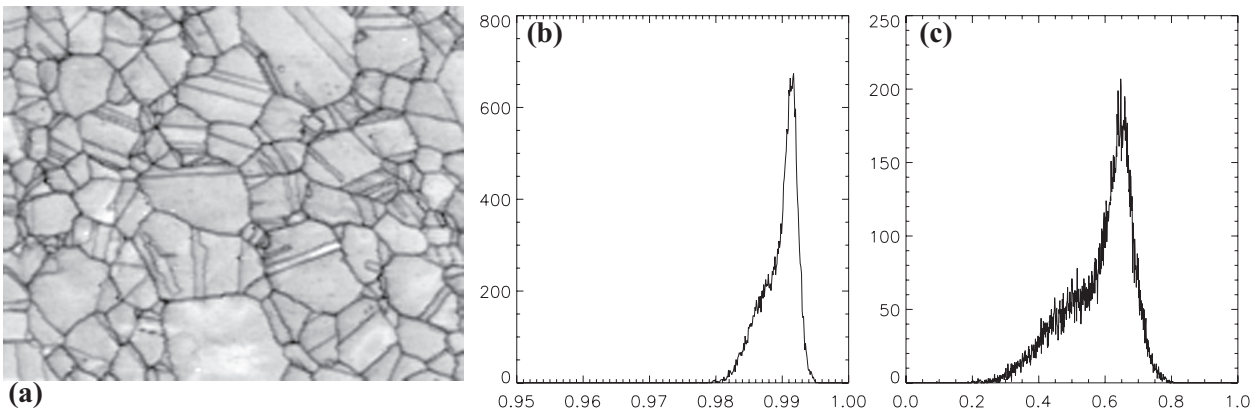

Figure 1: *(a) average neighbor dot product map for Scan 1 of the Ni data set; (b) and (c) show the dot product histograms without and with average pattern subtraction, respectively. Note the different horizontal scales.*

determined using the procedure described in section 3). Those patterns are then compared with the experimental patterns using dot products. The simulated patterns are computed for a series of crystal orientations; these orientations must be generated by uniformly sampling orientation space, or the portion of orientation space that represents unique orientations for a given crystal structure, i.e., the fundamental zone (FZ). To facilitate the creation of a uniform sample of orientations we make use of the so-called *cubochoric* rotation representation, first derived in [1]. Without going into too much detail, here is how the uniform sample is obtained:

1. Determine the point group for the crystal structure of the sample and convert it to a rotational point group.

2. Select a desired angular sampling step size $\langle\theta\rangle$ (e.g., 1.5°) and convert this to an integer number of sampling steps N using the following relation (for details, see [2]):

$$N = \text{nint}\left[\frac{131.97049}{\langle\theta\rangle - 0.03732}\right]. \tag{3}$$

3. Generate a uniform 3D cubic grid with $(2N+1)^3$ grid points inside a cube with edge length $\pi^{2/3}$.

4. For each grid point, convert the coordinates $(x, y, c)_c$ to a Rodrigues vector, using the relations described in [3].

5. Determine whether or not the Rodrigues vector lies inside the fundamental zone for the rotational point group; if the Rodrigues vector lies inside the RFZ, keep the original grid point, otherwise discard it.

6. Repeat the last two steps for all $(2N+1)^3$ grid points; this will result in a set of N_s sampling points that correspond to a uniform sampling of orientations inside the RFZ.

7. Compute a simulated EBSD pattern for all the orientations corresponding to the surviving grid points in the cubochoric cube.

The execution time for the dictionary indexing programs is directly proportional to the total number of surviving grid points. For a desired angular step size (average disorientation between

neighboring sampling points in orientation space) of $\langle\theta\rangle = 1.5^\circ$, we have $N = 90$, for which a total of $181^3 = 5,929,741$ grid points will be generated in the cubochoric cube. For cubic symmetry (the octahedral rotation group), approximately $1/24^{th}$ of this number lie inside the RFZ, or about $247,000$ sampling points. This means that even at the relatively coarse step size of 1.5°, several hundreds of thousands of dictionary patterns need to be generated. Each of those patterns must then be compared to each of the experimental patterns, as described in the next section. It should be clear to the reader that a dictionary indexing run can take a while, depending on the crystal symmetry and the sampling step size required. This extended computation time is offset by the increase in accuracy of the resulting orientation map; the dictionary indexing approach is known to be robust against pattern noise, so that very noisy EBSPs, for which the Hough-transform based indexing methods fail, can still be successfully indexed.

2.4 The Dictionary Indexing Approach

Consider an experimental data set with N_e EBSD patterns represented by normalized vectors $\hat{\mathbf{v}}_j$. Consider also a computed dictionary of N_d patterns (i.e., the number of surviving grid points inside the RFZ) represented by the normalized vectors $\hat{\mathbf{w}}_k$ $(1 \dots k \dots N_d)$. As we have seen in the previous section, N_d can be of order 10^5 or even larger for lower order crystal symmetries; similarly, an experimental data set can consist of many tens of thousands of patterns, depending on the scan area and step size.

The dictionary indexing approach is then simply based on the computation, for each experimental pattern $\hat{\mathbf{v}}_j$, of a vector $\mathbf{d}_j$ of length N_d of all possible dot products:

$$\mathbf{d}_j = (\hat{\mathbf{v}}_j \cdot \hat{\mathbf{w}}_1, \hat{\mathbf{v}}_j \cdot \hat{\mathbf{w}}_2, \dots, \hat{\mathbf{v}}_j \cdot \hat{\mathbf{w}}_k, \dots, \hat{\mathbf{v}}_j \cdot \hat{\mathbf{w}}_{N_d}) \,.$$

Once this vector has been computed, its elements are sorted in decreasing order, and the top N_m values are kept, along with a list of the original index numbers k corresponding to each dot product in the ordered list. Since the indices k $(1 \dots k \dots N_d)$ correspond to orientations, this ranking of the dot products immediately associates an orientation with each of the top N_m matches. Typically, this orientation is expressed as an Euler angle triplet, but any orientation representation can be used.

Since the computation of millions of dot products between potentially large normalized vectors can be time consuming, our implementation assigns the dot product computation to a Graphical Processing Unit (GPU). The standard indexing approach computes the dictionary patterns on-the-fly, using multiple cores (OpenMP), and does not store them on disk. The procedure has the following steps:

1. Initialize the set of N_s sampling orientations using the approach of the previous section.

2. Initialize N_o OpenMP threads $(0 \dots \tau \dots N_o - 1)$ where τ is the thread ID. Thread $\tau = 0$ is in charge of all communications with the GPU, whereas all other threads $\tau = 1 \dots N_o - 1$ are assigned to the computation of dictionary patterns.

