Results Processing in MATLAB for Photonics Applications

I.V. Guryev, I.A. Sukhoivanov, N.S. Gurieva, J.A. Andrade Lucio and O. Ibarra-Manzano
University of Guanajuato, Campus Irapuato-Salamanca, Division of Engineering Mexico

1. Introduction

The chapter is intended to provide the reader with powerful and flexible tools based on MATLAB and its open-source analogs for the processing and analyzing the results obtained by means of highly specialized software.
Particularly, in the chapter we give a brief overview of free and open-source as well as shareware software for computation of the photonic crystals characteristics. We concentrate attention mostly to their advantages and drawbacks and give short description of the data files they give results in.
The next parts of the chapter are dedicated to processing of the results obtained within the specific software.
Firstly, we consider the most basic case of plane results represented by a functional dependences. In this part, we are talking about the plane data interpolation, approximation and representation.
Data interpolation may be used as a tool intended to avoid high resolution computation (when scanning, say, through the wavelength or frequency) and, therefore, to reduce computation time which may achieve weeks in the modern numerical problems.
Data approximation allows highly efficient data analysis by means of finding the parameters of the target function. For this reason, the numerically obtained data is approximated (within the certain accuracy) by the analytical functions including variable coefficients which are usually posses physical meaning. As a result of the approximation, the unknown coefficients are found.
Correct and obvious data representation is very important when writing the scientific paper. Therefore, in the end of current part we give an idea of figure formatting.
The next part of the chapter shows reader how to process and represent three-dimensional and multi-dimensional data. Basically, the operations described for plane data is extrapolated to multidimensional data arrays.
In many cases, the results of the complex numerical research is represented in the form of multiple files with similar format. In the next part of the chapter we show how to use MATLAB for merging the data into a single multidimensional array and for representing it in convenient form.

The last part of the chapter is dedicated to the animations creation out of multiple instant data shots. Such form of the data representation is useful when representing time-dependent results as well as multidimensional data arrays.
During all the chapter, we provide reader with working MATLAB codes as well as figures illustrating the programs results. We believe this information together with learning of MATLAB fundamentals *MATLAB manual* (2011) will help young scientists, master and PhD students especially in the area of opto-electronics and photonics to analyze and represent their computation results in the most effective way.

2. Brief review of the Photonic crystals modeling software

Photonic crystals are optical media possessing the periodic modulation of the refractive index. Due to this, the light behavior inside such structures is similar to the one of the electrons in the atomic structures (this gives them the name "Photonic crystals"). Main property the PhCs possesses is the photonic band gap (PBG) which is an optical analog of the electronic band gap in semiconductor materials.
Depending on the geometrical peculiarities, the PhCs are subdivided to several categories. Particularly, there are wide classes of 1D, 2D and 3D PhCs classified by number of dimensions where the variations of the refractive index appear. These classes, in turn, are divided according to the lattice type, existence or absence of the PBG, etc.
The physical principles of the PhCs are well-known and are considered in numerous articles and books Joannopoulos et al. (1995), Lourtioz et al. (2005), Sakoda (2001). There exist numerous methods for modeling the PhC characteristics, which are well described Sakoda (2001), Taflove & Hagness (2005) and are implemented in a numerous software products. This chapter is considered by us as a natural addition to the book Sukhoivanov & Guryev (2009) which describes computation methods of the PhCs characteristics. It is intended to extend and improve the efficiency of the results representation.
Particularly, this section is dedicated to brief introduction into freeware and shareware software for PhC and PhC-based devices basic modeling.
Since the PhC-based devices are investigated intensively today, the crucial moment is the right choice of the computational methods which are implemented in a number of software.
Particularly, the development of the PhC devices require detailed analysis of the PhC eigenstates (or resonant frequencies) as a dependence of the direction of the light propagation. This kind of tasks may be done by means of different methods like analytical one (however, for 1D PhC only), finite differences time domain (FDTD) method, finite elements method (FEM), plane wave expansion method (PWE). Each method has its advantages and drawbacks, they all require different computation time and accuracy and are implemented in a different kinds of software.
The other important problem for PhC investigation is the computation of the field distribution inside the defined structure. Again, in the simplest cases (particularly, in case of 1D PhC with passive optical materials) the field distribution can be found analytically. However, in the most cases the numerical methods are required such as FDTD, FEM, PWE, etc.
Most of the software (especially, free ones) are intended to implement all the aspects of a single methods. However, professional shareware soft implements several different methods allowing the user to investigate the device from different points of view.

2.1 Free and open-source software

Within all the variety of the scientific software available, the developers are allowed to choose between low price and friendly interface. The friendly interface provides the user with a lot of default settings and saves a lot of time which may be spent to the research problems. On the other hand, being more complicated for user, free software requires deep knowledge of physical and mathematical aspects of the investigated area (which is considered by authors as an advantage of free software). Particularly, the scientist intended to investigate the optical properties of the periodic structures has to be well-educated in the fields of solid-state physics, electromagnetism, numerical methods, etc. Having these knowledge, the user may define the problem more precisely and carry out the research at higher level with deeper understanding of the matter of the problem.

Therefore, this part of our review is dedicated to the software which is available online and may be easily tested and learned by reader.

2.1.1 MPB

The MIT Photonic Bands (MPB) *MPB* (2011) software is one of the most powerful tools available freeware which is intended to compute the eigenstates of the photonic crystal of different configurations by means of PWE method. It is written by developers of the PhC theory whose investment into the physics of PhC devices was critical. Therefore, the MPB solves the problems in a most suitable way for the scientists.

However, since MBP runs script files with detailed description of the structure and computation conditions, using of the MBP requires from user high skills, particularly, in solid-state physics and turns computation of even the simplest case into a serious programming task. Several examples provided with the installation help to understand the principles of the structure setup and they may be used as a default structures which is easy to modify and use for other structures.

The program outputs results into console which allows to read them but not analyze. Therefore, file output should be used. Particularly, the field distribution is being output into HDF5 format which is supported by visualization tools like h5utils. However, even for simple analysis, approximation and decoration the user has to use additional software such as MATLAB.

2.1.2 MEEP

Meep (or MEEP) *MEEP* (2011) is a free finite-difference time-domain (FDTD) simulation software package developed at MIT to model electromagnetic systems.

It can be used to successfully compute electromagnetic field distribution inside the PhC-devices made either of linear or nonlinear materials (possessing Kerr of Pockels nonlinearities).

2.2 Shareware software

In contrast to free software, the shareware ones have, in most cases, highly advanced user interface which helps novice to study the new soft and, on the other hand, saves professionals' time keeping them from the necessity of learning side skills such as programming.

As for the PhC modeling, the most advanced computation tools available on the markets are RSoft (namely, BandSolve and FullWave) and Comsol Multiphysics (namely, RF module).

2.2.1 RSoft

RSoft *RSoft* (2011) being developed by the RSoft-group for many years and it have come a long way since its earlier versions to its modern view.
Though its interface has been changed from version to version, it uses the same methods for computation of the PhC characteristics. Namely, BanSolve module allows one to compute the band structures of the PhC as well as its modes' field distribution by means of PWE method while FullWave implements FDTD method applying it to the field distribution computation inside the arbitrary structures and, in special case, the band structure.
The RSoft interface provides user with many useful models generated with default parameters. Moreover, wise linking to the variables allows flexible modifications of the default structures. This allows you to create the simple structure in a several mouse clicks and more advanced structures in several minutes. Among the drawbacks of the interface it should be mentioned too simple 3D structure definition which does not allow visual representation of the structure. To do this, additional software such as MayaVi should be used.
Except the possibility of modifying the structure with variables, RSoft's dialog boxes provide dozens of visual controls allowing definition of an arbitrary structure, computation conditions and the output information.
It should also be mentioned the graphic output of the RSoft. Although it allows analyze the computation results, their quality is unsatisfactory and one have to use additional software to make results representation suitable for publications. Moreover, in case of FullWave module, continuous graphic output essentially slows down the computation process.
In general, the RSoft is a powerful PhC modeling tool which allows solution of wide range of problems. However, results output is performed into the data files only and requires additional postprocessing, particularly, with Matlab.

2.2.2 Comsol Multiphysics

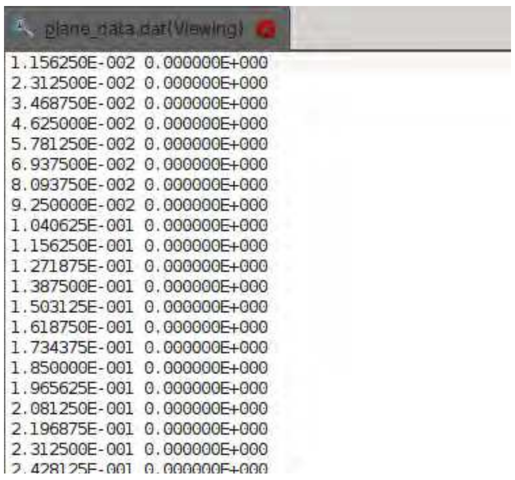
The Comsol Multiphysics *COMSOL* (2011) (previously known as Femlab) allows solving wide variety of problems defined by the partial differential equations in a number of fields of Physics by means of FEM. Particularly, in the area of PhC devices it allows to solve the Helmholtz equation with certain kinds of boundary conditions, thus, giving as a result the electromagnetic field distribution inside the device.
The structure definition as well as setting the properties of the structure, boundary conditions and node conditions are highly visualized and it makes no difficulties in creating computation model.
As for results representation, Comsol Multiphysics provides a lot of different visualization modes in form of 2D, 3D, contour, etc., thus, making for user unnecessary to deal with visualization of quite complicated data structures.
However, field distribution and basic data analysis is not always enough for serious scientific research. In case the PhC device has complex structure and requires detailed analysis of the field distribution (like confinement factor of the waveguide), the output files data should be investigated separately.
Therefore, independent on the convenience of the software itself, the problem of the output data processing is always arising and the best way to carry this out is by means of MATLAB.

