DISPHOTN v.0.1: User's Guide

v. 0.1

March 3, 2011

Contents

1 Introduction

DISPHOTN is a little software for the computation of the eigenfrequencies of the electromagnetic field in a 2D or 3D periodic structure composed of free-space, dielectrics and metals obeying the Lorentz permittivity model. The theoretical model underlying the calculation is based on a description of the problem using auxiliary mechanical fields that keep track of the electron response.

2 Installation

The program relies on the combination of the libraries `ARPACK` and `PARDISO` to perform simulations. These libraries, in turn, depends on `LAPACK` and `BLAS`. Moreover, since the language used for the libraries is `FORTRAN`, a suitable compiler should be used to compile those libraries. Once all the libraries required are obtained, the program can be compiled. Using the `gcc` compiler, the command issued could be:

```
gcc -c disphotn.c libpardiso.so libarpack.a lapack.a blas.a -lgfortran -lgomp
-o disphotn
```

This command compiles the C file and links the required libraries, producing the output binary file `disphotn`. Note that the `PARDISO` library is a dynamic library, and therefore must be made available to the program at runtime (e.g. in `/usr/lib`). The required packages can be found at the following addresses:

- `ARPACK`: `http://www.caam.rice.edu/software/ARPACK/`
- `PARDISO`: `http://www.pardiso-project.org/`
- `LAPACK` and `BLAS`: these software packages should be included in the `ARPACK` package.

3 `PARDISO` License

`PARDISO` is proprietary software, and is available only as a precompiled library. A valid license for `PARDISO` must be obtained, and the license file `pardiso.lic` must be placed in the same directory as the executable file. Please see `PARDISO`'s website (`http://www.pardiso-project.org/`) for more information, and to obtain a license.

4 Getting Started

To perform a typical simulation, two files must be provided by the user. A *configuration file*, that contains all the information and the parameters required by the simulation, and a *mesh file*, containing the spatial discretization (the mesh). The configuration file name is specified either as an argument in the command line, as an absolute or relative path, or is asked by the program at run-time. All the other informations (including the mesh file name) are contained in the configuration file.

4.1 The Configuration File

In the configuration file, the parameters required by the software must be specified. A configuration file for DISPHOTN is a simple plain text file that looks like this:

```
1 # Comment
2 parameter-name parameter-value
3 flag-name     # Other comment
4 sequence-name val1 val2 val3
```

The character # identifies a comment in the file. Everything after the # character will be ignored by the program. The second line shows how a parameter can be defined: the parameter name, space, and the paramenter value. Note that DISPHOTN recognizes numbers written both in decimal and in scentific notation. As an example, to set the parameter `sigmar`, one may write: `sigmar 102` or, equivalently, `sigmar 1.02e+2`.

Flags are boolean parameters. If the configuration file contains the name of the flag, the flag itself is set to *true*, otherwise it is defaulted to *false*. Sequences are lists of values with a certain order. The values are specified after the sequence name, leaving a blank space between them. The maximum number of values for a sequence is hardcoded to 300. Note that if you specify a parameter or a sequence name without writing any value, an error may occur. The following table presents the keywords that can be used in a configuration file:

Keyword	Type	Data Type	Usage
`meshfile`	P	String	**Required.** Specifies the mesh file to be used in the simulation. The value can be either a relative or an absolute path.
`nev`	P	Integer	**Required.** The number of eigenfrequencies requested.
`a1`	S	Float	**Required.** The first Bravais lattice vector. Only the first 2 (for 2D problems) or 3 (for 3D problems) values of the sequence are considered.
`a2`	S	Float	**Required.** The second Bravais lattice vector.
`a3`	S	Float	**Required. 3D only.** The third Bravais lattice vector.
`kx`	S	Float	The sequence of values for the x component of the **k** vector. The three sequences `kx`, `ky` (and `kz`) must have the same number of elements.
`ky`	S	Float	The sequence of values for the y component of the **k** vector.
`kz`	S	Float	**3D only.** The sequence of values for the z component of the **k** vector.
`xperiodic`	F		Set this flag to impose Bloch periodicity along the **a1** direction.
`yperiodic`	F		Set this flag to impose Bloch periodicity along the **a2** direction.
`zperiodic`	F		**3D only.** Set this flag to impose Bloch periodicity along the **a3** direction.
`zperiodic`	F		**3D only.** Set this flag to impose Bloch periodicity along the **a3** direction.
`ncv`	P	Integer	Number of Arnoldi vector used by `ARPACK`. If not specified, it is defaulted to 8*`nev`. Please see `ARPACK` documentation for more information.
`sigmar`	P	Float	The real part of the eigenvalue shift (the normalized target frequency around which look for eigenvalues). The default value is zero, however, due to the nature of the problem, a nonzero shift should always be specified.
`sigmai`	P	Float	The imaginary part of the eigenvalue shift. Usually it is better to leave this value zero.
`dielectric-domains`	S	Integer	The domains corresponding to a dielectric material. Each domain should appear only once.
`epsilon`	S	Float	The relative permittivity for the domains specified in the sequence `dielectric-domains`. This sequence and the `dielectric-domains` sequence must have the same length.

Keyword	Type	Data Type	Usage
`metal-domains`	S	Integer	The domains corresponding to a metallic material. Note that the same domain may appear more than once in the sequence. In this case, the domain will be modeled with a multi-pole Lorentz model.
`omega0`	S	Float	The resonance frequency for the domains specified in the sequence `metal-domains`.
`omegap`	S	Float	The plasma frequency for the domains specified in the sequence `metal-domains`. The last three sequences must have the same length. Note that a domain cannot appear both in the `metal-domains` and the `dielectric-domains` sequences.
`solver`	P	Integer	Specifies the solver used by `PARDISO`. 0 = iterative solver, 1 = direct solver (default). Please see `PARDISO` documentation for more information.
`bandsfile`	P	String	The output file in which to save the computed eigenfrequencies.
`modesfile`	P	String	The output file in which to save the electric field configuration.
`hfieldfile`	P	String	**2D only.** The output file in which to save the reconstructed magnetic field H_z.
`xgrid`	P	Integer	The size of the interpolation grid on the a1 direction.
`ygrid`	P	Integer	The size of the interpolation grid on the a2 direction.
`zgrid`	P	Integer	**3D only.** The size of the interpolation grid on the a3 direction. The default values for each grid size is 21.
`gridfile`	P	String	A file containing the points on which the electric field should be interpolated. Please see the section "Grid File" for more information. Note that setting this parameter overrides the values of the xgrid, ygrid and zgrid parameters.
`penalty`	P	Float	**2D only.** Penalty for the imposition of interface condition. The default value is 0.7. Please see the "Penalty" section for more information.

4.2 Parameters

An extensive description of some parameters follows.

- `nev`: this parameter specifies the number of eigenfrequencies to compute. The program will look for the eigenfrequencies around the specified shift expressed in the `sigmar` parameter. Note that due to the nature of the problem

4.3 Mesh File

DISPHOTN can read and import text mesh files created with COMSOL Multiphysics 3.5 and newer. The mesh file must follow some rules:

- The mesh file must be text files (`.mphtxt`). If using COMSOL Multiphysics 4.0 or newer, be sure to export the mesh as text file (not binary file).
- The recognized element types are only triangles (for 2D problems) and tetrahedra (for 3D problems). Other element types (quad, brick, etc.) are not recognized.
- The mesh must represent the unit cell of a Bravais lattice. One of the vertex of the unit cell must be in the origin, and the edges must be along the a1, a2, (and a3) vectors. This is very important in order to correctly identify the boundaries.
- The domain numbers to be specified in the sequences `metal-domains` and `dielectric-domains` are the same used in COMSOL.
- For periodic structures, the mesh must be EXACTLY the same on the two periodic boundaries. Use the "copy mesh" feature in COMSOL to obtain identical meshes.