3. Since it is not possible to send all $N_e \times N_d$ vectors to the GPU in one computational step, the user defines a chunk size for experimental and dictionary patterns; for reasons of compute efficiency, the chunk sizes must be multiples of 16. So, for instance, for the Ni data set used in this manual, there are $N_e = 28,086$ experimental patterns, and for a cubochoric sampling index of $N = 100$ there are $N_d = 333,227$ simulated patterns. If we define the chunk size to be $N_c = 512$ for both experimental and dictionary patterns, then the GPU will compute dot-products for blocks of N_c experimental and N_c dictionary patterns. The threads $\tau = 1 \dots N_o - 1$ will share the computation of those N_c dictionary patterns.

4. Thread $\tau = 0$ then takes these N_c dictionary patterns and sends them to the GPU, along with blocks of N_c experimental patterns. The GPU computes all the dot-products and returns them to thread $\tau = 0$, which will rank them and keep the top k dot product values for each experimental pattern. This is repeated until there are no further blocks of experimental patterns to be handled; at this point, thread $\tau = 0$ joins the other threads in the computation of the next block of dictionary patterns. This process can be made efficient by experimenting with the chunk size; this size should be set so that the computation of the dictionary patterns takes slightly longer than the time needed to compute the dot products of one dictionary block against the complete experimental data set.

At the end of the computation, thread $\tau = 0$ has a ranked list of the top k dot product values and associated orientations for each of the experimental patterns. These are then written to an HDF5 output file that is organized in the same way as the experimental EDAX/TSL .h5ebsd files, so that the indexing results can be viewed using standard software packages, including DREAM.3D. The user also has the option of creating a standard .ang or .ctf file.

2.5 Example Input

Consider Scan 1 from the Ni .h5ebsd data file. There are $N_e = 181 \times 156 = 28,086$ experimental patterns of 60×60 pixels each in this data set, along with the Euler angles, confidence index (CI), image quality (IQ), and related data, as determined by the EDAX/TSL acquisition program. The patterns were acquired at 20 kV, with the sample tilted by $\sigma = 75.7°$. The detector has 480×480 pixels and was binned by a factor of $8\times$. Each detector pixel has an edge length of 59.2 μm on the scintillator surface. A fit routine, described in section 3 was used to determine the optimal pattern center and scintillator-to-sample distance.

Using the approach described in the EMEBSD manual, a Monte Carlo simulation was carried out (EMMCOpenCL) for Ni and the experimental sample tilt angle σ. This was followed by a master pattern computation (EMEBSDmaster) for each of the 31 energy levels from 5 to 20 keV with a 0.5 keV step size. The Monte Carlo results have a grid size of $501 \times 501 \times 31$ pixels for the energy histograms, and $51 \times 51 \times 101 \times 31$ for the depth histograms, with a total of 101 depth steps from 0 to 100 nm. The master pattern was computed on a grid of $31 \times 1001 \times 1001$ points, with one array for the Northern hemisphere and one for the Southern; due to the lattice symmetry, those arrays are equal. The results from these two program runs are available to the user as example files (see section 5).

As is the case for all other EMsoft programs, the EMEBSDDI program expects a namelist (or json) input file, defining the following variables:[1]

```
 &EBSDIndexingdata
! The line above must not be changed
!
! The values below are the default values for this program
!
!#####################################################################
! INDEXING MODE
!#####################################################################
!
! 'dynamic' for on the fly indexing or 'static' for pre calculated dictionary
 indexingmode = 'dynamic',
```

[1]Note the space in front of each entry and the comma at the end of each non-comment line; these are present so that the forthcoming Windows version, compiled with the *ifort* compiler, will function properly.

```
!
!###################################################################
! DICTIONARY PARAMETERS: COMMON TO 'STATIC' AND 'DYNAMIC'
!###################################################################
!
! height of inverse pole figure in pixels
 ipf_ht = 100,
! width of inverse pole figure in pixels
 ipf_wd = 100,
! X and Y sampling step sizes
 stepX = 1.0,
 stepY = 1.0,
! number of top matches to keep from the dot product results
 nnk = 50,
! number of top matches to use for orientation averaging
 nnav =  20,
! number of top matches to use for Orientation Similarity Map computation
 nosm = 20,
! mask or not
 maskpattern = 'n',
! mask radius
 maskradius = 240,
! hi pass filter w parameter; 0.05 is a reasonable value
 hipassw = 0.05,
! number of regions for adaptive histogram equalization
 nregions = 10,

!###################################################################
! ONLY SPECIFY WHEN INDEXINGMODE IS 'DYNAMIC'
!###################################################################
!
! number of cubochoric points to generate list of orientations
 ncubochoric = 100,
! distance between scintillator and illumination point [microns]
 L = 15000.0,
! tilt angle of the camera (positive below horizontal, [degrees])
 thetac = 10.0,
! CCD pixel size on the scintillator surface [microns]
 delta = 50.0,
! number of CCD pixels along x and y
 numsx = 640,
 numsy = 480,
! pattern center coordinates in units of pixels
 xpc = 0.0,
 ypc = 0.0,
! angle between normal of sample and detector
 omega = 0.0,
! minimum and maximum energy to use for interpolation [keV]
 energymin = 10.0,
 energymax = 20.0,
! energy averaging method (0 for exact, 1 for approximate)
 energyaverage = 0,
! spatial averaging method ('y' or 'n' ;can't be used with approximate energy average)
 spatialaverage = 'n',
! incident beam current [nA]
 beamcurrent = 150.0,
! beam dwell time [micro s]
 dwelltime = 100.0,
! binning mode (1, 2, 4, or 8)
```

```
 binning = 1,
! intensity scaling mode 'not' = no scaling, 'lin' = linear, 'gam' = gamma correction
 scalingmode = 'not',
! gamma correction factor
 gammavalue = 1.0,
!
!#####################################################################
! FILE PARAMETERS: COMMON TO 'STATIC' AND 'DYNAMIC'
!#####################################################################
!
! name of datafile where the patterns are stored; path relative to EMdatapathname
 exptfile = 'undefined',
! temporary data storage file name ; will be stored in $HOME/.config/EMsoft/tmp
 tmpfile = 'EMEBSDDict_tmp.data',
! output file ; path relative to EMdatapathname
 datafile = 'undefined',
! ctf output file ; path relative to EMdatapathname
 ctffile = 'undefined',
! average ctf output file ; path relative to EMdatapathname
 avctffile = 'undefined',
! ang output file ; path relative to EMdatapathname [NOT AVAILABLE UNTIL RELEASE 3.2!!!]
! angfile = 'undefined',
! euler angle input file
 eulerfile = 'undefined'

!#####################################################################
! ONLY IF INDEXINGMODE IS STATIC
!#####################################################################
!
 dictfile = 'undefined',
!
!#####################################################################
! ONLY IF INDEXINGMODE IS DYNAMIC
!#####################################################################
!
! master pattern input file; path relative to EMdatapathname
 masterfile = 'undefined',
!
!#####################################################################
! SYSTEM PARAMETERS: COMMON TO 'STATIC' AND 'DYNAMIC'
!#####################################################################
!
! number of dictionary files arranged in column for dot product on GPU (multiples of 16 perform better)
 numdictsingle = 1024,
! number of experimental files arranged in column for dot product on GPU (multiples of 16 perform better)
 numexptsingle = 1024,
! number of threads for parallel execution
 nthreads = 1,
! platform and device IDs for OpenCL portion of program
 platid = 1
 devid = 1
 /
```

Here are the meanings of all variables in the order they appear above; note however, that the order is not important. The values in the list above are the default values for all variables.