3. Data files formats

As a result of computation of any PhC characteristic such as band structure, field distribution, temporal response, etc., the data is alway represented in some special format which can be read and processed with MATLAB.
With simulation software you may obtain several different kinds of data files, namely, the plane data (two-coordinate dependence), 3D data (tree-coordinate dependence), random data (where data points are linked to the mesh nodes), structured data (the data is represented in the form of known structures), multiple files data (usually, the parametric data where each file corresponds to the solution with a single value of the parameter).

3.1 Reading plane data

The data represented in a plane form usually represents the tabulated function of one variable and is really the simplest data format. In many cases, the computational programs output the data in a form of two rows as presented in figure 1



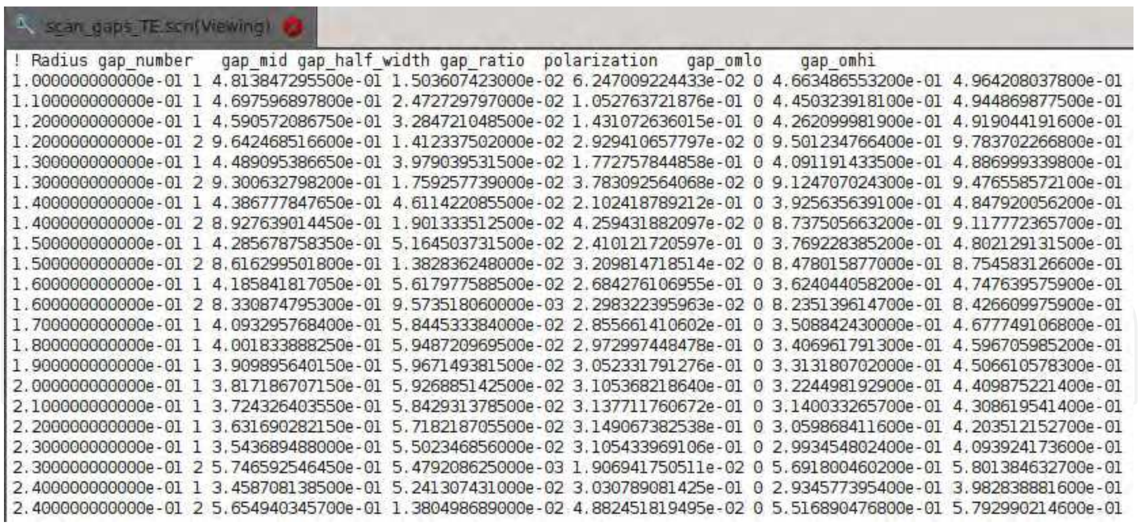
```
1.156250E-002 0.000000E+000
2.312500E-002 0.000000E+000
3.468750E-002 0.000000E+000
4.625000E-002 0.000000E+000
5.781250E-002 0.000000E+000
6.937500E-002 0.000000E+000
8.093750E-002 0.000000E+000
9.250000E-002 0.000000E+000
1.040625E-001 0.000000E+000
1.156250E-001 0.000000E+000
1.271875E-001 0.000000E+000
1.387500E-001 0.000000E+000
1.503125E-001 0.000000E+000
1.618750E-001 0.000000E+000
1.734375E-001 0.000000E+000
1.850000E-001 0.000000E+000
1.965625E-001 0.000000E+000
2.081250E-001 0.000000E+000
2.196875E-001 0.000000E+000
2.312500E-001 0.000000E+000
2.428125E-001 0.000000E+000
```

Fig. 1. The example of the plane data file

In the one wants to plot or process this kind of data, then it can be easily read by MATLAB script using a single command:

varName=load(fileName, `'-ASCII'`)

After this, the variable varName contains an array Nx2.
However, in some cases, the plane data file has the text header before the data itself (see figure 2 which demonstrates the data where the first line is a text header). This causes the problem since the data cannot be read with load command from the MATLAB. One of the solution is to use the Import Data dialog (Menu File/Import Data…) which performs smart data analysis and separates the numerical data from the text one.
On the other hand, it is more suitable to integrate the data import procedure into the script in order to avoid manual operations and, to be able to perform multiple import operations when processing multiple data files. For this reason, we have to use standard data reading operations to analyze file. The drawback of this method is necessity to know exactly the file structure, namely, how many strings takes the text header. Here is the simple example of opening the file generated by RSoft and having the text header.



```
! Radius gap_number   gap_mid gap_half_width gap_ratio  polarization   gap_omlo    gap_omhi
1.000000000000e-01 1 4.813847295500e-01 1.503607423000e-02 6.247009224433e-02 0 4.663486553200e-01 4.964208037800e-01
1.100000000000e-01 1 4.697596897800e-01 2.472729797000e-02 1.052763721876e-01 0 4.450323918100e-01 4.944869877500e-01
1.200000000000e-01 1 4.590572086750e-01 3.284721048500e-02 1.431072636015e-01 0 4.262099981900e-01 4.919044191600e-01
1.200000000000e-01 2 9.642468516600e-01 1.412337502000e-02 2.929410657797e-02 0 9.501234766400e-01 9.783702266800e-01
1.300000000000e-01 1 4.489095386650e-01 3.979039531500e-02 1.772757844858e-01 0 4.091191433500e-01 4.886999339800e-01
1.300000000000e-01 2 9.300632798200e-01 1.759257739000e-02 3.783092564068e-02 0 9.124707024300e-01 9.476558572100e-01
1.400000000000e-01 1 4.386777847650e-01 4.611422085500e-02 2.102418789212e-01 0 3.925635639100e-01 4.847920056200e-01
1.400000000000e-01 2 8.927639014450e-01 1.901333512500e-02 4.259431882097e-02 0 8.737505663200e-01 9.117772365700e-01
1.500000000000e-01 1 4.285678758350e-01 5.164503731500e-02 2.410121720597e-01 0 3.769228385200e-01 4.802129131500e-01
1.500000000000e-01 2 8.616299501800e-01 1.382836248000e-02 3.209814718514e-02 0 8.478015877000e-01 8.754583126600e-01
1.600000000000e-01 1 4.185841817050e-01 5.617977588500e-02 2.684276106955e-01 0 3.624044058200e-01 4.747639575900e-01
1.600000000000e-01 2 8.330874795300e-01 9.573518060000e-03 2.298322395963e-02 0 8.235139614700e-01 8.426609975900e-01
1.700000000000e-01 1 4.093295768400e-01 5.844533384000e-02 2.855661410602e-01 0 3.508842430000e-01 4.677749106800e-01
1.800000000000e-01 1 4.001833888250e-01 5.948720969500e-02 2.972997448478e-01 0 3.406961791300e-01 4.596705985200e-01
1.900000000000e-01 1 3.909895640150e-01 5.967149381500e-02 3.052331791276e-01 0 3.313180702000e-01 4.506610578300e-01
2.000000000000e-01 1 3.817186707150e-01 5.926885142500e-02 3.105368218640e-01 0 3.224498192900e-01 4.409875221400e-01
2.100000000000e-01 1 3.724326403550e-01 5.842931378500e-02 3.137711760672e-01 0 3.140033265700e-01 4.308619541400e-01
2.200000000000e-01 1 3.631690282150e-01 5.718218705500e-02 3.149067382538e-01 0 3.059868411600e-01 4.203512152700e-01
2.300000000000e-01 1 3.543689488000e-01 5.502346856000e-02 3.105433969106e-01 0 2.993454802400e-01 4.093924173600e-01
2.300000000000e-01 2 5.746592546450e-01 5.479208625000e-03 1.906941750511e-02 0 5.691800460200e-01 5.801384632700e-01
2.400000000000e-01 1 3.458708138500e-01 5.241307431000e-02 3.030789081425e-01 0 2.934577395400e-01 3.982838881600e-01
2.400000000000e-01 2 5.654940345700e-01 1.380498689000e-02 4.882451819495e-02 0 5.516890476800e-01 5.792990214600e-01
```

Fig. 2. The example of the data file containing text header

```
%Code for reading the data file containing text header

%opening the file
file=fopen('scan_gaps_TE.scn');

%skipping the file header (this part should be modified according
%to the specific file format)
fscanf(file,'%s',9);

%Reading the first line of the data.
%variable 'read' contains the data from the read string,
%variable 'num' contains the number of variables actually read.
%the length of the array to read to should be selected according
%to the dimensions of the data in the file
[read, num]=fscanf(file,'%f',[1,8]);

%Resetting the counter to write into the data array
counter=1;

%main reading cycle. Executes while the data read from the file
%contains actual data (num>0)
while(num)
  %copying the data from temporary variable into the data
  %variable
  data(counter,:)=read;

  %increasing the counter value
  counter=counter+1;

  %reading the next line from the file
  [read, num]=fscanf(file,'%f',[1,8]);
```

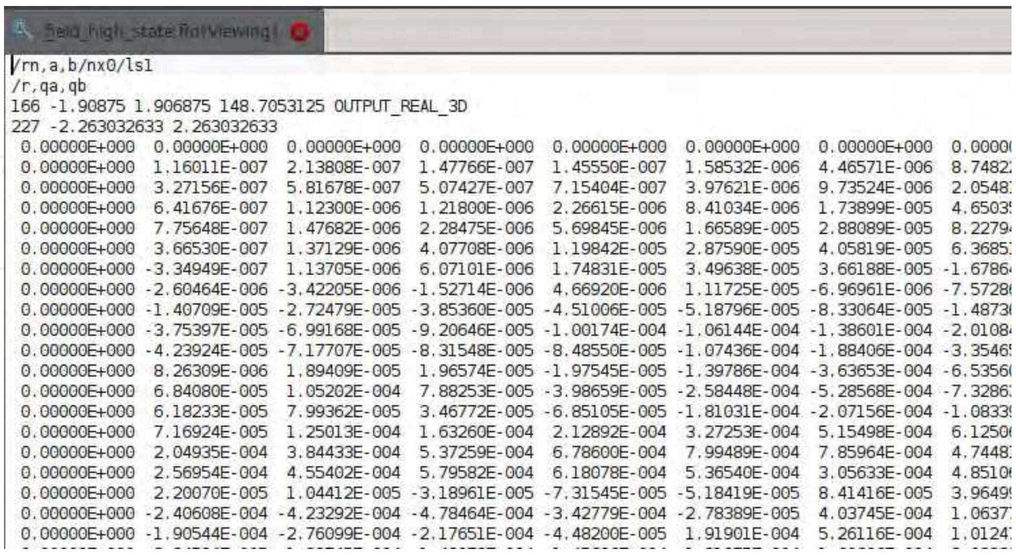
end

%closing the data file
fclose(file);

This code opens the data file as a regular file and then uses the function scanf() to read the data of the certain format. Particularly, for the file presented in the figure 2 it first reads 9 strings separated by the whitespaces and then reads each line containing 8 numbers into an array 1x8. Then in the cycle it copies each row into the resulting 2D array.
In case of plane data, each column has its meaning and have to be treated by the program accordingly. For instance, in case of the RSoft file generated by the scanning the full photonic band gaps (PBG) over some parameter, the first row indicates the value of the parameter, the second one is for the number of the PBG starting from the one with the lowest frequency, the third is for the central frequency of the PBG and so on. Detailed information on the specific files formats is usually contained in the manual for the specific software.