Figure **??** presents a correct mesh for a 2D triangular lattice. Note that the values of a1 and a2 vectors, in this case, are $(1, 0)$ and $(0.5, \sqrt{3}/2)$

5 Program Description

5.1 Theoretical Background

The aim of the program is to compute the eigenfrequencies of the electromagnetic field for a periodic structure containing lossless dispersive materials. When computing eigenfrequencies, Maxwell's Equations are usually reformulated as an eigenvalue problem

$$\nabla \times \frac{1}{\mu_0} \nabla \times \boldsymbol{E} = \omega^2 \epsilon_0 \epsilon \boldsymbol{E}. \tag{1}$$

This eigenvalue problem is in general non-linear for a frequency-dependent dielectric permittivity. However, a linear eigenvalue problem can be obtained in the case of a permittivity that follows the lossless Lorentz's model,

$$\epsilon(\omega) = 1 + \frac{\omega_p^2}{\omega_0^2 - \omega^2}, \tag{2}$$

by introducing an auxiliary field. The field is called **P** since it is proportional to the material polarization (i.e. the electron displacement). This field keeps track of the electron response in the metal. Using this formulation, the eigenvalue problem becomes:

$$\omega^2 \begin{bmatrix} \boldsymbol{E} \\ \boldsymbol{P} \end{bmatrix} = \begin{bmatrix} \frac{1}{\epsilon_0} \nabla \times \frac{1}{\mu_0} \nabla \times \bullet + \omega_p^2 \bullet & -\frac{\omega_0^2}{\epsilon_0} \bullet \\ -\omega_p^2 \epsilon_0 \bullet & \omega_0^2 \bullet \end{bmatrix} \begin{bmatrix} \boldsymbol{E} \\ \boldsymbol{P} \end{bmatrix}. \tag{3}$$

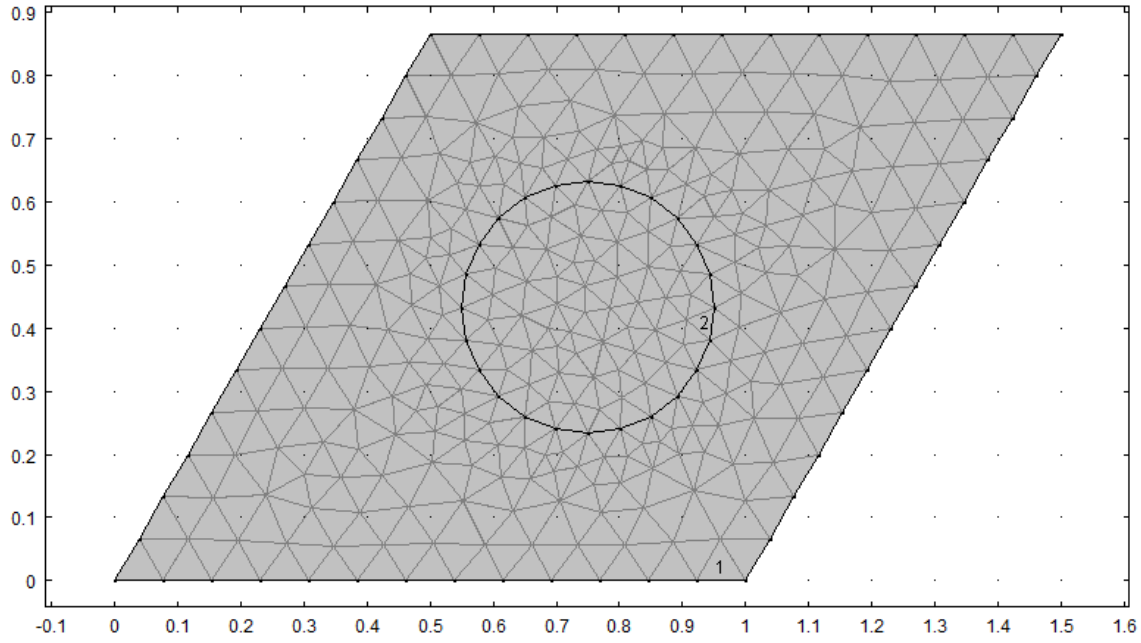

Figure 1:

This formulation is valid in the metallic regions. In the freespace or dielectric regions, the usual eigenvalue problem Eq. (**??**) can be used.