- indexingmode:: use dynamic for the on-the-fly indexing, and static to use a pre-calculated dictionary.

- ipf_ht: Number of vertical pixels in the ROI for the input data.
- ipf_wd: Number of horizontal pixels in the ROI for the input data.
- stepX, stepY:: sampling step sizes (note that, for now, only the square sampling grid has been implemented).
- nnk: Number of dictionary matches (dot products) to keep at the end of the indexing run. This number should be set to the value at which a plot of dot-product vs. matching rank levels out.
- maskpattern: Should a circular mask be used on the simulated patterns? If the experimental patterns are circular, then the use of a pattern mask is advised and this parameter should be set to y.
- maskradius: Radius of the pattern mask, in pixels; can be less than half the smallest pattern dimension; the mask is applied *after* the binning operation, so the mask radius should refer to the binned pattern size, not the original pattern size.
- nregions: This parameter defines how many subregions the EBSD patterns should be divided in for the application of the adaptive histogram equalization filter that is used in the pattern pre-processing step. This equalization combined with high-pass Fourier filtering converts both experimental and simulated pattern to a more or less common intensity scale, leading to improved indexing results.
- ncubochoric: This is the number N used to construct the uniform sample in cubochoric space, as described in section 2.3.
- L: Distance between scintillator and illuminated point on the sample, in microns; can be determined using the fitting program in section 3.
- thetac: Angle of the detector with respect to the horizontal plane, in degrees; a detector that looks up towards the sample has a positive angle.
- delta: Pixel edge size on the scintillator, in microns; we assume that each pixel is square. Note that this is *not* the CCD pixel size; all EBSD patterns are computed on the surface of the scintillator, not on the CCD camera, since the details of cameras vary from one vendor to the other, but the scintillator is common to all systems. Some cameras will have a fiber-optic bundle that channels the signal from the scintillator to the CCD, others have an optical lens. In all cases, the pixel size on the scintillator is likely larger than that of a single pixel on the CCD camera. This parameter can be determined by measuring the illuminated portion of the scintillator that is mapped onto the CCD camera, and dividing by the number of CCD pixels; this requires an independent calibration, since this parameter is fixed in the fitting routine.
- numsx, numsy: Number of x (horizontal) and y (vertical) pixels on the CCD camera, excluding binning.
- xpc, ypc: x and y coordinates of the pattern center on the scintillator surface, measured in units of scintillator pixels (delta).
- omega: Rotation angle of the sample around the RD axis (degrees); this value should be very close to zero for a well-aligned sample. Non-zero values will cause significant distortions of the EBSPs.

- energymin, energymax: Minimum and maximum electron energy to take into account in the EBSD pattern interpolation; normally, these values should be equal to the min and max energy values used in the Monte Carlo program run (EMMCOpenCL).
- energyaverage [to be removed in later EMsoft version]: Use 0 for an exact energy summation, 1 for an approximate summation (not very accurate, used originally in an attempt to speed up the program).
- spatialaverage [to be removed in later EMsoft version]: only the value 'n' is currently supported; used during debugging phase of program.
- beamcurrent: Incident beam current, in nA. In the absence of Poisson noise, any non-zero value is acceptable.
- dwelltime: Exposure dwell time in μs. In the absence of Poisson noise, any non-zero value is acceptable.
- binning: Binning factor; currently only the values 1, 2, 4, and 8 are supported.
- scalingmode: Intensity scaling mode, used to convert the computed intensities to a linear or non-linear (gamma) scale; for the dictionary program, this value should be set to gam.
- gammavalue: Gamma value for intensity scaling; values in the range $[0.3 \ldots 0.5]$ are reasonable. This value can be determined using the fitting program.
- exptfile: Name of the experimental pattern input file (see section **??**); note that this should be a relative path with respect to the EMdatapathname variable.
- tmpfile: Name of a temporary file that will be stored in the $HOME/.config/EMsoft/tmp folder; this file contains all the processed experimental patterns and can become very large. At the end of the run, this file is automatically deleted.
- datafile: Name of the EDAX/TSL HDF5 output file; note that this should be a relative path with respect to the EMdatapathname variable.
- ctffile: Name of a .ctf output file (Oxford text format).
- angfile: Name of a .ang output file (EDAX/TSL text format).
- dotproductfile: optional creation of an HDF5 file that contains all the dot products (can be a very large file!).
- dictfile: optional creation of an HDF5 file that contains all the dot products (can be a very large file!), only in static indexing mode.
- energyfile: Name of the Monte Carlo input file; note that this should be a relative path with respect to the EMdatapathname variable.
- masterfile: Name of the master pattern input file; note that this should be a relative path with respect to the EMdatapathname variable.
- numdictsingle: Chunk size for the dictionary patterns; should be a multiple of 16. For a system with a high end GPU, this value can be taken to be 512 or 1024, on a laptop perhaps 64 or 128 are more reasonable. Check the output of the EMOpenCLinfo program for an indication of reasonable values.

- numexptsingle: Chunk size for the experimental patterns; should be a multiple of 16, and it would make sense to take the same value as for numdictsingle.
- nthreads: Number of OpenMP threads to use for the dictionary pattern computation. Note that thread 0 will take care of communications with the GPU.
- platid: OpenCL platform ID; see output of EMOpenCLinfo program.
- devid: OpenCL device ID; see output of EMOpenCLinfo program.

The last series of variables deserves a bit more explanation. The dictionary indexing algorithm relies on the computation of dot products between large arrays of pattern vectors. The dimensions of these arrays will be set to the values of numdictsingle and numexptsingle for one dimension, and the number of pixels in each pattern for the other dimension. For reasons of GPU efficiency, the number of patterns in each array is taken to be a multiple of 16; the arrays will be padded with zeroes if the total number of experimental and/or dictionary patterns is not a multiple of 16. The number of threads must be smaller than or equal to the available number of CPU cores on the system. Similarly, in the program output file, all the large arrays will be padded with zeroes to have a dimension that is a multiple of the numdictsingle parameter.