3.2 Reading 3D data

3D data is represented in the file in form of 2D array. In this case, each dimension of the array corresponds to the space dimension and, therefore, each number placed in the position (X,Y) in the file is treated as an altitude of the point with coordinates (X*scale,Y*scale). The value of the variable 'scale' depends on the physical dimensions of the structure.
Usually, reading of the 3D data does not differ from reading the plane data. In case of files without headers, the data is read by a simple load() function of the MATLAB. The scale of the structure should be known in this case to be able to assign physical coordinates to the positions in the array.
However, in case of the header presence, we have to use the method described in the subsection 3.1 for plane data. In this case the header may contain useful information such as linking to the physical dimensions, position of the structure, etc (see figure 3).



```
/rn,a,b/nx0/ls1
/r,qa,qb
166 -1.90875 1.906875 148.7053125 OUTPUT_REAL_3D
227 -2.263032633 2.263032633
 0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.0000
 0.00000E+000  1.16011E-007  2.13808E-007  1.47766E-007  1.45550E-007  1.58532E-006  4.46571E-006  8.7482
 0.00000E+000  3.27156E-007  5.81678E-007  5.07427E-007  7.15404E-007  3.97621E-006  9.73524E-006  2.0548
 0.00000E+000  6.41676E-007  1.12300E-006  1.21800E-006  2.26615E-006  8.41034E-006  1.73899E-005  4.6503
 0.00000E+000  7.75648E-007  1.47682E-006  2.28475E-006  5.69845E-006  1.66589E-005  2.88089E-005  8.2279
 0.00000E+000  3.66530E-007  1.37129E-006  4.07708E-006  1.19842E-005  2.87590E-005  4.05819E-005  6.3685
 0.00000E+000 -3.34949E-007  1.13705E-006  6.07101E-006  1.74831E-005  3.49638E-005  3.66188E-005 -1.6786
 0.00000E+000 -2.60464E-006 -3.42205E-006 -1.52714E-006  4.66920E-006  1.11725E-005 -6.96961E-006 -7.5728
 0.00000E+000 -1.40709E-005 -2.72479E-005 -3.85360E-005 -4.51006E-005 -5.18796E-005 -8.33064E-005 -1.4873
 0.00000E+000 -3.75397E-005 -6.99168E-005 -9.20646E-005 -1.00174E-004 -1.06144E-004 -1.38601E-004 -2.0108
 0.00000E+000 -4.23924E-005 -7.17707E-005 -8.31548E-005 -8.48550E-005 -1.07436E-004 -1.88406E-004 -3.3546
 0.00000E+000  8.26309E-006  1.89409E-005  1.96574E-005 -1.97545E-005 -1.39786E-004 -3.63653E-004 -6.5356
 0.00000E+000  6.84080E-005  1.05202E-004  7.88253E-005 -3.98659E-005 -2.58448E-004 -5.28568E-004 -7.3286
 0.00000E+000  6.18233E-005  7.99362E-005  3.46772E-005 -6.85105E-005 -1.81031E-004 -2.07156E-004 -1.0833
 0.00000E+000  7.16924E-005  1.25013E-004  1.63260E-004  2.12892E-004  3.27253E-004  5.15498E-004  6.1250
 0.00000E+000  2.04935E-004  3.84433E-004  5.37259E-004  6.78600E-004  7.99489E-004  7.85964E-004  4.7448
 0.00000E+000  2.56954E-004  4.55402E-004  5.79582E-004  6.18078E-004  5.36540E-004  3.05633E-004  4.8510
 0.00000E+000  2.20070E-005  1.04412E-005 -3.18961E-005 -7.31545E-005 -5.18419E-005  8.41416E-005  3.9649
 0.00000E+000 -2.40608E-004 -4.23292E-004 -4.78464E-004 -3.42779E-004 -2.78389E-005  4.03745E-004  1.0637
 0.00000E+000 -1.90544E-004 -2.76099E-004 -2.17651E-004 -4.48200E-005  1.91901E-004  5.26116E-004  1.0124
```

Fig. 3. The 3D data file with header

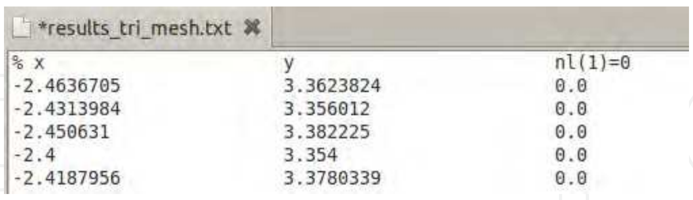
In this case, instead of just skipping the header block, we can extract useful information from it. The following code demonstrates how to read and analyze the header info when drawing the field distribution computed by RSoft:

```
%Opening the data file
fff=fopen('field.fld');

%Skipping two lines containing picture formatting info
%(namely, two strings separated by the <CR>)
fscanf(fff,'%s',2);

%Reading contents of the numerical part of the header into
%variable 'field_info'
field_info=fscanf(fff,'%i %f %f %f OUTPUT_REAL_3D\n %i %f %f\n',7);

%using the variable 'field_info' to read properly formatted
%data on the field distribution
field=fscanf(fff, '%f', [field_info(5), field_info(1)]);
```

Here, after opening the file, we are skipping two strings which mean nothing for us. Then we read all the numerical values into the array and can use this variable to determine the dimension of the array, the physical size of the computation region, etc.

3.3 Reading the random data

Using the header to define the parameters of the figure is not always enough. Particularly, it works for the uniformly distributed data points such as in FDTD method. However, if you use FEM, for example, Comsol Multiphysics, you will deal with non-uniform triangular mesh. In this case, the data is organized into structured. Each unit contains X and Y coordinate (in case of 2D model) or X, Y and Z coordinate (in case of 3D physical model) and the value of the field intensity corresponding to this point (see figure 4).



```
% x                    y                    nl(1)=0
-2.4636705             3.3623824            0.0
-2.4313984             3.356012             0.0
-2.450631              3.382225             0.0
-2.4                   3.354                0.0
-2.4187956             3.3780339            0.0
```

Fig. 4. The file containing random data linked to the specific points of the mesh

Reading of the date in this case is performed just as in case of the plane data, however, one have to fill two or three coordinate arrays X, Y and possibly Z, and the array of corresponding values of the field intensity.
It is also possible to convert this data into a regular mesh data which will be described in following section.

3.4 Reading structured data

In case of MPB and MEEP software, the field distribution is being output in so-called HDF5 format which is supported by many visualization tools as well as programming languages.

Particularly, in MATLAB there is a set of functions providing correct work with the HDF5. The MATLAB command HDF5Read(...) reads the contents of the file into the structure from where it can be easily extracted by standard structure addressing command (in case of MATLAB it is the dot operator). Here is simple MATLAB program which reads results computed by MEEP from HDF5 file and analyzes parameters of the results.

```
%Program allows reading from the the .h5 file containing the results of the
%field distribution computed with MEEP

clear all

%The function hdf5info allows obtatining the information from the .h5 file
%required to read the actual data
hinfo=hdf5info('/home/igor/temp/wg_bend−ez.h5');

%Then we are reading the information using the query string from the hinfo
%structure
res=hdf5read(hinfo.GroupHierarchy.Datasets(1));

%In this particular case, the computation results contain several frames
%with field distribution at different moments of time. The cycle reads them
%from the data structure and then draws.
for i=1:(size(res))(1)
  %Copying the frame into a separate 2D array
  drawres(:,:)=res(i,:,:);

  %Drawing the frame
  image(drawres∗1000);
  colormap('gray');

  %Drawing immediately
  drawnow;
end
```

However, as it have been described in the previous section, the data in the files may be represented in the forms of an arbitrary structures which requires a lot of work to dig in and to write corresponding unique decoders.

3.5 Reading multiple files data

In many cases, when you perform the parametric computations, the results have form of multiple files. Each file corresponds to a single value of the parameter and may have one of the formats described earlier.
The main problem in this case is to gather the data from all the files into a single data structure. This problem will be discussed in the following section. Here we will focus on how to open multiple files. The following piece of code shows how to open the files which have similar names with a counter attached to them:

```
%Scanning over all the numbers of the files
for num=0:90

  name=strcat(path,'metadata',num2str(num,'%02i'),'.dat')


  clear temporal_response;
  temporal_response=load(name,'-ASCII');

%Here performing all the operations we need
... ...

end
```

We presume to have file names in the following form: “metadata00.dat”, “metadata01.dat”, “metadata02.dat”,…,“metadata90.dat”. In this code, the outer loop scans over all the numbers from 0 to 90. The function strcat() allows concatenating several strings into one and generate the name. Using the function num2str() we convert the numerical type into a string.

The problem, may appear with the filenames since for the numbers lower than 10 we have zero-filler in the name. We can fix the problem in two different ways. The first one is just to compare the variable ’num’ with 10 and if it’s lower than 10, just to add ’0’ to the string manually. However, this is a solution only in particular cases. Suppose, you have numbers from 0 to 100000. In this case, you will have to make 5 different comparisons each for one digit ($num < 10$, $num < 100$, $num < 1000$, $num < 10000$, $num < 100000$). This is already a sort of annoying and far from elegance.