5.2 Domain and Boundary Conditions

Since the program is aimed to the simulation of periodic structure, the domain of the problem is assumed to be an unit cell of a Bravais lattice. This means that it must be spanned by two (2D) or three (3D) linearly independent vectors (the Bravais primitive vectors). As a consequence, the domain must always be a quadrilateral (2D) or a 6-faced polyhedron (3D). This makes easier to compute the photonic bands in a periodic structure. Note that the unit cell may actually contain anything, so even traditional non-Bravais lattices (such as a diamond lattice) can be simulated.

On such cells, two kinds of boundary conditions can be imposed: a Perfect Electric Conductor (PEC) condition, and Bloch Periodic conditions. Opposite boundaries share the same condition type, i.e. both of them are obviously either PEC or periodic. Setting the `x-`, `y-` and `zperiodic` flags in the configuration file imposes the periodicity on the three pairs of boundaries; if the flags are not set, the PEC conditions are enforced. The PEC condition is simply imposed by setting the tangential component of the electric field to zero on the boundary. The Bloch periodic condition establishes a relation between the field on opposite boundaries, according to Bloch's Theorem:

$$\boldsymbol{E}_{\boldsymbol{k}}(\boldsymbol{x}+\boldsymbol{R}) = e^{i\boldsymbol{k}\cdot\boldsymbol{R}}\boldsymbol{E}_{\boldsymbol{k}}(\boldsymbol{x}). \tag{4}$$

Using this relation, the value of the field on a boundary is proportional to the value of the field on the opposite boundary, with a phase coefficient dependent on the geometry of the unit cell and the $\boldsymbol{k}$ vector specified by the user.

5.3 Discretization

In order to discretize the problem, a Finite-Elements Method (FEM) is used, with edge-element type basis function. The domain of interest is subdivided into a mesh composed of triangles (2D) or tetrahedra (3D). The equations are then enforced in weak form on each mesh element. This transforms the differential eigenvalue problem in a matrix eigenvalue problem. The size of the matrices involved depends on some of the properties of the problem. The base matrix size is the number of edges in the mesh. For each metallic domain specified in the `metal-domains` sequence, the size of the matrix is increased of the number of edges contained in the domain. Note that if the same domain appears more than once in the sequence (for example, to simulate a multi-pole Lorentz model) its edges are counted multiple times. From this total, the number of edges on PEC boundaries must be subtracted, and half of the number of edges on periodic boundaries (since in this last case the field on each edge can be expressed in terms of the field on the opposite edge).

5.4 Interface Conditions and Penalty

The use of edge-based elements has some advantages and some disadvantages. The solutions obtained are guaranteed to have null divergence inside each domain, and the tangential continuity of the electric field is automatic. However, the other conditions at the interface (namely, the normal continuity of the dielectric displacement field) are difficultly implemented. The failure in the imposition of correct interface conditions at a metallic interface give rise to spurious modes, localized on the interface itself, that pollute the compute spectrum. To remove these modes, the edges on the interface are treated as if they belong interely to the metallic region; unfortunately, using this method alters the computed eigenfrequencies of the higher order surface modes (the ones more localized on the metallic interface), moving them to higher values. In 2D problems, the `penalty` parameter can be used to tune the interface condition. The default value for this parameter is 0.7; lower values give more correct eigenfrequencies, but values below 0.5 may cause spurious modes to occur. In general, finer meshes give correct eigenmodes even with lower `penalty` values. If the lowest order surface modes are of interest, and if a sufficiently fine mesh can be used, the `penalty` parameter can be set to a very low value (or zero) in order to obtain correct eigenfrequencies for the first modes. In 3D problems, due to performance reasons, the value of the `penalty` parameter is fixed at 1.