2.6 OpenCL platforms and devices

The platform(s) and device(s) available on a given system can be listed with the EMOpenCLinfo program. The following is example output from the EMOpenCLinfo program for a 2015 Mac Pro (also known as the "trashcan"):

```
 Number of Platforms: 1
--------
 Platform: 1

  Profile: FULL_PROFILE
  Version: OpenCL 1.2 (May 10 2015 19:38:45)
  Name: Apple
  Vendor: Apple
  Extensions: cl_APPLE_SetMemObjectDestructor cl_APPLE_ContextLoggingFunctions
  cl_APPLE_clut cl_APPLE_query_kernel_names cl_APPLE_gl_sharing cl_khr_gl_event

 Num CPU Devices:  1
  Device (# 1, CU/MWGS/MWIS/GMS:   12/1024/1024,   1,   1/ 64) -  Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz

 Num GPU Devices:  2
  Device (# 1, CU/MWGS/MWIS/GMS:   32/ 256/ 256, 256, 256/  6) -  AMD Radeon HD - FirePro D700 Compute Engine
  Device (# 2, CU/MWGS/MWIS/GMS:   32/ 256/ 256, 256, 256/  6) -  AMD Radeon HD - FirePro D700 Compute Engine

  [CU = Compute Units; MWGS = Maximum Work Group Size; MWIS = Maximum Work Item Sizes (3D);
  GMS = Global Memory Size (Gb)]
--------
```

In terms of the dictionary indexing program, the most important information is listed at the end of this output, namely the specifics of each device. On this compute platform (of the type Apple; there is only one compute platform on this iMac), there are three devices. The first device is the regular Intel Xeon CPU; the other devices, with IDs 1 and 2, are AMD Radeon GPUs. The GPU has a device ID (1 or 2), so that is the value you should enter for the devid parameter in the name list file. The abbreviations CU/MWGS/MWIS/GMS stand for Compute Units (also known as

"cores"), Maximum Work Group Size, Maximum Work Item Sizes (3 values), and Global Memory Size (in Gb); the corresponding numbers are then listed, also separated by / symbols. For the CPU we have a total 12 compute units (cores), so this would be the maximum value for the nthreads parameter. Note that picking the maximum value may make your computer system unresponsive for the duration of the program run. The other parameters are not as important for the CPU.

For the GPUs there are 32 compute units, the maximum work group size is 256 (meaning that a work group can have no more than 256 work items), and the maximum work item sizes are 256 each; these numbers have to do with the internal organization of the computations on the GPU. In the current EMEBSDDI implementation, the GPU is considered to be a 2D compute device; in other words, the work items are organized on a 2D grid, and the maximum number of work items along each dimension is 256. Note that this particular graphics card is a medium to high level card.

The following is a second example from the EMOpenCLinfo program, this time for a high-end Linux box with several dedicated GPUs:

```
 Number of Platforms: 2
--------
 Platform: 1

  Profile: FULL_PROFILE
  Version: OpenCL 1.1 CUDA 7.0.65
  Name: NVIDIA CUDA
  Vendor: NVIDIA Corporation
  Extensions: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options
  cl_nv_device_attribute_query cl_nv_pragma_unroll cl_nv_copy_opts

 No CPU devices found:           -1

 Num GPU Devices:  8
  Device (# 1, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  Device (# 2, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  Device (# 3, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  Device (# 4, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  Device (# 5, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  Device (# 6, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  Device (# 7, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  Device (# 8, CU/MWGS/MWIS/GMS:   13/1024/1024,1024,  64/ 11) -  Tesla K80
  --------
 Platform: 2

  Profile: FULL_PROFILE
  Version: OpenCL 1.2 LINUX
  Name: Intel(R) OpenCL
  Vendor: Intel(R) Corporation
  Extensions: cl_khr_icd cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics
  cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store
  cl_khr_depth_images cl_khr_3d_image_writes cl_intel_exec_by_local_thread cl_khr_spir cl_khr_fp64

 Num CPU Devices:  1
  Device (# 1, CU/MWGS/MWIS/GMS:   48/8192/8192,8192,8192/125) -  Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz

 No GPU devices found:           -1

  [CU = Compute Units; MWGS = Maximum Work Group Size; MWIS = Maximum Work Item Sizes (3D);
  GMS = Global Memory Size (Gb)]
--------
```

This system has 8 Tesla K80 GPUs, each with 13 compute units, and each card can handle a maximum work group size of 1024 work items. Each card has 11 Gb of global memory. In this case, the values of numdictsingle and numexptsingle should be set to higher values, to make better use of the available resources. In the current implementation of EMEBSDDI, only one GPU is used at any moment in time; this may change in a future version. The number of available CPU compute units on this system is 48; since the CPU manufacturer (Intel) is different from the GPU manufacturer (nVidia), the CPU belongs to a different platform than the GPUs.

3 Fitting the Detector Parameters

The dictionary indexing approach relies on the availability of an accurate set of detector parameters (in the same way that the standard Hough transform approach also needs accurate determination of the pattern center and scintillator-to-sample distance). Determination of these parameters is a bit of a bootstrapping process, since it is necessary to know the orientation (Euler angles) for the pattern to be used in the detector parameter fit, and knowing this orientation requires knowledge of the detector parameters.

The user should make a habit of recording a high quality, maximum size, EBSD pattern as part of the data acquisition run. For instance, for a camera with 640×480 pixels, a single long exposure of a well defined full size EBSP (i.e., 640×480, no binning, with clearly defined Kikuchi bands) should be recorded and stored along with the data set (which can consist of smaller binned patterns). The standard Hough transform should be used to obtain the orientation $(\varphi_1, \Phi, \varphi_2)$ for this pattern. The user should make note of the pattern center coordinates and the scintillator-to-sample distance (SSD) obtained via the system calibration routines (manual or iterative). In addition, knowledge of the detector pixel size on the scintillator surface is needed. Finally, the detector tilt angle should be recorded as well. This full-size reference pattern will be used to obtain optimized detector parameters that can be used as input parameters for the dictionary indexing program.

