How to use MTEX to plot pole figures and inverse pole figures

October 24, 2016

MTEX is a free Matlab toolbox for analyzing and modeling crystallographic textures by means of EBSD or pole figure data. This document only explains the basis of how to plot pole figures and inverse pole figures from pole figures data obtained by X-ray diffraction, with the mtex version 4.0.23. It has been highly inspired from the MTEX toolbox documentation. To access this documentation and to download MTEX, go to http://mtex-toolbox.github.io/. A link to the forum managed by the creator of MTEX, Ralf Hielscher, is included on the site if you have any questions/problems with MTEX.

Contents

1 Import the data

The data presented in this document have been generated by X-rays diffraction using the machine Bruker, with a 2D detector to record the data. The sample to detector distance was of 162.5 mm, with 1024 x 1024 pixels active and two tilt conditions (25 and 75 degrees). The data were sampled with 1 degree resolution along gamma angle. The data were then extracted by MultiTex2 to generate a .uxd file for each pole figure. MTEX only accepts .uxd files with a specific structure; a matlab code converting the .uxd file coming from MultiTex2 into the .uxd file requested by MTEX is given in Annex.

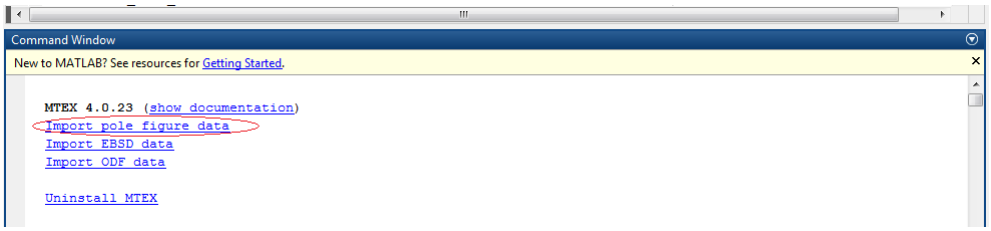


Figure 1: Three functions given by MTEX to import data.