5.5 Band Structure Computation

The program is able to compute band structures for a periodic structure. To do so, simply specify the values of the $\boldsymbol{k}$ vector (k-points) at which the eigenfrequencies should be computed. The program will then compute `nev` eigenvalues for each k-point. All the eigenfrequencies and eigenmodes are saved in the results files. Note that the values of the `k` vector are chosen by the user, so they should be chosen accordingly to the geometrical properties of the structure. As an example, to compute the Γ-X-M-Γ band structure for the 2D square lattice spanned by the vectors $(1, 0)$ and $(0, 1)$, the `kx` and `ky` parameters may be specified as follows:

```
kx 0 .2 .4 .6 .8 1 1 1 1 1 1 .8 .6 .4 .2 0
ky 0 0 0 0 0 0 .2 .4 .6 .8 1 .8 .6 .4 .2 0
```

Doing so, the program will compute the eigenfrequencies at various k points going from $\boldsymbol{k} = (0, 0)$, to $\boldsymbol{k} = (1, 0)$, then to $\boldsymbol{k} = (1, 1)$, and finally back to $\boldsymbol{k} = (0, 0)$. The value of $\boldsymbol{k}$ affects the periodic boundary conditions. Those kind of conditions, therefore, need to be reimposed at every k-point. Note that if no boundary has a Bloch-type boundary conditions, the values in the `kx` and `ky` sequences are ignored and the simulation will only run once; anyway, the program

will not check if the specified k-points are valid for the structure.

5.6 Eigensolving

Once the matrices are built and the boundary conditions are imposed, the eigensolver is started. Two external libraries are used in the eigensolving process: the `ARPACK` eigensolver and the `PARDISO` linear system solver. `ARPACK` is an iterative solver that finds an approximation of the eigenvalues and the eigenvector of the matrices. `PARDISO` is a linear system solver used to solve the problem in shift-and-invert mode. These two libraries are chosen in order to reduce the memory consumption and allow the solution of large-scale problems. The user is granted a limited control on the actual eigensolving process. The `ncv` parameter, used by `ARPACK`, specifies the number of Arnoldi vectors to be used. This can affect in various ways the eigensolving; for more information, we refer to `ARPACK`'s documentation. The `solver` parameter is used by `PARDISO` and specifies whether an iterative (`solver` = 0) or direct (`solver` = 1) solver must be used. Once again, we refer to `PARDISO` documentation for more information.

5.7 Target Frequency

As mentioned, the parameter `sigmar` specifies a target frequency around which the eigenfrequencies should be computed. The choice of this parameter is of crucial importance in the simulation, directly influencing the eigenmodes found and the computation time. In order to choose the right target frequency, a general knowledge of the problem and the physics involved is needed. Moreover, the discretization using edge-elements introduces a great number of non-physical modes with frequency zero and at all the frequencies in which the Lorentz dielectric permittivity vanishes. Since these effectively count as very large eigenvalue clusters, `ARPACK` has problems when trying to compute eigenvalues at those frequencies. Therefore, the parameters `sigmar` and `nev` must be chosen accordingly to avoid trying to compute eigenvalues belonging to those clusters.

5.8 Postprocessing and Interpolation

Once the desired number of eigenvectors converged, an usable set of results must be computed. The eigenvectors obtained are the edge-elements coefficients and can be physically interpreted as the value of the fields on each edge in the tangential direction. However, this set of value is of little use, and must be transformed in a more useful form. The program performs an interpolation of the field on a discrete regular grid. The value of the Cartesian components of the field at each grid points are computed using the Finite Elements basis functions. There are two ways to specify an interpolation grid. The first is to use the built-in grid generator, that creates a regular grid of the same shape of the unit cell. The parameters `xgrid`, `ygrid`, and `zgrid` specify the size of the grid in the `a1`, `a2`, and `a3` directions (Note: not the Cartesian x, y and z directions!). Especially in the case of 3D problems, a large value for this parameters can result in a very large grid, and consequently a long interpolation time. Alternatively, the user may provide a text file containing the grid points on which the interpolation must be carried on. The file name must be specified in the `gridfile` parameter. If this parameter is set, the values of the `x-`, `y-` and `zgrid` parameters are ignored.