The second possible solution is to use formatted conversion of the number. In this case we just have to use format string “%02i” to fill the spaces before the number with zeros. Here the symbol of “%” tells MATLAB about the beginning of the format string, following “0” makes MATLAB to fill the spaces with zeros, “2” tells exactly how many digits it is suppose to be in the number and “i” indicates integer data type. Detailed information about the string formatting can be found in the MATLAB documentation in the section of the function fprintf().

4. Processing plane data

The plane numerical data is represented by two arrays, one of which contains values along one coordinate axis while another one corresponds to another axis. This simplest form of the data representation is actually the most clear and understandable when you want to present your results.

However, in some cases direct plotting of the data is not really representable. Particularly, when you make an expensive experimental research like temporal response measurement of the nonlinear PhC using ultra-short pulses, or when you perform the computation which is so time- and resource-consuming that you are unable to obtain sufficient number of data points like, for instance, computation of the photonic band gap maps. In these cases, before to create the data output, you have to convert it, improve it or make some other handling. This section is dedicated to the manipulations on the data which allow to improve its representation.

4.1 Data interpolation

The interpolation is the most useful operation on the data obtained during the experimental research on numerical computation. In the mathematical subfield of numerical analysis, interpolation is a method of constructing new data points within the range of a discrete set of known data points. There are a lot of different ways to interpolate the data like linear interpolation, polynomial interpolation, spline interpolation, etc. We will not consider mathematical side of the interpolation process concentrating on the implementation of this operation in MATLAB.

For the interpolation of the plane data, MATLAB uses the function interp1(). As an input arguments, the function has initial X and Y values of the data as well as new values of X at which you want to obtain the new values. It is also possible to select the interpolation method. As an output, this function returns the new values of Y found by interpolation process.

Here is the simple example of the data interpolation. In this example, we have the dependence of the photonic band gap width of the 1D PhC on the layer width. The data points are calculated for the limited number of values of the witdh. However, due to interpolation, we have smooth curve which looks much nicer.

```
%The program demonstrates different interpolation techniques
%performed by MATLAB. The interpolation is applied
%to the dependence of the photonic band gap width of 1D PhC on the
%layers witdh

%Loading the information on the PBG width from the file
data=load('pbg_width.dat','-ASCII');

%Getting the values of X from the data array
x=data(:,1)';

%Getting the values of Y for the second band (third column)
%in the datafile
y=data(:,5)';

%Creating the mesh 10 times more dense which is required
%for interpolation
x1=interp1(1:length(x),x,1:0.1:length(x));

%Creating new figurefigure;
%Drawing at the same figure
hold on;
%Using the function interp1, interpolating the data with nearest values
y1=interp1(x,y,x1,'nearest');
%Plotting the initial data with blue round markers
plot(x,y,'bo','LineWidth',2);
%Plotting the interpolated data with grin solid line
plot(x1,y1,'g','LineWidth',2);

%Doing the same with linear interpolation
```

```
figure;
hold on
y1=interp1(x,y,x1,'linear');
plot(x,y,'bo','LineWidth',2);
plot(x1,y1,'g','LineWidth',2);

%Doing the same with spline interpolation
figure;
hold on;
y1=interp1(x,y,x1,'spline');
plot(x,y,'bo','LineWidth',2);
plot(x1,y1,'g','LineWidth',2);

%Doing the same with cubic interpolation
figure;
hold on;
y1=interp1(x,y,x1,'cubic');
plot(x,y,'bo','LineWidth',2);
plot(x1,y1,'g','LineWidth',2);
```

As a result of the program we obtain four figures each for one interpolation type.

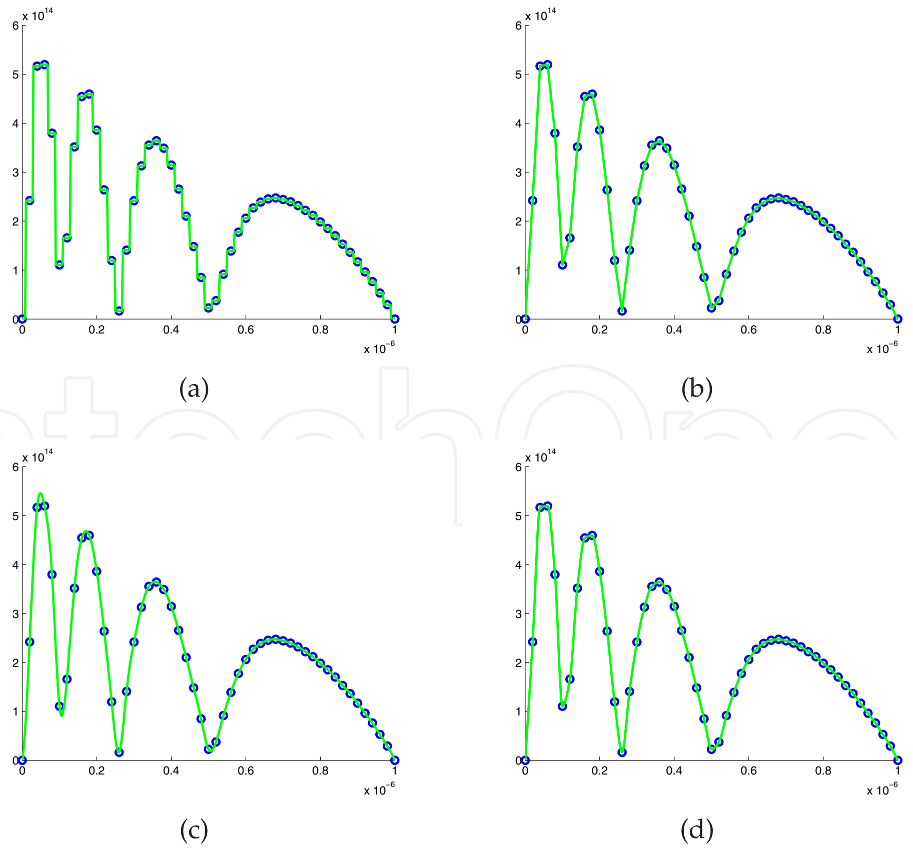


Fig. 5. Examples of basic interpolation. a) Nearest, b) linear, c)spline and d) cubic

In the program, the interpolation is performed in four different ways provided by MATLAB (actually, there are also “pchip” and “v5cubic” types. However, pchip is similar to cubic and v5cubic is just an old version of cubic). From the results, it is seen that for the selected curve the best interpolation technique is a spline interpolation.

4.2 Data approximation

Data approximation is an operation which allows to describe (approximate) the tabulated data by an analytical function.
The approximation is always used when the general view of the analytical dependence can be predicted, however, the coefficients still remain unknown. This is usually happens in the experimental researches when it is necessary to determine the parameters of the analytical dependence. Particularly, when the experiment is intent to determine the effective refractive index of the PhC by measuring the radiation deviation at different wavelength. After the experiment, the data is approximated by parametric analytical function where the parameter is the effective refractive index. The best approximation result gives the value of the mentioned parameter.
In MATLAB there is a separate toolbox providing the approximation of the tabulated data, namely, the Curve Fitting Toolbox. And again, using of the toolbox is possible in a single case only. However, using the toolbox requires definite time and when you need to perform multiple operations, this turns into a mess.
In this section we describe how to approximate the data in MATLAB without using the toolbox. The following program generates randomly five points and then approximates them with polynomial function of the fourth order.

```
%This function approximates five random points with 4−th order
%polynomial function. Since 4−th order polynom allows exact approximation,
%the error should be absent.

function approximation_plane
%Creating five X values
x=1:5;

%Randomly generating 5 Y values
y=rand(1,5)∗10;

%Creating the initial values for approximation process
start=[1 1 1 1 1];

%Starting five minimization interations
for i=1:5
  %Using the function fminsearch() to optimize the 'coefs' a the selected
  %points (X, Y)
  estimated_coefs = (fminsearch(@(coefs)fitfun(coefs,x,y),start));

  %changing the starting values to the optimized ones and then starting
  %over
  start=estimated_coefs;
```

```
end

%Building the smooth analytical function with coefficients obtained during
%optimization process
x1=0:0.1:6;
y1=analyt_fn(estimated_coefs,x1);

%Creating the figure
figure;
hold on;
%Plotting the initial points
plot(x,y,'bo','LineWidth',2);
%and the smooth analytical function
plot(x1,y1,'g','LineWidth',2);

%In the title writing the final polynom
title(strcat('y=',num2str(estimated_coefs(1)),'\cdotx^4+(',...
  num2str(estimated_coefs(2)),')\cdotx^3+(',...
  num2str(estimated_coefs(3)),')\cdotx^2+(',...
  num2str(estimated_coefs(4)),')\cdotx+(',...
  num2str(estimated_coefs(1)),')'),'FontSize',16);


%Function for optimization
function err=fitfun(coefs,x,y)
  %The function returns the difference between the analytical function
  %with coefficients 'coefs' at points X and the actual values of Y
  err=norm(analyt_fn(coefs,x)-y);
end

%Analytical parametric function to be optimized
function analyt_fn=analyt_fn(coefs,x)
  analyt_fn=coefs(1)*x.^4+coefs(2)*x.^3+coefs(3)*x.^2+coefs(4)*x+coefs(5);
end
end
```

The result of the function work is presented in figure 6. It can bee seen that approximation is quite accurate.

For the optimization procedure, MATLAB uses simplex search method which is implemented in the form of function fminsearch(). As an input parameters it has a pointer to a function to be minimized and the initial values of the parameter. In our case, the function to minimize returns the difference between actual points and the analytical function which, in turn, is defined in the function analyt_fn().

It should be paid attention to the fact that in current example the polynomial function of the fourth power approximates five data points. In this case, there is only one strict solution. However, in most cases, the approximation is made with certain accuracy which should be properly estimated. The best way to estimate the accuracy is to measure minimum, maximum and avarage deviation.