3.1 A few important notes about our detector geometry

In the EMsoft package, the detector geometry is represented by the pattern center coordinates (x_{pc}, y_{pc}) and the SSD L. The internal convention for the PC coordinates is that they are expressed in units of the scintillator pixel size,[2] δ, with respect to the center of the scintillator (i.e., the center pixel in the EBSP). This definition is different from that used by the EBSD vendors, so it is useful to consider how each of the vendors defines the detector geometry in the following sections. Here, we define the internal EMsoft detector geometry. The detector, which coincides with the plane of the scintillator, *not* the CCD camera, is a rectangular array of $N_x \times N_y$ pixels of edge length δ (in microns). The origin is taken to be at the center of the detector, as illustrated in Fig. 2(a). The detector coordinate axes are chosen as follows: looking at the detector from the sample, the $\mathbf{e}_x^d$ axis points horizontally to the right, the $\mathbf{e}_y^d$ axis points upward, and the $\mathbf{e}_z^d$ axis completes the right-handed detector reference frame (and points towards the sample). The detector plane may be tilted away from the vertical plane by a detector tilt angle θ_c, which is taken to be positive when $\mathbf{e}_z^d$ points above the horizontal plane.

The sample is characterized by the $\mathbf{e}_i^s$ Cartesian reference frame, which is typically labeled as $\mathbf{e}_x^s \parallel RD$, $\mathbf{e}_y^s \parallel TD$,and $\mathbf{e}_z^s \parallel ND$, as illustrated in Fig. 2(b).

While the angle θ_c is typically fixed by the detector manufacturer, the value entered by the user may be different from this fixed value if the sample inclination is slightly off the value set for the

[2] We assume that the pixels are square.

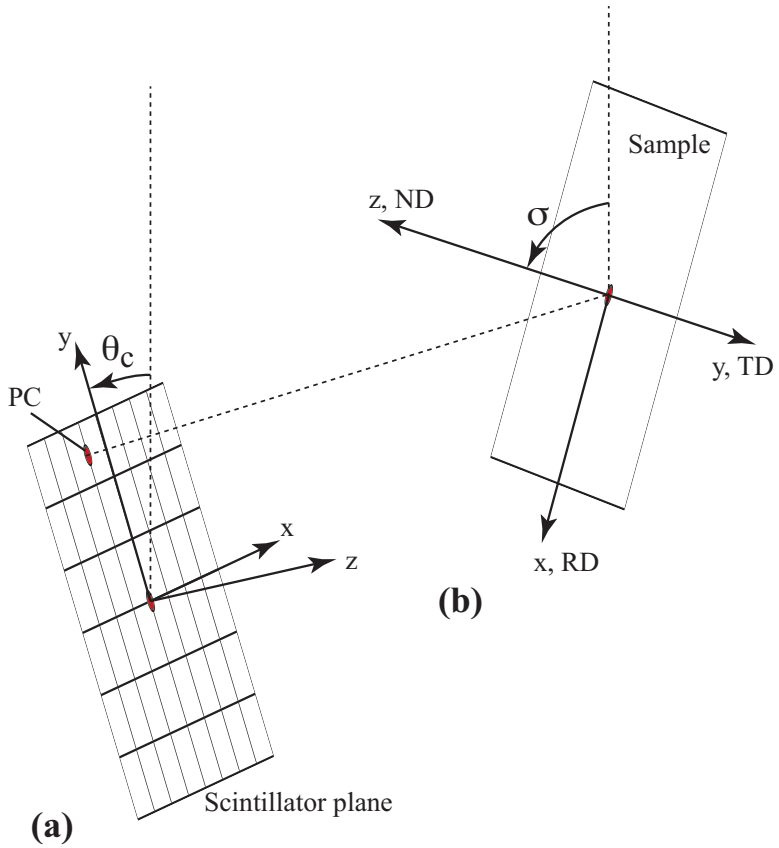

Figure 2: *Schematic illustration of a scintillator (a) with* 8×6 *pixels, inclined by* θ_c *around the negative* $\mathbf{e}_x^d$ *axis; the pattern center PC has coordinates* $(-1, 2.5)$*. In (b), the sample reference frame is illustrated. Note that both z axes lie in the same plane as the two origins.*

Monte Carlo simulation. Consider a Monte Carlo simulation carried out for a sample inclination angle of $\sigma = 70°$, measured clockwise around the sample's TD axis (see Fig. 2(b)). If the true sample tilt angle deviates slightly from σ by an amount $\Delta\sigma$ (positive for counterclockwise rotation around TD), then this can be taken into account in the indexing program by setting the detector tilt angle to $\theta_c + \Delta\sigma$ and using the $\sigma = 70°$ Monte Carlo simulation. Note that the sign of the detector tilt angle θ_c is taken for the same view point, i.e., along the TD axis, which makes θ_c *clockwise* with respect to $\mathbf{e}_x^d$. The pattern center (PC in Fig. 2(a)) coordinates are expressed in units of the pixel size δ, so that the coordinates for the simple 8×6 grid drawn on the scintillator surface are $(-1, 2.5)$. Finally, the sample may also have a small rotation by an angle ω around the RD axis (not shown in the figure).

3.2 Fitting the detector parameters

It is crucial to employ optimal detector parameters for the dictionary indexing approach. In our experience, this is best achieved by a multi-step process. First, as already stated before, the user should acquire a full resolution high quality EBSP with the sample in the position and orientation subsequently used for the full data acquisition run (which can be performed with binned patterns). This full resolution pattern will be used to determine the optimal detector parameters. Then, the IDL Efit user interface can be used to obtain a set of near-match detector parameters; sometimes those will be sufficiently accurate to initiate a dictionary indexing run, but often additional refinement is