5.9 Grid File

The grid file is a plain text file. The first (non-empty or non-comment) line must contain a single integer value equal to the number of grid points contained in the file. The following lines must

contain the coordinates x y (or x y z) of the interpolation points, expressed as floats separated by a blank space. As with the configuration file, the # character introduces a comment. The filename (as an absolute or relative path) of the grid file must then be specified in the `gridfile` parameter in the configuration file.

5.10 Results and MATLAB functions

The program produces two series of data as a result of the computation. The eigenfrequencies computed at each k-point are saved in a text file, if a filename is specified in the `bandsfile` parameter. The file consists in a table with `nev` rows and a number of columns equal to the number of k-points required. Each column presents the eigenfrequencies computed for the corresponding k-point.

The computed eigenmodes of the electric field are stored in the `modesfile` file, if specified. For 2D (3D) problems, the first two (three) rows give the coordinates x, y and z of the interpolation grid points, separated. Then, the data is organized in a number of blocks equal to the number of k-points. In each block, the first `nev` lines give the interpolated values of the x component for each computed eigenmode at that specific k-point, followed by other `nev` lines containing the y component values, and (for 3D problems), other `nev` lines with the z component values. All the values of the field may be complex. Some code for MATLAB has been provided in order to allow a manipulation and visualization of the results. To import the `bandsfile` (the file containing the computed eigenfrequencies), simply use MATLAB's function `dlmread`. To import the results from the `modesfile`, use the function `disphotn_extract_mode_2d` (or `_3d`), that extracts the interpolated electric field for the specified eigenmode. For example, one can visualize a vector plot of the electric field of the i-th eigenmode of the j-th k-point in the following way. First, the grid coordinates and the field values are extracted from the result file:

```
[x y fx fy] = disphotn_extract_mode_2d(modesfile, nev, j, i);
```

and then plotted as a vector plot:

```
quiver(x, y, real(fx), real(fy));
```

For 2D problems, a third set of data is produced. The reconstructed z component of the magnetic field is stored in `hfieldfile`, if specified. The format of this file is the following: it contains a number of blocks equal to the number of k-points, and each block contains `nev` lines. Each line, in turn, contains the value of the Hz component of the magnetic field in each triangle on the mesh. The value can, in general, be complex. The magnetic field is not interpolated in order to take advantage of MATLAB's `pdeplot` function. In MATLAB, first import the mesh with the provided `disphotn_import_comsol_mesh` function:

```
[points edges triangles] = disphotn_import_comsol_mesh(meshfile);
```

retrieve the magnetic field value from the result file:

```
h = disphotn_extract_mode_h(hfieldfile, nev, k, modn);
```

and, finally, plot the Hz field:

```
pdeplot(points, edges, triangles, 'xydata', real(h));
```

6 Example

The following is an example of a 2D structure that can be simulated using DISPHOTN. The structure is a simple square lattice of cylinders, with the ratio between radius and lattice parameter equal to 0.25. The square lattice with unitary lattice parameter is defined by the primitive vectors $(1, 0)$ and $(0, 1)$. The mesh is represented in Figure **??**, contained in the file `square_lattice_example.mphtxt`. The metal will be described by a Drude model with normalized plasma frequency equal to one. To obtain a Drude model from a Lorentz model, it is neces-