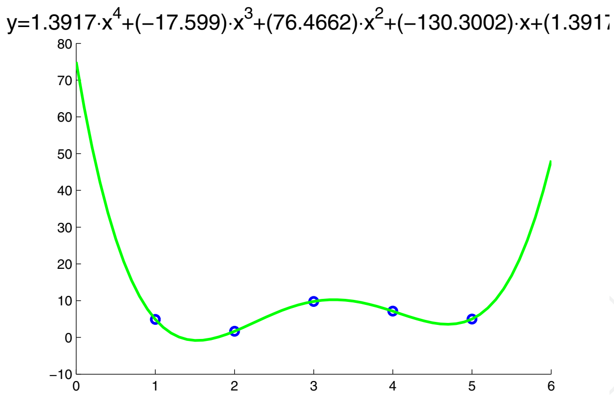


Fig. 6. Result of approximation of the data by the polynomial function

As an example, we give the code which approximates the dependence of central frequency of the photonic band gap of 1D PhC on the layer width, with hyperbolic function and estimates the approximation error. The data is obtained by means of plane wave expansion method Sukhoivanov & Guryev (2009) with low accuracy. With approximation it is possible to determine the analytic function describing such a dependence.
Consider the following code:

```
%This function approximates the dependence of the central
%wavelength of the photonic band gap of 1D PhC on the layer
%width with hyperbolic function.
%It also performs the error estimation.


function approximation_plane_err_estimation
%Loading information about the central frequency of the
%bandgap of 1D PhC from the file
data=load('pbg_center.dat','-ASCII');

%Getting the values of X from the data array
x=data(:,1)';

%Getting the values of Y for the second band (third column)
%in the datafile
y=data(:,3)';

%Creating the initial values for approximation process
%Now there's only four values since we use 3-rd order polynom
start=[1e14 0];



%Using the function fminsearch() to optimize the 'coefs' a the selected
%points (X, Y)
estimated_coefs = (fminsearch(@(coefs)fitfun(coefs,x,y),start));
```

```
%Building the smooth analytical function with coefficients obtained during
%optimization process
x1=interp1(1:length(data(:,1)),data(:,1),1:0.1:length(data(:,1)));
y1=analyt_fn(estimated_coefs,x1);

%using the function 'find' searching for indices of the refined X values
%which are equal to the initial ones
[row est_ind]=find(meshgrid(x1,x)-meshgrid(x,x1)' == 0);

%Searching for minimum deviation of the analytical function from the initial
%data (using est_index to compare corresponding points)
minerr=min(abs(y1(est_ind)-y));

%In the same manner, searching for maximum deviation
maxerr=max(abs(y1(est_ind)-y));

%Calculating avarage error
avgerr=sum(abs(y1(est_ind)-y))/length(y)


%Creating the figure
figure;
hold on;

%plotting the solution
plot(x,y,'bo','LineWidth',2);
%and the smooth analytical function
plot(x1,y1,'g','LineWidth',2);


%Writing the error values at the figure
text(min(x1),min(y1)+(max(y1)-min(y1))/10,...
   strcat('Minimum error = ',num2str(minerr,'%.2e')),'FontSize',14);
text(min(x1),min(y1)+(max(y1)-min(y1))/10*2,...
   strcat('Maximum error = ',num2str(maxerr,'%.2e')),'FontSize',14);
text(min(x1),min(y1)+(max(y1)-min(y1))/10*3,...
   strcat('Avarage error = ',num2str(avgerr,'%.2e')),'FontSize',14);

%Creating the title to the figure
title(strcat('y=',num2str(estimated_coefs(1),'%.2e'),'/(x+',...
   num2str(estimated_coefs(2),'%.2e'),')'),'FontSize',16);


%Function for optimization
function err=fitfun(coefs,x,y)
```

```
    %The function returns the difference between the analytical function
    %with coefficients 'coefs' at points X and the actual values of Y
    err=norm(analyt_fn(coefs,x)-y);
end

%Analytical parametric function to be optimized
function analyt_fn=analyt_fn(coefs,x)
    analyt_fn=coefs(1)./(x+coefs(2));
end
end
```

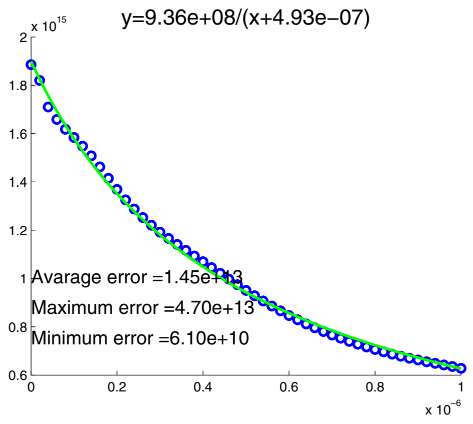


Fig. 7. Displaying the approximation accuracy information

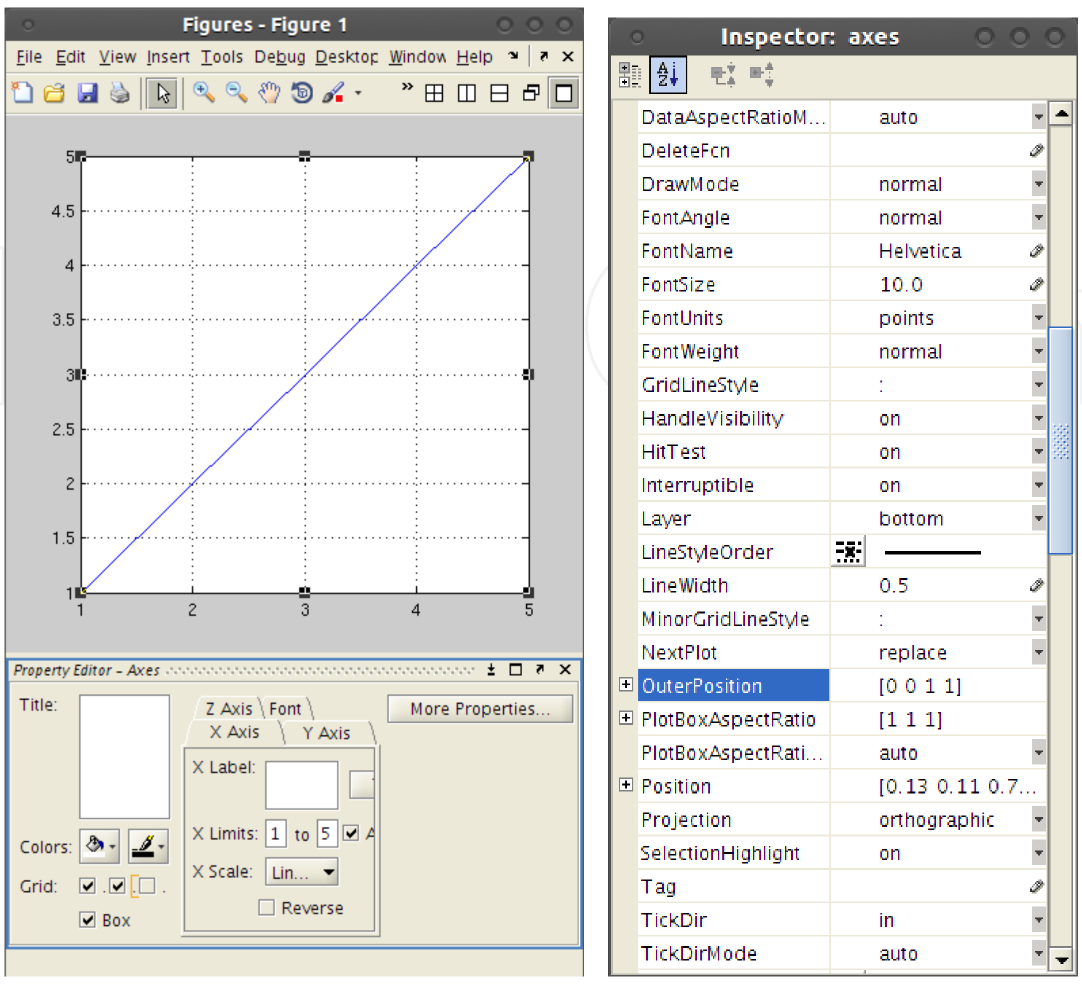
Now the result is represented in the form shown in the figure 7. We currently used the hyperbolic function which gave good agreement with numerical results. In this case, not all the points (in general case, neither of them) are fall at the analytical curve. Thus, estimating the error, we are able to confirm the validity of the approximation. In case of large error value, the function is unable to approximate the data at any conditions. This may happen when you have selected wrong kind of approximating function or, in case of too complex function, have selected wrong values of the initial parameters.

4.3 Data representation

In the scientific publication the data representation plays main role when it comes to correct understanding of the computation or experimental results. In two previous subsections we have described the way for data interpolation and approximation which helps to perform the data analysis as well as smooth the data. However, the data presentation is also important. Some basic techniques have already been used earlier without explanation such as manipulation of the figure's and plot's parameters. In this subsection we will try to describe and explain the most useful of them.

First thing which should be known about MATLAB, it is the object-oriented environment. This means that after creating some object such as plot, text, line, etc., they can be modified and their properties can be easily changed.