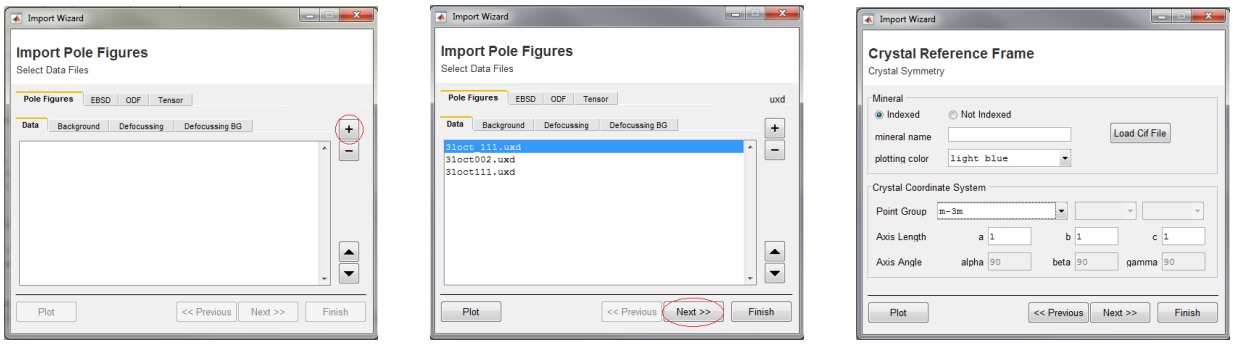


Figure 2: Import pole figure data

Download the toolbox, and install it. Three functions to import data should appear in matlab (see Fig. 1). Click on "Import pole figure data", a box should appear on the screen (Fig. 2(a)). Click on "+" and add your uxd files, then click on next. Fill the informations related to your material and click on "next" until the last box, then click "Finish". Be careful at the step shown in Fig. 2(e) that your Miller indices correspond to your pole figure. Mtex tries to guess it from the name of your file and a minus can be missing. When these steps are completed, an m-file script is generated. An example of this file is given below.

```
%% Import Script for PoleFigure Data
%
% This script was automatically created by the import wizard. You should
% run the whoole script or parts of it in order to import your data. There
% is no problem in making any changes to this script.

%% Specify Crystal and Specimen Symmetries

% crystal symmetry
CS = symmetry('m-3m');

% specimen symmetry
SS = symmetry('triclinic');

% plotting convention
setMTEXpref('xAxisDirection','north');
setMTEXpref('zAxisDirection','outOfPlane');

%% Specify File Names

% path to files
pname = 'path_name_to_your_file';

% which files to be imported
fname = [pname '\PF100.uxd',
         pname '\PF010.uxd',
         pname '\PF001.uxd',
         pname '\PF111.uxd'];

%% Specify Miller Indice

h = { ...
  Miller(1,0,0,CS),...
  Miller(0,1,0,CS),...
  Miller(0,0,1,CS),...
  Miller(1,1,1,CS),...
  };

%% Import the Data

% create a Pole Figure variable containing the data
pf = loadPoleFigure(fname,h,CS,SS,'interface','uxd',...
  'wizard');
```

You can change directly in this script the name of your .uxd files and their Miller indices to import other pole figures or click again on "import pole figure data" to create more scripts.

2 Correct the data

Once your pole figures data are imported, you can plot them to see how they look like. The figures obtained are given in Fig. 3. To do so, just add:

```
plot(pf);
colorbar;
```

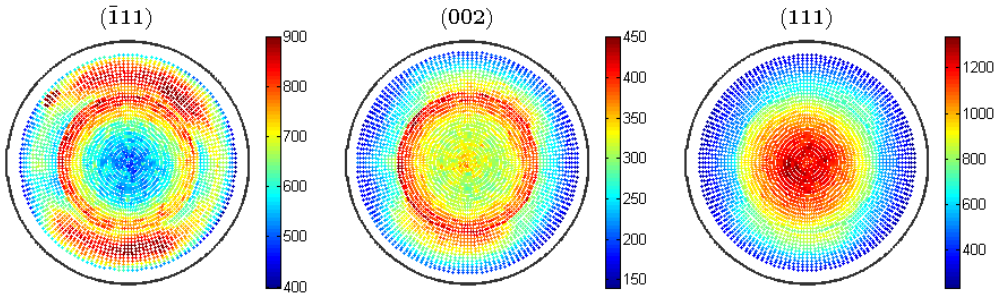


Figure 3: Pole figures from the raw data.

Sometimes some artifacts appear during the X-ray diffraction measurements, showing incredibly high intensities. In other to remove these artifacts, one can use the following code:

```
% Remove intensities higher than 900
condition = pf.intensities > 900;
% cap the values in the pole figures
pf(condition).intensities = 900;
```

Replace pf by pf({1}) if you want to limit this correction to the first pole figure imported.

MTEX provides a function "correct" to reduce the error made during the measurement of the pole figures data. This function corrects the defocusing and the error on the background:

```
pf = correct(pf,'background',pf_background);
pf = correct(pf,'def',def_pf);
```

The background correction is just the substraction between the values of your pole figure pf and the values of pf background. pf background is a pole figure containing the value of the background intensities (ie the incoherent radiations). It can be made by generating a .uxd file with the background value for each tilt angle (KHI) given at one rotation angle (0 degree). The other rotation angles (from 2.5 to 357.5 degree) can be removed in the code as they are all the same for the background correction. When plotting pf background, one should get a line of intensity in the pole figure (see Fig.4).