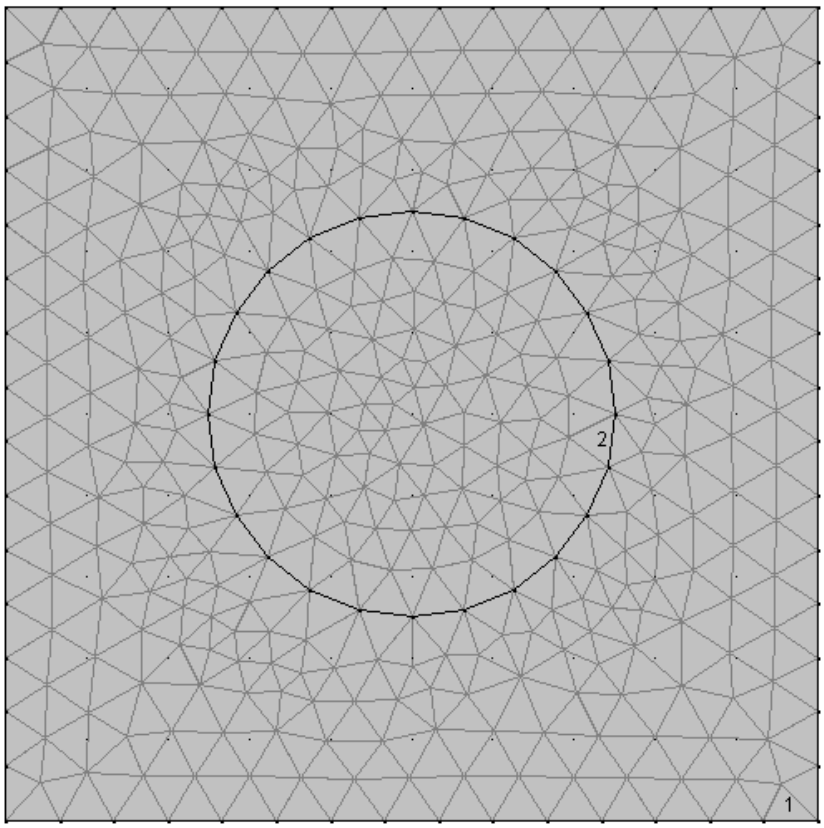

Figure 2: Mesh for the example problem in Sect. **??**.

sary to take the limit $\omega_0 \to 0$; in our simulation, we put $\omega_0 = 0.001$. We will ask the program to compute the complete photonic band diagram (Γ-X-M-Γ), finding 16 bands in the region of the lower order surface modes. The resulting configuration file is `square_lattice_example.txt`:

```
# Sample Configuration File for DISPHOTN.
# Name of the mesh file
meshfile square_lattice_example.txt

# Geometry of the lattice
a1 1 0
a2 0 1

# Periodicity
xperiodic
yperiodic

# Metallic domains and their properties
metal-domains 2
omegap 1
omega0 0.001
```

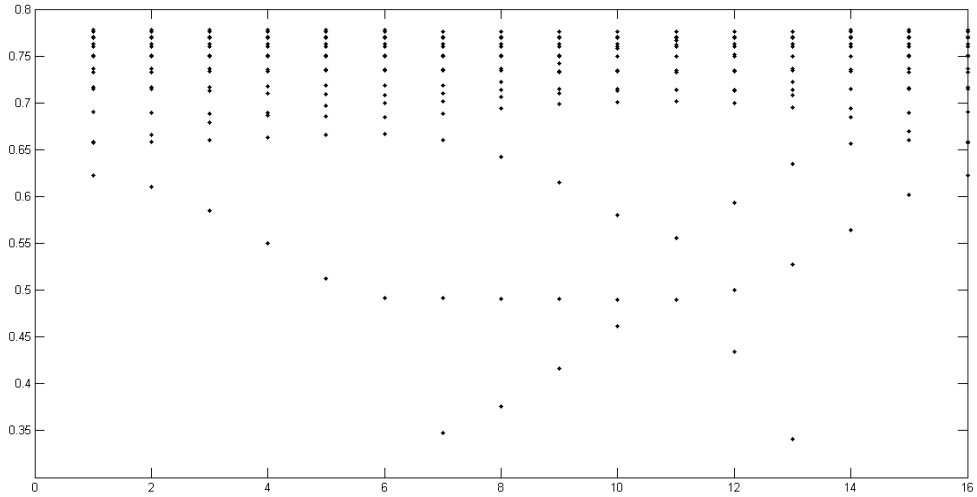

Figure 3: The band diagram obtained plotting the results with MATLAB.

```
# Band Structure computation
kx 0 .2 .4 .6 .8 1 1 1 1 1 1 .8 .6 .4 .2 0
ky 0 0 0 0 0 0 .2 .4 .6 .8 1 .8 .6 .4 .2 0

# Target Frequency
sigmar .6

# Number of requested eigenfrequencies
nev 16

# Output files
bandsfile example_bands.txt
modesfile example_modes.txt
hfieldfile example_hfield.txt
```