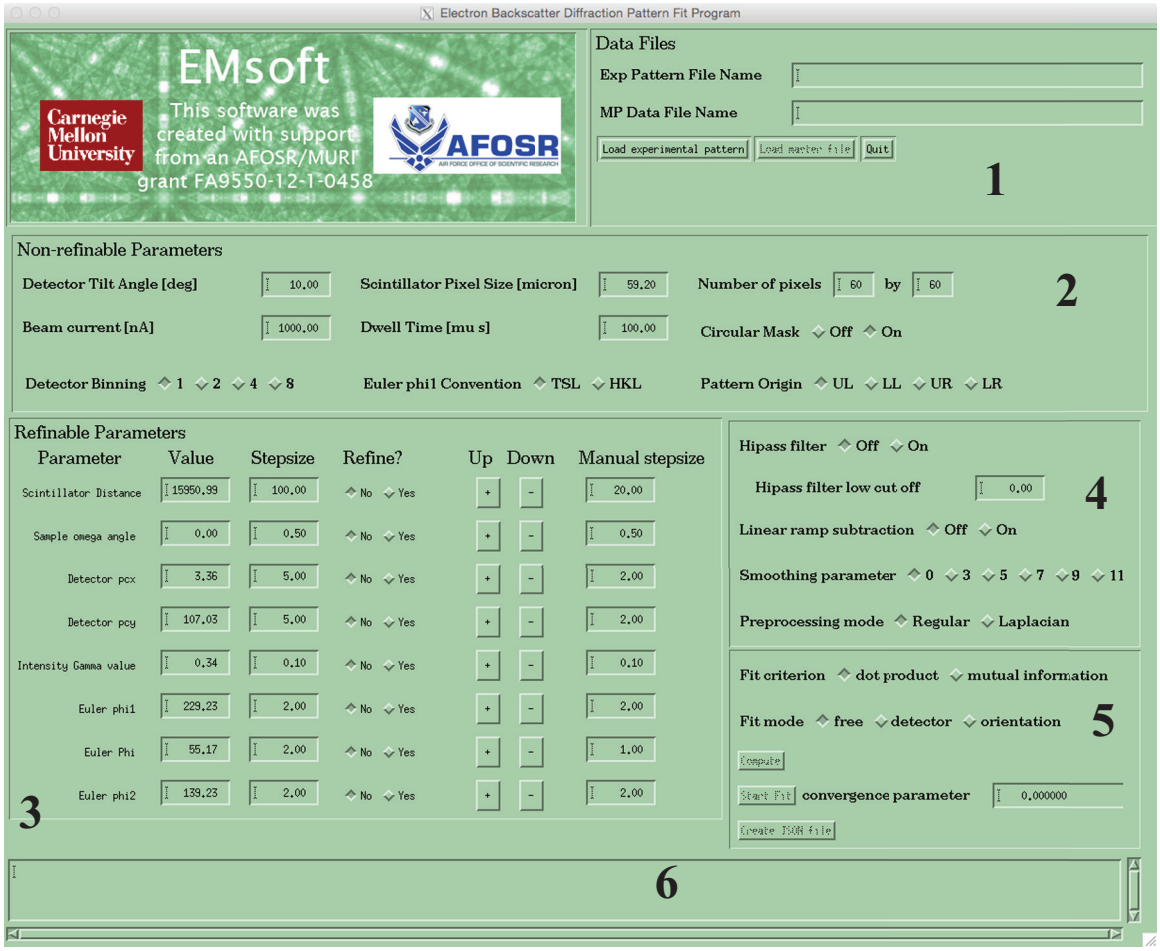

Figure 3: *Screen shot of the main Efit window; the various sections are explained in the text.*

needed using the EMDPfit program. Both programs are described in the following sections.

3.2.1 Using the interactive IDL Efit virtual app

The interactive Efit IDL virtual app is made available as a convenient tool to assist in determining the optimal detector parameters to be used for a dictionary indexing run. The app can be started by double clicking on its icon; since the size of the widget display is rather large, this app should be executed on a sufficiently large screen, probably not on a laptop. Fig. 3 shows the layout of the main program window.

The user should start in Section **1** by pressing the Load Experimental Pattern button. This will bring up a file selection window which should be used to navigate to the folder containing the experimental pattern to be used for the parameter fit. Recall that this should ideally be a pattern using the full detector size, not a binned pattern. Once this pattern has been loaded, the Number of pixels fields in Section **2** will display the pattern size; these fields should not be changed. Once the pattern has been loaded into memory, the Load master file button in Section **1** will be activated; the user should press this button to bring up a file navigation window to load the master pattern file to be used for the pattern simulation. Note that this file should also contain the correct path to the associated Monte Carlo file. Once both files have been loaded, the program will be ready to begin the parameter refinement. Informational messages will appear in Section **6** at the bottom of the

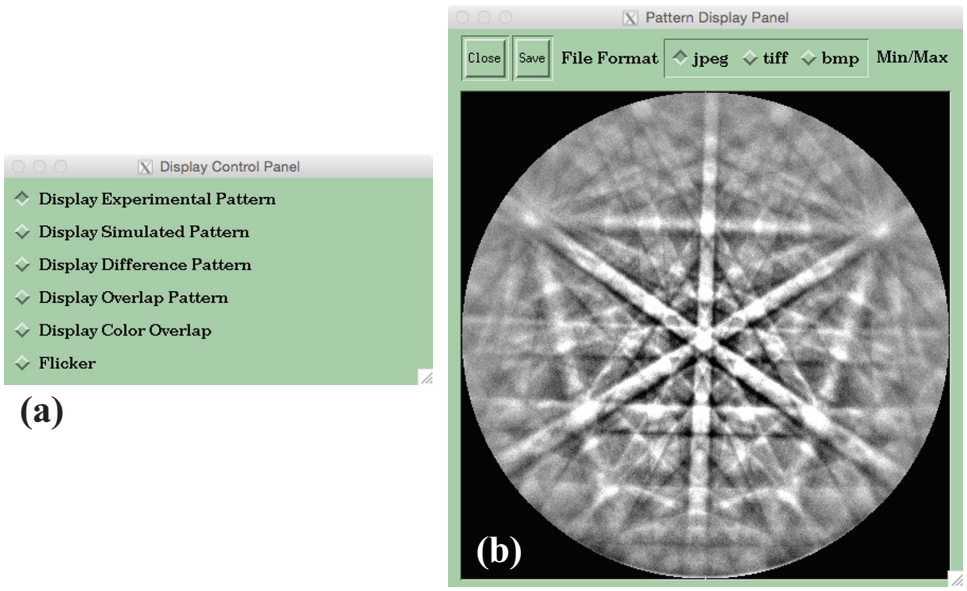

Figure 4: *Screen shot of (a) the Display Control Panel and (b) the Pattern Display Panel.*

interface.

In Section **2**, the user should enter the correct non-refinable parameters:

- *Detector tilt angle [deg]*: This is the angle θ_c ($+\Delta\sigma$) defined in section 3.1.
- *Scintillator pixel size [microns]*: This is the parameter δ, the pixel size on the scintillator surface.
- *Beam Current [nA]*: beam current in nA; any strictly positive value will do.
- *Dwell Time [mu s]*: beam dwell time in micro-seconds; any strictly positive value will do.
- *Circular Mask*: turn the circular mask on or off.
- *Detector binning*: This should be kept at 1 at all times; other values will be implemented in a later version.
- *Euler phi1 convention*: Select whether the Euler angles use the TSL or HKL convention for φ_1.
- *Pattern Origin*: Define where the pattern origin (according to the manufacturer) lies: upper left, upper right, lower left, lower right. This will cause all patterns to be flipped in orientation.

Once the parameters in Section **2** have been set, the Compute button in Section **5** can be pressed to compute an EBSD pattern for the current set of parameters. Which pattern is displayed will depend on the settings in the Display Control Panel, shown in Fig. 4(a). The user can select the following display modes:

- *Display experimental pattern*: displays only the experimental pattern.
- *Display simulated pattern*: displays only the simulated pattern.
- *Display difference pattern*: displays the difference pattern.