As always, the most simple way to manipulate the objects parameters is to use visual environment provided by MATLAB. For this reason, it is enough just to double-click at some object while editing mode is active (for this, you have to press the mouse pointer icon at the



X Limits: 1 to 5

DataAspectRatioM...	auto
DeleteFcn	
DrawMode	normal
FontAngle	normal
FontName	Helvetica
FontSize	10.0
FontUnits	points
FontWeight	normal
GridLineStyle	:
HandleVisibility	on
HitTest	on
Interruptible	on
Layer	bottom
LineStyleOrder	———
LineWidth	0.5
MinorGridLineStyle	:
NextPlot	replace
⊞ OuterPosition	[0 0 1 1]
⊞ PlotBoxAspectRatio	[1 1 1]
PlotBoxAspectRati...	auto
⊞ Position	[0.13 0.11 0.7...
Projection	orthographic
SelectionHighlight	on
Tag	
TickDir	in
TickDirMode	auto

(a) (b)

Fig. 8. Visual editing of the object's properties

tool panel). After this, you will see the editing panel which allows editing some parameters of the selected object (see figure 8(a)).
Deeper understanding of the parameters of the objects can be achieved when opening additional parameters panel or properties inspector (see figure 8(b)). Here you can see and edit all the parameters available for the object.
However, editing parameters inside your script is the most fast and productive way to change the figure appearance (imagine that you have to redraw the figure over and over and modify dozens of parameters manually).
There are two ways to set up the parameters of the objects. The first one is to set them on the stage of their creation. And the second one is modify them after the object is created with default properties.
In the first method, the parameters are transfered as arguments to the function which creates an objects like in following code:

```
%Approximation process
...
...
...
```

```
%Plotting the initial data
plot(x,y,'LineWidth',2,'LineStyle','none','Marker','o','Color','k');
%Plotting the approximated data
plot(x1,y1,'LineWidth',2,'LineStyle','--','Color',[0 0.3 1]);

%Adding the text marks
text(min(x1)+0.1,min(y1)+(max(y1)-min(y1))/10,...
  strcat('Minimum error = ',num2str(minerr)),'FontSize',14);
text(min(x1)+0.1,min(y1)+(max(y1)-min(y1))/10*2,...
  strcat('Maximum error = ',num2str(maxerr)),'FontSize',14);
text(min(x1)+0.1,min(y1)+(max(y1)-min(y1))/10*3,...
  strcat('Avarage error = ',num2str(avgerr)),'FontSize',14);
```

Here we show more advanced plotting from the approximation example. Pay attention that after plotting the data (X-values and Y-values), we place a lot of textual and numerical parameters. Parameters are usually defined by a pair name-value. The value can be a string, a number or an array depending on the parameter's requirements.
Particularly, the first plot function will draw the data with the LineWidth=2 (initially its 0.5), without the lines connecting the data points (LineStyle=none), but with circular markers (Marker=o) drawn with black color (Color=k).
The second plot draws the approximated data with LineWidth=2, with dashed line (LineStyle='–') and with color defined by three RGB values (each value varies from 0 to 1).
The text marks have only the parameters Fontsize=14.
As we have mentioned previously, there is another way to change the parameters of the objects using the object identifier. Note that each function which creates an objects, also returns an object identifier which can be stored into a variable and used lately to manipulate the object. Particularly, the properties of the object can be easily changed using the function set() which has as a parameters the object identifier and the set of object's properties. Consider this simple example which makes the same as previous one but modifying the properties after the object is created:

```
%Approximation process
...
...
...

%Plotting the initial data
p1=plot(x,y);
set(p1,'LineWidth',2,'LineStyle','none','Marker','o','Color','k');

%Plotting the approximated data
p2=plot(x1,y1);
set(p2,'LineWidth',2,'LineStyle','--','Color',[0 0.3 1]);

%Adding the text marks
t1=text();
set(t1,'Position',[min(x1)+0.1,min(y1)+(max(y1)-min(y1))/10,0],...
    'String',strcat('Minimum error = ',num2str(minerr)),...
    'FontSize',14);
t2=text();
set(t2,'Position',[min(x1)+0.1,min(y1)+(max(y1)−min(y1))/10∗2,0],...
    'String',strcat('Maximum error = ',num2str(maxerr)),...
    'FontSize',14);
t3=text();
set(t3,'Position',[min(x1)+0.1,min(y1)+(max(y1)−min(y1))/10∗3,0],...
    'String',strcat('Maximum error = ',num2str(avgerr)),...
    'FontSize',14);
```

This code does exactly the same as previous one, however with significant difference. When the function creating the object is called, the object identifier is stored into a corresponding variable. After this, the function set() is called to modify the objects properties. Pay attention, in case of text labels, the object is created with default properties. All the properties including the text strings are set later using the identifiers.
This code seems to be unnecessarily more awkward then the previous one but actually, it has one great advantage. After you stored the variable identifier, you are able to manipulate the object in any way you want. Particularly, you can easily create an animation by varying the parameters of the object (such as camera view) in a loop and saving each frame (for information about animations see the last section).

5. Simple 3D data processing

Operating with 3D data is more complicated than with plane one. Particularly, the most simple data to be work with is data organized into a regular 2D mesh such as results of FDTD computation. However, in some cases it is important to perform the data interpolation.

5.1 Data on rectangular mesh

The most frequently, the data organized into a rectangular mesh may be found in the field distribution output of the software implementing the FDTD method. However, each of the programs creates its own data format and, therefore, should be treated individually. Here as an example, we will give the programs reading the data from RSoft and MEEP.

5.1.1 Processing the data from RSoft

When the RSoft outputs the field distribution, it usually created two different files. The first one contains the field distribution and the second one contains the information about the structure boundaries. Therefore, it is necessary to read both files for correct representation the computed data.
Consider the following code for reading the RSoft field output:

```
%The program draws the field distribution computed by RSoft
%Over the field distribution it draws the structure which is
%read from the file generated by RSoft

clear all
```

```
%The variable 'name' contains filename without the extension
name='field_high_state'

%Opening the file containing the field info
fff=fopen(strcat(name,'.fld'));

%Skipping the inforation about the plotting
%which is unuseful for us
fscanf(fff,'%s',2);

%Into the variable 'field_info' reading useful information
field_info=fscanf(fff,'%i %f %f %f OUTPUT_REAL_3D\n %i %f %f\n',7);

%Reading the data points into 2D array. The dimensions of the
%array are taken from the variable 'field_info'
field=fscanf(fff,'%f',[field_info(5), field_info(1)]);

%Plotting the field with filled contour plot
figure;

%Again, the information about absolute coordinates is taken from the
%variable 'field_info'
contourf(field_info(2)*1e-6:...                    %from minimum X

    %with the step computed from the number of points over X
    ((field_info(3)*1e-6)-field_info(2)*1e-6)/(field_info(1)-1):

    field_info(3)*1e-6,... %to maximum X

    field_info(6)*1e-6:... %from minimum Y

    %with the step computed from the number of points over Y
    ((field_info(7)*1e-6)-field_info(6)*1e-6)/(field_info(5)-1):...

    field_info(7)*1e-6,... %to maximum Y

    %plotting the field data points without contour lines
    field,'LineStyle','none');

%Setting the color map
colormap('copper');

%Making MATLAB to draw at the same plot
hold on;

%Opening the file with the structure boundaries
fff=fopen(strcat(name,'.pdo'));
```

```
%Reading header with useful information about filling
[isfill count]=fscanf(fff,'fill %i\n');
[color count] =fscanf(fff,'color %i\n');
num_points=fscanf(fff,'polygon %i\n',1);
while(num_points)

        str=(fscanf(fff,'%f ',[2 num_points]))'*1e–6;
        fscanf(fff, 'end polygon\n',1);
        patch(str(:,1),str(:,2),zeros(num_points,1),'FaceColor',[1 1 1],...
              'EdgeColor',[1 1 1],'LineWidth',2);
        num_points=fscanf(fff,'polygon %i\n',1)
end
fclose(fff);
```

In this case, the first file "field_high_state.fld" is a simple 3D data with header containing information linking to the field distribution to real coordinate system, some auxiliary information about the number of data points, etc. Before writing the program reading and displaying such kind of data, it is useful to consult with the software manual and, which is very important, to open the file with text editor and check the header and data formats manually.
The second file "field_high_state.pdo" which contains information about the optical structure, contains structured data and should be read accordingly. Namely, each of the boundaries of the structure is stored in the form of polygon. The information about the polygon size is contained in the polygon's header. Before working with this file, it is also useful to take a look with a simple text editor.

5.2 3D data visualization

Unlike 2D data containing only two independent coordinates, the 3D data may be visualized in a variety of different ways. Depending on convenience, the following visualizing functions are most frequently used:

- surf()
- mesh()
- contour()
- contourf()
- image()

All the function actually take the same parameters, namely, X, Y arrays containing coordinates and Z 2D-array containing actual data.
Such kinds of visualization possess different functionality and speed which condition their usage. Particularly, if you need to express the intensity levels, the "contour()" function is the right choice. However, this function is too slow and trying to make animation with it requires a lot of time. The best choice for the animation is the function "image()".

6. Random data processing

In several cases, such as computation of the electromagnetic field distribution inside the PhC by means of FEM, the results appear within the nodes of irregular mesh. In this case it is impossible to perform the interpolation or other regular operation by means of standard MATLAB functions. In a professional software such as Comsol Multiphysics, this problem is overcome and now it is possible to export the results linked to a regular or user-defined mesh.
However, for example, in case of PDE Toolbox available in MATLAB, the results are exported in a form of random data where the specific value correspond to a node of triangular mesh. Dealing with this kind of data becomes a total mess.