The simulation is then run by executing `disphotn` and passing the configuration file on the command line or inputting the name when prompted. The program will compute the band diagram, calculating 16 eigefrequencies around the normalized frequency of 0.6, varying the value of the $\boldsymbol{k}$ vector according to the values specified in `kx` and `ky`. At the end of the computation, the resulting eigenfrequencies are saved in the file `example_bands.txt`. To visualize the band diagram, one can import the data in MATLAB:

```
bands = dlmread('example_bands.txt');
```

and then plot it:

```
plot(bands.', 'k.');
```

The band plot is presented in Fig. **??**. As mentioned, due to the correction to spurious modes introduced, the surface plasmon modes (the flat bands) have frequencies higher than the expected convergency frequency ($1/\sqrt{2} \simeq 0.7071$). The eigenmodes of the magnetic field can be visualized as well, since we are dealing with a 2D problem. First, the mesh file and the H field values must be imported:

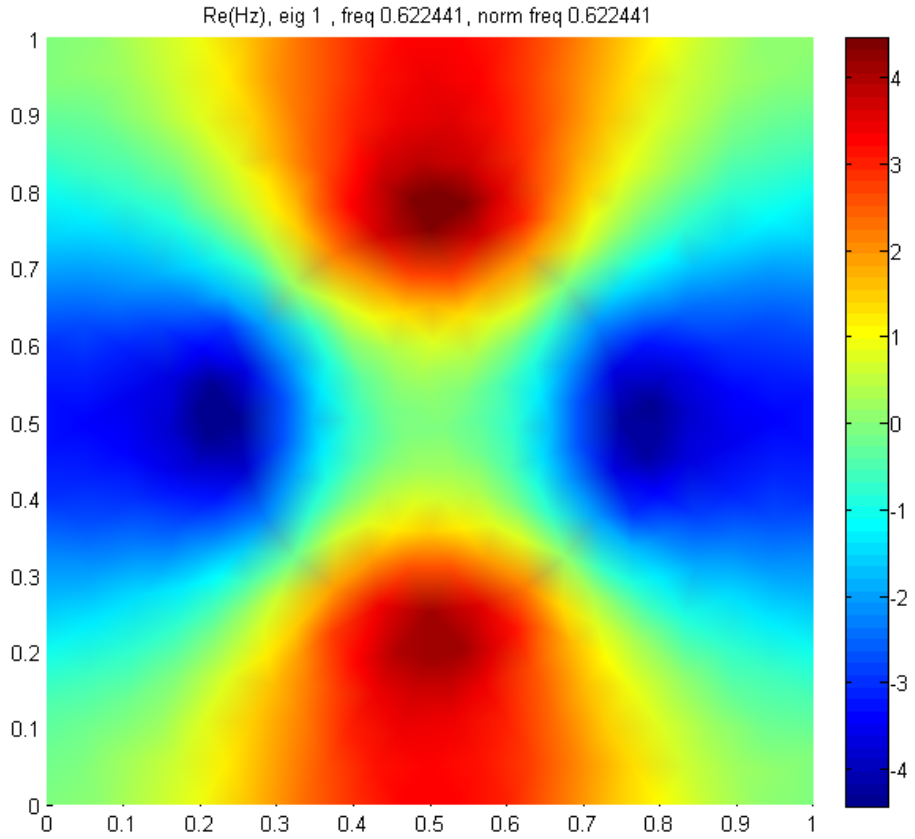

Figure 4: Magnetic field plot for the first eigenvalue computed for the first k-point.

```
[points edges triangles] = disphotn_import_comsol_mesh('square_lattice_example.mphtxt');
h = disphotn_extract_mode_h('example_hfield.txt', 16, 1, 1);
```

The two 1's in the second command select the first mode of the first k-point. The actual visualization of the mode is carried on using the `pdeplot` function:

```
pdeplot(points, edges, triangles, 'xydata', real(h));
```

The result is reported in Fig. **??**