- *Display overlap pattern*: displays both patterns added together.
- *Display color overlap*: displays one pattern in the red color channel and the other in the green+blue channel; for perfect match the resulting pattern should become grayscale.
- *Flicker*: displays the two patterns in an alternating fashion (10 times); to repeat this display, first select another option and then reselect *Flicker*.

The options at the top of the Pattern Display Panel (Fig. 4(b)) can be used to save the currently displayed pattern to a jpeg, tiff, or bmp file.

Section **3** shows a list of eight refinable parameters, the top five related to the detector and the final three the pattern Euler angles. For each parameter, there are a number of fields available. The Value and Stepsize fields allow the user to enter a starting value and a maximum stepsize for the iterative fitting procedure. When the program starts up, the values saved at the end of the previous program run will be present in the fields; they can be edited by selecting the current value, typing a new value and pressing the enter key. The next field, labeled "Refine?", allows the user to decide whether or not this parameter should be included in the fitting run. The Up and Down buttons can be used to increment or decrement each parameter manually by an amount set in the Manual Stepsize field; when either the Up or Down button is pressed, the parameter is updated and the new pattern is computed and displayed in the Pattern Display Panel, depending on the setting of the Display Control Panel.

In section **4**, the user can select one or more pre-processing steps to modify the appearance of the patterns. Sometimes it is useful to enhance the contrast, or subtract the background. Any setting changes in this section are automatically applied to both the experimental and simulated patterns and will be visible in the Pattern Display Panel. The user can adjust these settings as needed to improve the fit between the simulated and experimental pattern.

Finally, in section **5**, the user can control the parameter fitting. There are two convergence parameter choices: either the standard dot product between the two patterns or the mutual information. Experience shows that the dot product option usually performs better. The Fit Mode offers three options: in the Free mode, the user can select any combination of "Refine?" settings in Section **3** to determine which parameters will be refined. In "detector" mode, the first five parameters will be set to "Yes" and the Euler angles to "No"; this results in a fitting run for the detector parameters only, but the user can still manually change any parameter. For instance, for an initial fit, it is useful to turn off the omega sample tilt parameter as well as the Intensity Gamma Value (which sets how the pattern intensity is mapped onto the [0,255] range of pixel intensities. Finally, in "orientation" mode, all detector parameters are turned off and the Euler angles are turned on.

A parameter fit is initiated by pressing the Start Fit button. This will create a small pop-up window with an Abort button so that the user can stop the fit if it does not converge. It should be noted that this abort button may require multiple clicks before it responds and terminates the run; this is due to an internal problem and may be resolved in a future version. The last option, Create JSON File, will generate a .json input file that contains all the current settings of the detector parameters and can be used to execute the EMDPfit program, described in the next section.

3.2.2 Using the EMDPfit program

EMDPFit is a complementary program to the Interactive IDL EFit virtual app. While the EFit program relies on the Nelder-Mead downhill simplex, this program uses the **B**ound **O**ptimization **BY Q**uadratic **A**pproximation (**BOBYQA**) algorithm to optimize the similarity metric between

experimental and simulated patterns, the common theme for both techniques being the unavailability of closed form derivatives with respect to the model's parameters. The following section gives a brief overview of the method and an example input file to help the reader understand various parameters of the algorithm. However, these are very introductory remarks and the reader is encouraged to study the excellent original paper by M.J.D. Powell[3] to get an intuition of how the various parameters of the optimization routines might effect the final results.

3.3 Theoretical Framework

Consider a function $F(x), x \in \mathbb{R}^n$, subject to the bounds $a \leq x \leq b$ which we wish to optimize. The function is specified by a "black-box" with the input as x. The derivatives of F with respect to the input variables are not known analytically. Such problems are usually handled using the class of Derivative Free Optimization (DFO) algorithms. These methods typically approximate F in the region $||x|| \leq \Delta_k$ using a substitute function, Q such that $Q(y_j) = F(y_j), j = 1, 2, \cdots, m$ for each interpolating point y_j. The algorithm iteratively optimizes the function F. As its name suggests, BOBYQA uses a quadratic approximation for interpolation. The interpolating points are chosen and adjusted automatically. However, the number of points used for interpolation, m is typically a constant with $m = 2n + 1$ being a good choice. For the k-th iteration, the neighborhood in which the function Q is a good approximation is given by Δ_k. There is a lot of freedom in choosing Q which only satisfies the interpolation conditions. These are fullfilled by choosing Q such that the Frobenius or L^2 norm of the change in the second derivation, $||\nabla^2 Q_{k+1} - \nabla^2 Q_k||$ is minimized.

For the EMDPFit program, the number of interpolating points, m, has been set to $n + 6$ as it typically gives the best results and is not a choice for the user. The radius of trust region for each iteration k, Δ_k is also adjusted automatically. However, the starting and final values, ρ_{beg} and ρ_{end} are choices for the user. The ratio $(\rho_{\text{beg}}/\rho_{\text{end}})$ is a good measure of how fast the algorithm terminates and how accurate the final result is. A high value of this ratio leads to a more accurate but higher convergence time and a low value leads to a good result but faster performance. A good choice for these parameters to balance performance and accuracy is 10^{-1} and 10^{-5} respectively.

3.4 Example script

This section lists an example input name list file for the EMDPFit program and explains what each of the input parameters does.

```
 &DPFitdata
! template file for the EMDPFit program
! name of modality
 modalityname = 'EBSD' ! EBSD or ECP for now; PED will be added later,
! name of EBSD master output file; path relative to EMdatapathname
 masterfile = 'master.h5',
! name of Monte Carlo output file; path relative to EMdatapathname
 energyfile = 'MC.h5',
!#####################################################################
! FILL OUT exptfile_pat1 for ECP; ALL FOR EBSD
!#####################################################################
! name of file with the single experimental pattern
 exptfile_pat1 = 'undefined',
```