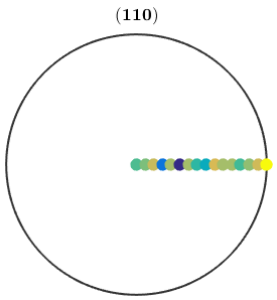


Figure 4: Pole figure containing the background intensities to be removed in order to correct the main pole figure. See http://mtex-toolbox.github.io/files/doc/ModifyPoleFigureData.html

These pole figures are expressed in counts, depending on the amount of X-rays diffracted and recorded by the detector. In order to compare several pole figures, one should normalize them in m.r.d. (multiples of a random distribution). This normalization express the regions of the pole figures as a comparison with a random

one, in which all data would be 1. The regions with an intensity higher than one indicates that more lattice planes are aligned in this direction than they would be in a random texture. The normalization can be done by this function:

```
pf_norm = normalize(pf);
```

However, in the case of incomplete pole figures, the normalization can only by computed from an ODF. This is our case as one can see that the intensities between 80 and 90 degrees could not be recorded. The way to normalize these incomplete pole figure is to use the Orientation Distribution Function as explained in the next chapter.

3 Plot the data

3.1 Pole figure

A complete texture requires a 3D representation, and pole figures only show a 2D projection of the texture so some information is lost. In order to correctly stock all the information, a 3D-representation must be calculated. This 3D-representation is called an Orientation Distribution Function or ODF. An ODF expresses the 3 rotations needed to align each crystal with the specimen (the whole material) in a Euler space. An ODF can not be directly measured and must be calculated from the pole figures. The number of pole figures needed to build an ODF depends on the quality of the measurement and on the crystal symmetry. Only 3-4 pole figures are usually needed for a cubic crystal, 5-6 for an hexagonal one and more for systems with less symmetry. The MTEX function to calculate an ODF is:

```
odf = calcODF(pf);
```

This function also take the ghost error into account. The ghost error is caused by the missing odd-order series expansion coefficients in the reconstruction of the ODF from the pole figures, in the series expansion method. The ghost error caused negative ghosts (missing intensities in ODF) or positive ghosts (added intensities).
One can check the accuracy of its ODF by plotting back the pole figures from the ODF and by comparing them with the original pole figures.

```
plotPDF(odf,pf.h);
colorbar;
```

MTEX also supports the comparison between the ODF and the original pole figures with the function plotDiff, which plot the difference between the original pole figures and the reconstructed ones from the ODF.

```
plotDiff(pf,odf);
colorbar;
```

One can see that the ODF is not perfect in our cases, by comparing Fig.5(a) and Fig.5(b), or by checking the error in Fig.5(c). These errors are due to the fact that the crystal analysed is monoclinic and more than 3 poles figures should be used to plot the ODF. If other pole figures are not available and/or if one just wants to use the function plotPDF to show the raw data in a smoother appearance, then the plotPDF function can be used with an ODF constructed from only one pole figure at a time. The ODF won't represent the real texture of the material but the reconstructed pole figure will look exactly like the original pole figure (see Fig.7).

By clicking on the figure, one can obtain the exact intensity in that region, and the angle defining this region. After having plotted your pole figure, one can also make some changements:

- : Change the range of the colorbar: Edit - Colormap;
- : Change the color of the colormap: MTEX - Colormap;
- : Change the orientation of the pole figure: MTEX - X axis direction and select North South East or West; MTEX - Z axis direction and select Out of plane or Into plane.

The orientation of the pole figure can be indicated by using the function "annotate": it will indicate the specimen orientations in the pole figure.

```
plotPDF(odf,pf.h);
annotate([xvector,yvector,zvector],'label',{'X','Y','Z'},'BackgroundColor','w');
colorbar;
```