6.1 Processing data on triangular mesh

In many cases, it is not necessary to output the data but we need to obtain some information from it. For instance, when one investigates the transmission characteristics of the microstructure, it is necessary to analyze the field distribution before the structure and after the structure. Namely, one knows exactly the regions of space where the field intensity should be found. In case of triangular mesh, the simplest way to do this is to find all the points within each region, the intensity in each point and then divide it by the number of points in the region. Consider the following code which finds average intensity within the region in space using the results exported from Comsol Multiphysics:

```
%Function for computation of the transmittance of the nonlinear
%photonic crystal from the field distribution computed in Comsol

clear all

%Loading the file with random data containing the field distribution
%inside the investigated structure
field=load('results_tri_mesh.txt','ASCII');

%Determining maximum and minimum values of coordinates
%Considering y−coordinate only since we are interested in
%the whole width of the structure
ymax=max(field(:,2));
ymin=min(field(:,2));

%Defining the limits for optical intensity calculation. Namely,
%from the bottom
y00=ymin;
%to bottom+3 microns
y01=ymin+3;
%from the top−3 microns
y10=ymax−3;
%to the top
y11=ymax;

%Counter of nodes in the bottom region (corresponding to reflection)
```

```
count_reflect=1;

%Counter of nodes in the top region (corresponding to transmission)
count_transmit=1;

%Determinig the length of the field structure
len=length(field);

%Counter for different intensity values
count_intensity=1;

%The cycle for the intensity
for intensity=0:0.01:0.5
  %The variable to sum the field intensity in all the points of
  %the bottom region
  reflect_total=0;

  %The variable to sum the field intensity in all the points of
  %the top region
  transmit_total=0;

  %The cycle for all the points in the mesh
  for iii=1:len

    %If the point is inside the bottom region,
    if (field(iii,2)>y00)&&(field(iii,2)<y01)
      %and if it is not NaN value
      if(abs(field(iii,3))>=0)
        %Adding its absolute value to the total reflected field
        reflect_total=reflect_total+(abs(field(iii,2+count_intensity)));
        count_reflect=count_reflect+1;
      end
    end

    %If the point is inside the bottom region,
    if (field(iii,2)>y10)&&(field(iii,2)<y11)
      %and if it is not NaN value
      if(abs(field(iii,3))>=0)
        %Adding its absolute value to the total transmitted field
        transmit_total=transmit_total+(abs(field(iii,2+count_intensity)));
        count_transmit=count_transmit+1;
      end
    end
  end


  %Normalizing the field intensities by the number of
```

```
    %points
    reflect=reflect_total/count_reflect;
    transmit=transmit_total/count_transmit;

    %Assuming the total field intensity as a sum of reflectance
    %and transmittance, calculating the reflectance of the structure
    reflectance(count_intensity)=reflect/(reflect+transmit);
    intensities(count_intensity)=intensity;
    count_intensity=count_intensity+1;

end

%Plotting the result and formatting the plot
figure;
plot(intensities,reflectance,'LineWidth',2);
xlabel('Radiation intensity','FontSize',14,'FontWeight','bold');
ylabel('Transmittance','FontSize',14,'FontWeight','bold');
set(gca,'FontSize',12)
grid on;
```

The program, in general, is quite simple. The triangular mesh is written into a file as a number of point coordinates with corresponding field intensity. It is just necessary to find all the points falling at the region required and sum all the intensities within the region.

6.2 Converting into a regular mesh

To improve the data quality and perform regional analysis of the field distribution, it is more convenient to use the regular data array instead of random data. Therefore, in this section, we provide reader with useful program able to convert the triangular mesh into a regular mesh by performing simple linear 3D interpolation.

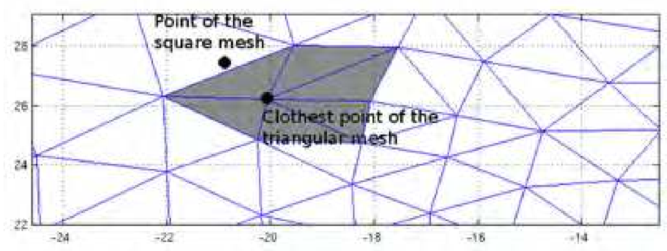
In PDE Toolbox, the data is always exported as one-dimensional or two-dimensional array (depending on the specific task). In case of two-dimensional array, each column corresponds to a single solution. It is also can be exported the mesh information. Particularly, we are interested in nodes coordinates and also particular interest represents the information about nodes which form the triangles. By default, this information is contained in the variables named "u" (the solution), "p" (coordinates of the nodes) and "t" (information about triangles formation) respectively. The information about the triangles is contained in the array "t" in a form of indices of the coordinate points stored in the array "p".

After exporting this information, we can now start converting the mesh using the following code:

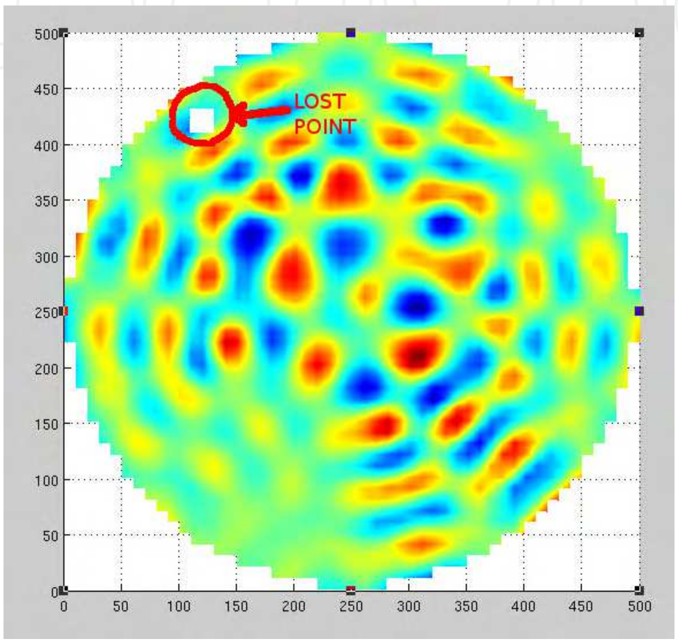
```
%The program converts the data on triangular mesh
%obtained in PDE Tool of the MATLAB into a data on square mesh
%The data from PDE Tool, namely, array of data points 'u',
%Data of the triangles of the mesh 't' and the array of
%coordinates 'p' should be previously exported to MATLAB workspace

clear data
clear datanew

%For convenience, we will use the 2D array to store the coordinates
%and corresponding data values
%First writing the coordinates of the points
data=p';

%and then, the values computed by PDE Toolbox in these points
data(:,3)=u(:,3)';

%Finding minimum and maximum values of the coorditates
xmin=min(data(:,1));
xmax=max(data(:,1));

ymin=min(data(:,2));
ymax=max(data(:,2));

%Defining the number of square mesh nodes in both dimensions
%Pay attention that the best way is to sellect the number of
%points in correspondence with the size on the structure in
%each dimension
xnpoints=50;
ynpoints=50;

%Generating coordinated of the rectangular mesh
xnew=xmin:(xmax-xmin)/xnpoints:xmax;
ynew=ymin:(ymax-ymin)/ynpoints:ymax;

%Now we have to find the data value in each of the points of the mesh
for xc=1:length(xnew)
  for yc=1:length(ynew)

            %To determine the point of the triangular mesh which is the closest
            %one to the current point of rectangular mesh, we compute the
            %Cartesian distances between them.
    len=(sqrt((data(:,1)-xnew(xc)).^2+(data(:,2)-ynew(yc)).^2));

            %Using the fucntion find(), we are searching for the minimum distance.
    i1=find(len==min(len), 1, 'first');
    if(len(i1)==0)
      datanew(yc,xc)=data(i1,3);
      continue;
    end

            %Now we are searching for all the triangles containing the point found

    [row, triangles]=find(t==i1);
    num_t=−1;

  %Then checking if any of the triangles contains the point of the
  %rectangular mesh
    for tc=1:length(triangles)
      %For this reason, calculating the area of the triangle
      area_mesh=abs(tri_area([p(1,t(1,triangles(tc)))...
                      p(2,t(1,triangles(tc)))], ...
                     [p(1,t(2,triangles(tc)))...
                      p(2,t(2,triangles(tc)))],...
                     [p(1,t(3,triangles(tc)))...
                      p(2,t(3,triangles(tc)))]));

      %and the sum of areas of three triangles formed by the
      %point of the square mesh and each two points of the
      %triangular mesh element
      area_sum=abs(tri_area([xnew(xc) ynew(yc)],...
                     [p(1,t(2,triangles(tc)))...
                      p(2,t(2,triangles(tc)))],...
                     [p(1,t(3,triangles(tc)))...
                      p(2,t(3,triangles(tc)))]))+...
            abs(tri_area([p(1,t(1,triangles(tc)))...
                      p(2,t(1,triangles(tc)))],...
                     [xnew(xc) ynew(yc)],...
                     [p(1,t(3,triangles(tc)))...
                      p(2,t(3,triangles(tc)))]))+...
            abs(tri_area([p(1,t(1,triangles(tc)))...
                      p(2,t(1,triangles(tc)))],...
                     [p(1,t(2,triangles(tc)))...
                      p(2,t(2,triangles(tc)))],...
                     [xnew(xc) ynew(yc)]));
   %The points is inside the triangular element if the sum of
   %the areas of three triangles is less or equivalent to the
   %one of the trianglular element
      if(abs(area_mesh−area_sum)<=0)
        %writing the number of the triangle
        num_t=triangles(tc);
        %and finising the cycle
        break;
      end
    end
    %if the trianle not found then the requested point is outside
    %the compoutation area
    if(num_t==−1)
      datanew(yc,xc)=NaN;
      continue;
```

```
        end

        %otherwise, reading the coordinate points and
        %the field values within these points
        x1=data(t(1, num_t),1);
        y1=data(t(1, num_t),2);
        z1=data(t(1, num_t),3);

        x2=data(t(2, num_t),1);
        y2=data(t(2, num_t),2);
        z2=data(t(2, num_t),3);

        x3=data(t(3, num_t),1);
        y3=data(t(3, num_t),2);
        z3=data(t(3, num_t),3);

        %Building the equation of the plane using vector
        %cross-product
        n=cross([x1-x2 y1-y2 z1-z2], [x3-x2 y3-y2 z3-z2]);
        A=n(1);
        B=n(2);
        C=n(3);
        D=-n(1)*x1-n(2)*y1-n(3)*z1;
        %and from the plane equation finding new value of the
        %field intensity within the point of square mesh
        datanew(yc,xc)=(-D-A*xnew(xc)-B*ynew(yc))/C;

    end
end

%Refining the mesh using the interp2() function
[xrefined yrefined]=meshgrid(xmin:(xmax-xmin)/ynpoints/10:xmax,...
                    ymin:(ymax-ymin)/ynpoints/10:ymax);
datanew_interp=interp2(xnew, ynew, datanew, xrefined, yrefined);

%And plotting the resulting field distribution
surfc(datanew_interp,'LineStyle','none');figure(gcf)

function tri_area=tri_area(p1,p2,p3)
    tri_area=0.5*(p1(1)*p2(2) + p1(2)*p3(1) + p3(2)*p2(1) - p3(1)*p2(2) -...
                  p3(2)*p1(1) - p2(1)*p1(2));
end
```

Here in this program we are using the PDE Toolbox data structures to demonstrate the mesh conversion technique. In case of using other computation software, the results output may differ from the one shown here.
In this case, the program generates the regular square mesh. Then it scans over all the points in the mesh and for each point searches the closest one of the initial triangular mesh. After



(a)



(b)

Fig. 9. Field distribution of the leaky mode of the PhC fiber containing the defect of triangular to rectangular mesh conversion

it finds the closest point, it uses the array "t" to find all the triangles containing this point. Then it checks which triangle contains current point of the rectangular mesh. For this reason it uses the method of the areas summation. If the one is not found, the point is outside the computation area and NaN is written into a resulting array. Otherwise, it builds the equation of the plane using the normal vector and one point. Then, using this equation, it calculates the value of the data at the required point.
The conversion of from the triangular mesh into a rectangular one has one serious drawback. Since, triangular mesh may be highly non-uniform, it is possible to lost some points independently on the meshes resolution. Consider situation presented in figure 9(a)
Here, two points mark the node of the rectangular mesh and the closest point of the triangular mesh. The grayed triangles are the ones containing the closest point. Pay attention that the node of the rectangular mesh does not belong to any of the triangles. In this case, the program puts NaN into this node, thus creating the hole in the resulting field distribution (see figure 9(b)).