[3] M.J.D. Powell, "The BOBYQA algorithm for bound constrained optimization without derivatives" (Report), Department of Applied Mathematics and Theoretical Physics, Cambridge University. DAMTP 2009/NA06.

```
 exptfile_pat2 = 'undefined',
 exptfile_pat3 = 'undefined',
 exptfile_pat4 = 'undefined',

!-------number of runs of the algorithm----------------------------
!-------stepsize reduced by half for each run----------------------
!
 nrun = 2
! size of trust region (sort of)
 rhobeg = 1.D0-1,
 rhoend = 1.D0-7,
! verbose output or not
 verbose = .TRUE.,
! mask pattern while doing fit or not
 mask = .TRUE.,
!
!this section deal with the initial guess of some of the parameters together with the known parameters
!fill out the relevant parameters depending on the modality (EBSD or ECP)
!
!####################################################################
! SHARED PARAMETERS
!####################################################################
!
! mask radius
 maskradius = 240.0,
! gamma correction factor
 gammavalue = 1.0,
!####################################################################
! FILL OUT phi1_pat1,phi_pat1 and phi2_pat1 for ECP; ALL FOR EBSD
!####################################################################
! bunge euler angles (in degrees)
 phi1_pat1 = 0.0,
 phi_pat1 = 0.0,
 phi2_pat1 = 0.0,

 phi1_pat2 = 0.0,
 phi_pat2 = 0.0,
 phi2_pat2 = 0.0,

 phi1_pat3 = 0.0,
 phi_pat3 = 0.0,
 phi2_pat3 = 0.0,

 phi1_pat4 = 0.0,
 phi_pat4 = 0.0,
 phi2_pat4 = 0.0,

! step size for construction of quadratic surrogate function (degrees)
 step_phi1 = 2.0
 step_phi = 2.0
 step_phi2 = 2.0

!
!####################################################################
! EBSD PARAMETERS
!####################################################################
!
!---------known parameters-----------------------------------------
! tilt angle of the camera (positive below horizontal, [degrees])
```

```
 thetac = 10.0,
! CCD pixel size on the scintillator surface [microns]
 delta = 50.0,
! number of CCD pixels along x and y
 numsx = 640,
 numsy = 480,
! incident beam current [nA]
 beamcurrent = 150.0,
! beam dwell time [micro s]
 dwelltime = 100.0,
! binning mode (1, 2, 4, or 8)
 binning = 1,
!
!----------initial guess for fitting parameters--------------------
! distance between scintillator and illumination point [microns]
 L = 15000.0,
! pattern center coordinates in units of pixels
 xpc = 0.0,
 ypc = 0.0,
!
!----------step size for constructing surrogate function------------
! xpc and ypc in pixels
 step_xpc = 5.0
 step_ypc = 5.0
 step_L = 5.0
!
!----------EBSD scan parameters-------------------------------------
! x coordinate of scan point
 pixx_pat1 = 0
 pixx_pat2 = 0
 pixx_pat3 = 0
 pixx_pat4 = 0
! y coordinate of scan point
 pixy_pat1 = 0
 pixy_pat2 = 0
 pixy_pat3 = 0
 pixy_pat4 = 0
! step size in x
 stepx = 5.0
! step size in y
 stepy = 5.0
!########################################################################
! ECP PARAMETERS
!########################################################################
!
!---------known parameters-----------------------------------------
 npix = 256,
! inner and outer radius of annular integrating detector (in mm)
 Rin = 2.0,
 Rout = 5.0,
!
!----------initial guess for fitting parameters--------------------
! half angle of cone (in degrees)
 thetacone = 5.0,
! sample tilt (in degrees)
 sampletilt = 0.0,
! working distance (in mm)
 workingdistance = 5.0
!----------step size for constructing surrogate function------------
```

```
! in degrees
 step_thetacone = 1.0
 /
```

- modalityname: use EBSD or ECP for the appropriate diffraction modality. All ECP parameters are ignored if modality name is EBSD and vice versa. PED modality will be implemented in a future release.
- masterfile: name of the master input file; note that this should be a relative path with respect to the EMdatapathname variable.
- metric: similarity metric between experimental and simulated patterns. Options include JD for Jaccard Distance, which is a normalized form of mutual information and DP for dot product.
- exptfile: name of the experimental file; note that this should be a relative path with respect to the EMdatapathname variable. The format is the same as used in the EMEBSDDI program.
- nrun: number of minimizations runs. The search space is halved for each time the minimization routine runs.
- rhobeg: starting value for radius of the trust region,a good choice being 10^{-1}.
- rhoend: final value for radius of trust region, a good choice being 10^{-5}.
- verbose: verbose output or not
- mask: decides if an external circular mask should be applied to the pattern or not. Useful when acquired diffraction patterns are already masked.
- maskradius: radius of the applied circular mask in unit of pixel. A good choice is half of the smallest dimension of the image. For e.g. , if the image is 640×480, the mask can be set to 240.
- nregions: number of pixels in each square tile for adaptive histogram equalization.
- phi1,phi,phi2: initial guess for euler angles.
- step_phi1,step_phi,step_phi2: maximum search space for the corresponding variables. The algorithm will only look at phi $\pm$ step_phi etc.
- thetac:tilt angle of camera in degrees.
- delta:physical size of the each pixel on the detector in μm.
- numsx: dimension of the detector in x direction in units of pixels.
- numsy: dimension of the detector in y direction in units of pixels.
- beamcurrent: incident beam current in nA.
- dwelltime: dwell time at each sample location in mus.
- binning: binning mode used in experiment. This parameter, together with numsx and numsy decides the final size of the output image.

- L: distance between sample and scintillator in μm.
- xpc: x pattern center in units of pixel.
- ypc: y pattern center in units of pixel.
- step_L,step_xpc, step_ypc: maximum search space for the corresponding EBSD variables. The algorithm will only look at L $\pm$ step_L etc.
- npix: size of the image in case the diffraction pattern is an ECP.
- Rin: inner radius of backscatter detector in mm.
- Rout: outer radius of backscatter detector in mm.
- thetacone: half angle of incident cone in the ECP pattern in degrees.
- sampletilt: sample tilt while recording the channeling pattern in degrees.
- workingdistance: working distance in mm.
- step_thetacone: maximum search space for thetac variables. The algorithm will only look at thetac $\pm$ step_thetac etc.

Again. it is important to point out again that the same program can handle both EBSD and ECP modalities. The modalityname variable is used to specify which modality the diffraction pattern belongs to. Further, when the modalityname is set to EBSD, all ECP-specific parameters are redundant and vice versa.

4 Example Dictionary Runs for a Ni data set

4.1 Getting the EBSD data in the correct format

4.2 Fitting the detector parameters

4.3 Executing the indexing run

4.4 Interpreting the indexing output

5 Availability of Example Data Files

The data files used for the examples in this manual are all available as open source data.

References

[1] D. Roşca, A. Morawiec, and M. De Graef. A new method of constructing a grid in the space of 3D rotations and its applications to texture analysis. *Modeling and Simulations in Materials Science and Engineering*, 22:075013, 2014.

[2] S. Singh, A.D. Rollett, and M. De Graef. Orientation sampling for dictionary-based diffraction pattern indexing methods. *MSMSE*, 2016 (in preparation).

[3] D.J. Rowenhorst, A.D. Rollett, G.S. Roher, M.A. Groeber, M.A. Jackson, P.J. Konijnenberg, and M. De Graef. Tutorial: consistent representations of and conversions between 3d rotations. *Modeling and Simulations in Materials Science and Engineering*, 23:083501, 2015.