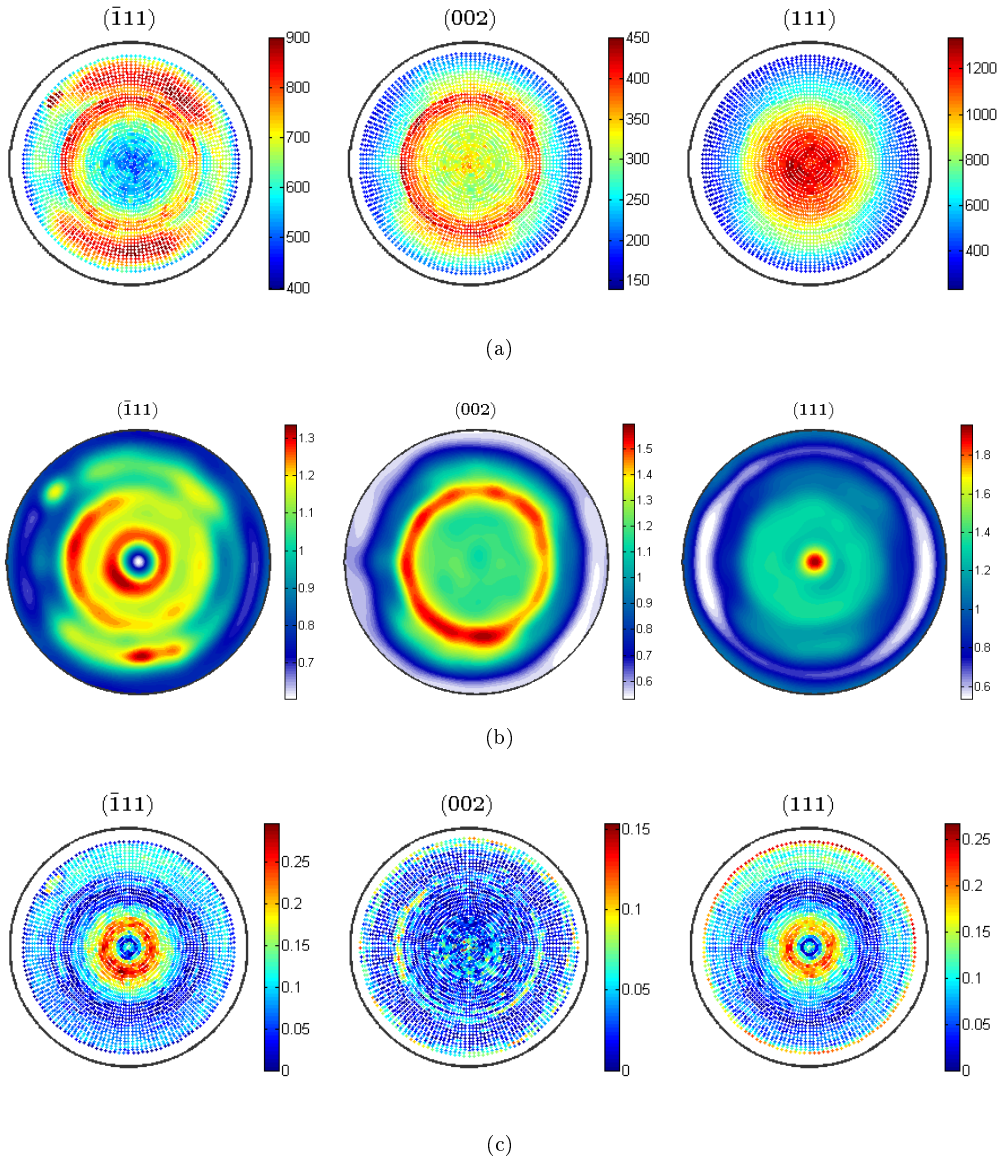


Figure 5: (a) Original pole figures normalized (b) Reconstructed pole figures from the ODF, calculated with the 3 pole figures (c) Loss of information during the calculation of the ODF: difference between the original pole figures (a) and the reconstructed ones (b).