This drawback, while being almost unessential for analysis, makes the method useless for data representation. In this case, the bes solution is to use something like neural networks interpolation which are presented in the following section.

7. Multiple files data processing

In most cases, you have the results in a form of a multiple files. For instance, when you computing the characteristics for the structure which parameters are varied during the computation process. In this case, each file corresponds to the specific value of the parameter.

7.1 Gathering the multiple files with MATLAB

The main problem in this case is to gather the files in a single data structure. This particular task depends on the kind of data. For example, in case of plane data you may want to plot the surface. However, when each file contains 3D data, the only possibility to visualize it is to make slices or animate the data. Particularly, when time-dependent field distribution is computed, it is wise to animate the results for better presentation.
In the first section we have considered how to open several files in a row. Here is the MATLAB program which gathers the data of the PhC band structures computed by RSoft into a single 3D array and plots the surfaces each one of which corresponds to a single PhC band. At this, the band structures of the PhC having different elements radius are suppose to be saved is a separate files with names like "bs_separate00_eigs_TE.dat".

```
%The program opens the files with the results of the band structure
%computation as a function of the refractive index
%obtained by RSoft and merges them into a single data structure.
%Then each band is displayed as a single surface

clear all;

%Parts of the filename
name='bs_single';
name_fin='_eigs_TE.dat';

%Cycle for the numbers of files
for nc=0:50

   %Forming the name
   full_name=strcat(name, num2str(nc,'%02i'), name_fin);

   %Opening the file
   f=fopen(full_name,'r');

   %Skipping the header
   skip=fscanf(f,'%s\n',5);

   %Reading the data from file into a 2D array
   data=fscanf(f,'%f ',[9 16]);
```

```
    %Closing the file
    fclose(f);

    %The first raw of the data corresponds to the X−values and is copied
    %Into array data_x
    data_x(1,:)=data(1,:);

    %the array data_y is formed automatically from number of file
    data_y(nc+1)=nc*0.01;

    %The actual 3D data is contained within the raws from 2 to 9 and
    %is being copied into corresponding 3D array
    data_z(nc+1,:,:)=data(2:9,:);
end

%Creating the figure
figure;

%Making MATLAB to add new plot to the existing figure
hold on;

%The cycle for the number of levels now nl scans over another dimension
%of the array (namely, to a band number).
for nl=1:8
    %Copying the data for a single band into a temporal 2D array
    draw_data(:,:)=data_z(:,nl,:);

    %Plotting the surface
    surf(data_x, data_y, draw_data,'MeshStyle','row');
end

%Creating the labels and decorating the plot
xlabel('Wave vector','FontSize',14,'FontWeight','bold');
ylabel('Radius r/a','FontSize',14,'FontWeight','bold');
zlabel('Frequency, \omegaa/2\pic','FontSize',14,'FontWeight','bold');
set(gca,'XGrid','on','YGrid','on','ZGrid','on',...
                  'CameraPosition',[7.9 −2.89 4.6]);
ylim([0 0.5]);
colormap('gray');
```

The function simply merges the multiple files into a single data structure and then plots separately each surface corresponding to each PhC band.

8. Building animations

Animations may become important when you are creating the presentation. In this case, dynamic representation of the results gives listener deep understanding of the problem.

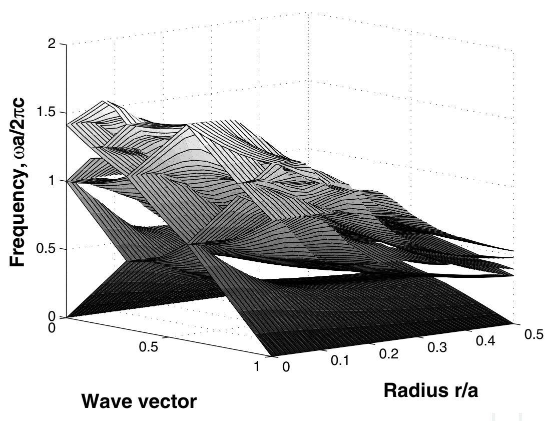


Fig. 10. The result of merging the data from multiple files

MATLAB provides users with methods for creating basic animations capturing frame by frame from a graph area. To record an animation, you first have to perform the graphic output of the results. To output the data in the real time, the function “drawnow” is required. Then, using the function “getframe” each frame is saved into an array which then can be animated in MATLAB or stored into a file to insert into the presentation.
As an example, we will consider previous case with the band structure computation and will animate the band structure in time instead of representing it in a form of surfaces. Consider the following program:

```
%The program reads the series of files from the specific location
%and builds the animation which is then written into a file

%Clearing the workspace
clear all;

%The first and the last constant parts of the name
name='bs_single';
name_fin='_eigs_TE.dat';
figure;

%The cycle through all the file numbers
for nc=0:50
  %Forming the filename
  full_name=strcat(name, num2str(nc,'%02i'), name_fin);
  %Reading the current file data
  f=fopen(full_name,'r');
  skip=fscanf(f,'%s\n',5);
  data=fscanf(f,'%f ',[9 16]);
  fclose(f);

  %Extracting the X data
  data_x(1,:)=data(1,:);

  %Extracting the Y datafor 8 bands
```

```
    data_y=data(2:9,:);

    %Drawing the bands
    for nl=1:8
        plot(data_x, data_y(nl,:),'k','LineWidth',2);
        hold on;
    end

    %Setting up the parameters of the plot
    ylim([0 1.5]);
    xlabel('Wave vector','FontSize',14,'FontWeight','bold');
    ylabel('Frequency, \omegaa/2\pic','FontSize',14,'FontWeight','bold');
    set(gca,'XGrid','on','YGrid','on');

    %Making MATLAB to display the plot immidiately
    drawnow;

    %Saving current figure as a frame
    ani(nc+1)=getframe(gcf);
    hold off;

end

%Playing back all the array of frames
movie(ani);

%Saving the animation into a video file
movie2avi(ani,'bs_on_radius');
```

Here in this program, three different kinds of animations in MATLAB are represented. Namely, using the command "drawnow", the animation is performed in the first time. Then, after it have been stored into an array "ani", the animation is played by the command "movie" which makes nothing more but displaying frame by frame the array. After this, the animation is stored into a ".avi" file which can be played separately in a video player or can be inserted into a presentation.

9. Conclusion

In this chapter, we have shared the experience of using the MATLAB for the photonic crystals modeling. Computation techniques considered in this chapter represent the examples aplicable to different fields of physics and electronics. The codes demonstrating the PhC computations, have been selected by authors following their experience and represent particular problems being solved during their investigation in the PhC field.
Particularly, there have been given the brief review of several well-knows software projects for PhC devices investigation, presented the techniques for working with different useful data formats for different kinds of the data representation; presented several working codes for reading and analyzing the data from RSoft, COMSOL multiphysics, MPB and MEEP;

processing data represented in different forms such as plane data, 3D (rectangular and triangular mesh) data, the data represented by several files.
The examples of working codes given in the chapter allow wide range of investigators and, in the first place, PhD and master students, to start quickly the solution of their own original problems.

10. References

COMSOL (2011).
URL: *http://www.comsol.com*

Joannopoulos, J., Meade, R. & Winn, J. (1995). *Photonic Crystals: Molding the Flow of Light*, Princeton University Press, Princeton.

Lourtioz, J.-M., Benisty, H. & Berger, V. (2005). *Photonic Crystals, Towards Nanoscale Photonic Devices*, Springer-Verlag.

MATLAB manual (2011).
URL: *http://www.mathworks.com/help/techdoc/*

MEEP (2011).
URL: *http://ab-initio.mit.edu/wiki/index.php/Meep_manual*

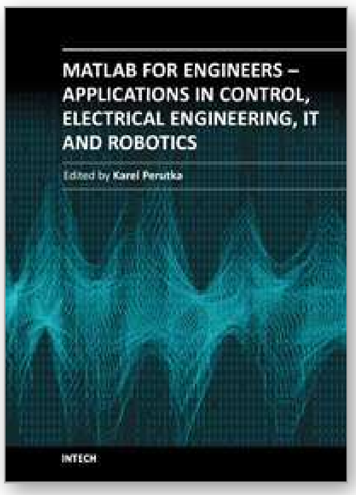
MPB (2011).
URL: *http://ab-initio.mit.edu/wiki/index.php/MPB_manual*

RSoft (2011).
URL: *http://www.rsoftdesign.com/*

Sakoda, K. (2001). *Optical Properties of Photonic Crystals*, Springer-Verlag.

Sukhoivanov, I. & Guryev, I. (2009). *Photonic crystals – physics and practical modeling*, Optical sciences, 1 edn, Springer, Berlin.

Taflove, A. & Hagness, S. (2005). *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3 edn, Artech House, Norwood, MA.


MATLAB FOR ENGINEERS – APPLICATIONS IN CONTROL, ELECTRICAL ENGINEERING, IT AND ROBOTICS
Edited by Karel Perutka
INTECH

ISBN 978-953-307-914-1
Hard cover, 512 pages
Publisher InTech
Published online 13, October, 2011
Published in print edition October, 2011