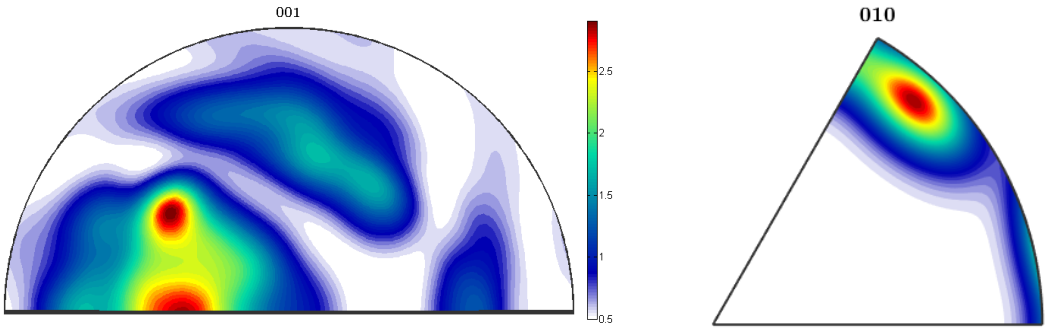
3.2 Inverse pole figure

Inverse pole figures can be plotted from the ODF by using this function (see Fig. 6):

```
plotIPDF(odf_norm_Stress,[zvector],'antipodal');
annotate([Miller(1,1,1,CS),Miller(0,1,1,CS),Miller(1,1,0,CS)],'label',{'(111)','(011)','(110)'},'BackgroundColor'
colorbar;
```

[zvector] will show the (001) inverse pole figure (ie the intensities of all the cristallographic orientations in the (001) specimen direction). One can also use [xvector yvector zvector] to plot the (100), (010) and (001) inverse pole figures, or even 3dvector(i,j,k) to plot the (ijk) inverse pole figure.
Use the word "complete" to plot the entire inverse pole figure, or the word "antipodal" to only plot the fundamental region of the inverse pole figure (to avoid the repetition in the inverse pole figure due to the antipodal symmetry). In our case, we have a monoclinic structure so the "antipodal" inverse pole figure is half a circle, but in the case of more highly symmetric structure the fundamental region of the inverse pole figure is reduced. For example, a cubic structure of order 48 has 24 equivalent points in the complete inverse pole figure (24 in the upper side of the complete inverse pole figrue and 24 in the lower side) so a twelfth of this circle is sufficient to represent the inverse pole figure. The usual fundamental region chosen for a cubic structure is one of the inner triangles (see http://www.ebsd.com/popup/figureset.htm?fig42).
The inverse pole figure of a monoclinic and a cubic structure can be seen on the Fig. 6.

The function "annotate" allows one to indicate specific cristallographic orientations in the inverse pole figure. One can also click on the figure once plotted to obtain these crystallographic orientations and their associated intensity.



(a) (b)

Figure 6: Inverse pole figure (a) Monoclinic structure (b) Cubic structure

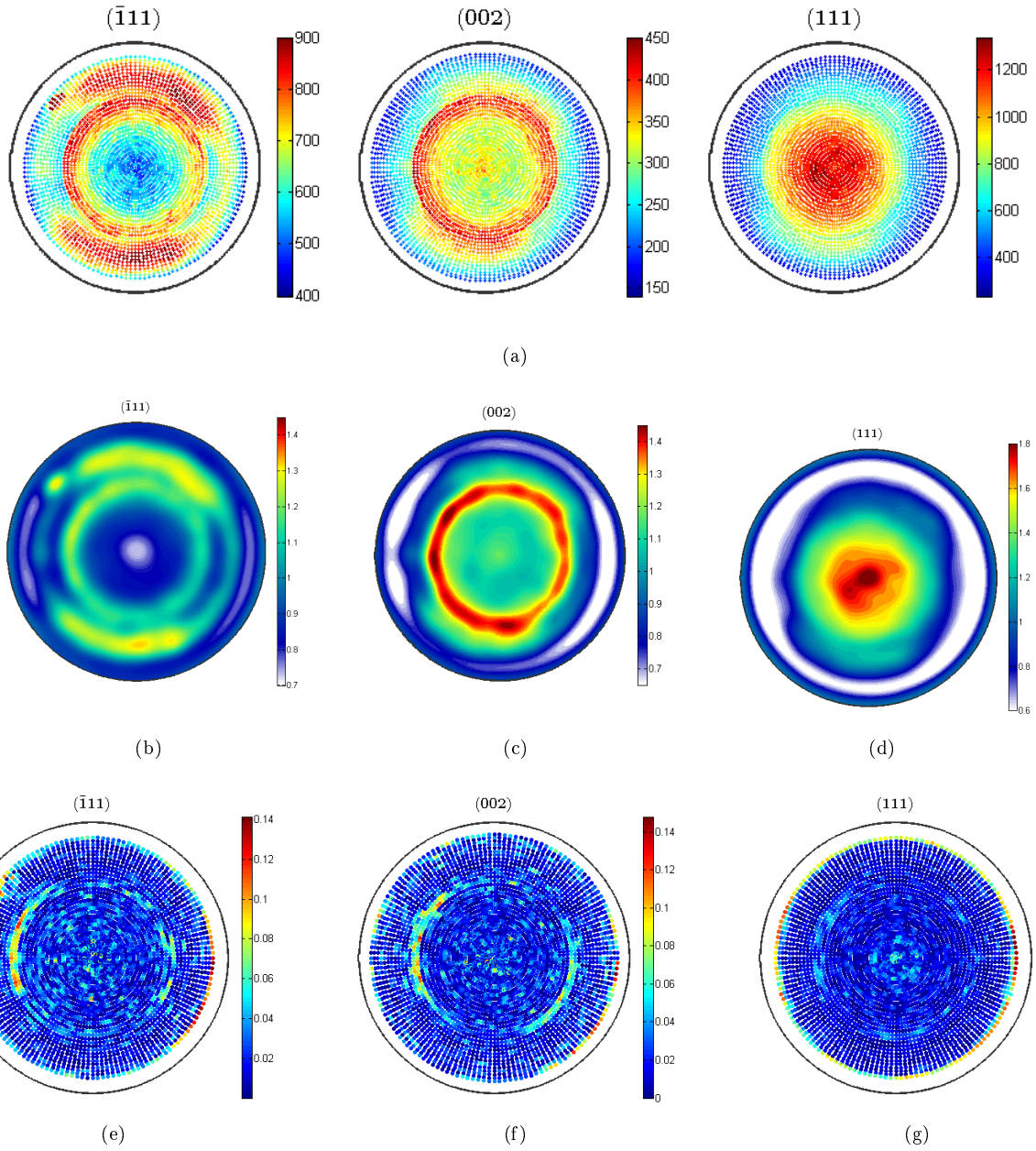


Figure 7: (a) Original pole figures. (b) to (d): Pole figures reconstructed with an ODF calculated for each pole figures. (e) to (g): Loss of information during the calculation of each ODF.

4 Annex: matlab code to convert .uxd file from MultiTex2 to .uxd file requested by mtex

```
function [] = PoleFigureTransfer()
% Read a .uxd file coming from Multitex 2, take the data and write them in another .uxd file
% that can be uploaded as a pole figure data into mtex.
% i is the number of matrices read (Data for Range number i).
i=0;
% Put the path to the first textfile
fid = fopen('C:\Users\Laurine\Dropbox\MIT\Texture\Pole figures data\Random002.uxd','r');
% Skip some unnecessary lines in the text
for a = 1:4
    tline=fgetl(fid);
end
% Read the text until it is over
while feof(fid)==0
```

```
    i = i+1;
    % Memorise the Range
    A(i,1)=i;
    % Skip some lines
    for a=1:6
        tline = fgetl(fid);
    end
    % Save the KHI number (tilt angle)
    for b=1:1
       tline = fgetl(fid);
       B = strread(tline,'%s','delimiter','=');
       A(i,2) = str2double(B(2));
    end
    % Skip some lines
    for a = 1:2
        tline=fgetl(fid);
    end
    % j is the number of lines in the matrix for one Range (for a fixed KHI)
    for j=1:18
        tline = fgetl(fid);
        A2 = strread(tline,'%d','delimiter',' ')';
        indice = j+(j-1)*7+2;
        A(i,indice:indice+7)=A2;
    end
end
fclose(fid);
% Put the path and the name of the second file which must be save then as a .uxd
fileID = fopen('C:\Users\Laurine\Dropbox\MIT\Texture\Pole figures data\Random002.uxd','w');
format2 = '; E:\\data\\texture\\Zhangxinming\\lili\\Deepdrawing1_R.raw(Diffrac Plus V1.01 file) converted by XCH
fprintf(fileID,format2);
formatSpec = '; (Data for Range number %d)\n_DRIVE=''PHI''\n_STEPTIME=1.000000\n_STEPSIZE=5.000000\n_STEPMODE=''C
0.0000    %d\n   2.5000    %d\n   5.0000    %d\n   7.5000    %d\n  10.0000    %d\n  12.5000    %d\n
15.0000    %d\n  17.5000    %d\n  20.0000    %d\n  22.5000    %d\n  25.0000    %d\n  27.5000    %d\n
30.0000    %d\n  32.5000    %d\n  35.0000    %d\n  37.5000    %d\n  40.0000    %d\n  42.5000    %d\n
45.0000    %d \n  47.5000    %d\n  50.0000    %d\n  52.5000    %d\n  55.0000    %d\n  57.5000    %d\n
60.0000    %d\n  62.5000    %d\n  65.0000    %d\n  67.5000    %d\n  70.0000    %d\n  72.5000    %d\n
75.0000    %d\n  77.5000    %d\n  80.0000    %d \n  82.5000    %d\n  85.0000    %d\n  87.5000    %d\n
90.0000    %d\n  92.5000    %d\n  95.0000    %d\n  97.5000    %d\n 100.0000    %d\n 102.5000    %d\n 105.0000
%d\n 107.5000    %d\n 110.0000    %d\n 112.5000    %d\n 115.0000    %d\n 117.5000    %d\n 120.0000
%d\n 122.5000    %d\n 125.0000    %d\n 127.5000    %d\n 130.0000    %d\n 132.5000    %d\n 135.0000
%d\n 137.5000    %d\n 140.0000    %d\n 142.5000    %d\n 145.0000    %d\n 147.5000    %d\n 150.0000
%d\n 152.5000    %d\n 155.0000    %d\n 157.5000    %d\n 160.0000    %d\n 162.5000    %d\n 165.0000
%d\n 167.5000    %d\n 170.0000    %d\n 172.5000    %d\n 175.0000    %d\n 177.5000    %d\n 180.0000
%d\n 182.5000    %d\n 185.0000    %d\n 187.5000    %d\n 190.0000    %d\n 192.5000    %d\n 195.0000
%d\n 197.5000    %d\n 200.0000    %d\n 202.5000    %d\n 205.0000    %d\n 207.5000    %d\n 210.0000
%d\n 212.5000    %d\n 215.0000    %d\n 217.5000    %d\n 220.0000    %d\n 222.5000    %d\n 225.0000
%d\n 227.5000    %d\n 230.0000    %d \n 232.5000    %d\n 235.0000    %d\n 237.5000    %d\n 240.0000
%d\n 242.5000    %d\n 245.0000    %d\n 247.5000    %d\n 250.0000    %d\n 252.5000    %d\n 255.0000
%d\n 257.5000    %d\n 260.0000    %d\n 262.5000    %d\n 265.0000    %d\n 267.5000    %d\n 270.0000
%d\n 272.5000    %d\n 275.0000    %d\n 277.5000    %d\n 280.0000    %d\n 282.5000    %d\n 285.0000
%d\n 287.5000    %d\n 290.0000    %d\n 292.5000    %d\n 295.0000    %d\n 297.5000    %d\n 300.0000
%d\n 302.5000    %d\n 305.0000    %d\n 307.5000    %d\n 310.0000    %d\n 312.5000    %d\n 315.0000
%d\n 317.5000    %d\n 320.0000    %d\n 322.5000    %d\n 325.0000    %d\n 327.5000    %d\n 330.0000
%d\n 332.5000    %d\n 335.0000    %d\n 337.5000    %d\n 340.0000    %d\n 342.5000    %d\n 345.0000
%d\n 347.5000    %d\n 350.0000    %d\n 352.5000    %d\n 355.0000    %d\n 357.5000    %d\n' ;
fprintf(fileID, formatSpec, A');
fclose(fileID);


end
```