Annexes

A. Summary of available software

MIT Photonic-Bands *for Linux/Unix 32/64Bit*

A free downloadable program package for computing the photonic band structures (dispersion relations) and electromagnetic modes of periodic dielectric structures. It is applicable both to photonic crystals (photonic band gap materials) and a wide range of other optical problems. It was developed by Steven G. Johnson at MIT in the Joannopoulos Ab Initio Physics group.

http://ab-initio.mit.edu/mpb/

Used for all calculations in this paper. Needs to be compiled.

MULTEM2

A new version of the program for transmission and band-structure calculations of photonic crystals – N. Stefanou, V. Yannopapas, and A. Modinos, Comput. Phys. Commun. 132, 189-196 (2000).

Not yet tested.

OPAL

Optical Program ALgorithm: A computational tool for photonics

OPAL calculates the transfer matrix for a given system once the permittivity and permeability have been specified as functions of position and frequency. OPAL can be used to calculate photonic band structure, transmission and reflection coefficients, and dispersion surfaces. The software is available at small fee just to cover postage – from Pendry Group, Imperial College of Science

http://www.sst.ph.ic.ac.uk/photonics/

Not yet tested.

Translight *for Windows 32Bit*

Photonic crystal modelling code

A transfer matrix method code developed by Andrew L. Reynolds at the University of Glasgow. This version of code has a powerful graphical user interface allowing the control of many aspects of the calculation, the crystal, incidence angles, materials, and so on. The crystal structure can be output in virtual reality macro language VRML to be viewed in a web-browser (with the appropriate plug-in). The program computes the transmission and reflection coefficients for the crystal. There are several common crystals predefined within the code that can be changed. A beta version can now be downloaded.

http://www.elec.gla.ac.uk/~areynolds/Software/SoftwareMain.htm

Installed for try out – seems to work but the interface is somewhat intransparent or tricky.

Diverse

Programs to calculate the photonic band structure of crystals with spherical "atoms"

Free downloadable programs that calculate the photonic band structure of dielectric crystals with a single spherical 'atom' per unit cell. The spheres must not overlap. Examples of such systems are photonic band gap materials based on colloidal crystals and air spheres in a dielectric matrix. Developed by the group of Ad Lagendijk and Rudolf Sprik, at the Van Der Waals-Zeeman Institute, Physics Department, University of Amsterdam.

http://www.wins.uva.nl/research/scm/sprik/pbs.htm

Not yet tested. Seems to use the spherical wave expansion.

B. MIT Photonic-Bands

B.1 Characteristics and capabilities

User interface

The program offers a command line and a batch file interface. With some simple text input, the user specifies the geometrical layout and the material characteristics of the unit cell of a photonic crystal. Next, he has to give some information about how to numerically quantify the problem and what to compute. Now, the iterative eigensolver is executed to compute the results that were asked for. Finally, the user can compute, manipulate and output additional results – either at each intermediate result or at the end. The band frequencies and parities, the group velocities and single values are written as human readable strings to the console, whereas the dielectric constant, the electric and magnetic field, the field power and other data is put into binary *HDF5* scientific data files [1].

As already mentioned, *MPB* can be used interactively or in batch mode. The input syntax is exactly the same for both modes. In batch mode, the console text output can be redirected into a file to make run *MPB* as a background process – think of some computations that may produce Mbytes of data and may take weeks to finish. The text interface offers programming like capabilities like assignment of values to variables, functions, loops and so on. The GNU Guile interpreter provides most of these capabilities. Unfortunately, its functions are very sparely or not at all documented.

Capabilities [2]

MPB computes a user-defined number of eigenmodes of photonic crystals by the planewave expansion method. Its computation algorithm is restricted to real eigenvalues of periodic structures. This means that the program cannot handle absorption nor structures of limited size [3]. Fortunately, it can deal with anisotrope dielectric materials.

Accuracy

The results are coherent in the sense, that an increasing number of volume elements used to represent the crystal's unit cell yield a converging series of eigenmodes. As a rule of thumb, every geometric element in the unit cell should be divided into at least 8 to 16 elements in each dimension. For simple crystals with one element per unit cell, the results differ by about 2% to 5% when 16 respectively 32 divisions per dimension are used. More than 32 divisions lead to expensive calculation but do no longer significantly modify the results.

B.2 Installation

MIT Photonic-Bands *MPB* comes as a software package build on several other freely downloadable packages. On a machine, the installation begins with building up the linear algebra subroutines in the *BLAS* and *LAPACK* packages if they are not already present. To speed up calculation, it is strongly advised to use a machine specific binary if available (for example from Sun's high performance workshop). Another way is to use the *ATLAS* package that tries to optimise performance by profiling and automatic code modification. Unfortunately, its installation takes some hours and the resulting libraries cannot be linked sometimes due to mismatching function naming conventions. The *FFTW* package provides the fast Fourier transform and is the last pure computation package.

Some optional packages can now be installed. *MPI* is an interface providing parallel computation on multiprocessor machines. *HDF5* is an interface to the commonly used scientific data file format and should be installed. At this moment, the very useful *h5togif* and *h5topng* data to graphic converter tools are not distributed or just buggy. A workaround will be presented below. *GNU Readline* provides a command history, autocompletion and command line edition to make command line input handier. Finally, *GNU Autoconf* helps software developers to provide an automatic installation routine with their package distribution. Could be handy to distribute some self-made *MPB* extensions.

Next, the command line interface to *MPB* has to be installed. *GNU Guile* provides a command line to C shell interpreter and the *libctl* library adds some particular functionality. Finally, the *MPB* package itself has

1 Consult "The MIT Photonic-Bands Manual" giving a detailed tutorial on use, data analysis and installation.
2 Look at the mathematics in "Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis".
3 Though they could be computed surrounding the sample with a large empty space – a heavily time consuming task!

to be installed.

Hints

Only the administrator can install the packages locally on a machine. This would give all users access to the *MPB* software. But there is a simple workaround:

Install the packages in the users home folder is possible. To do this, create a "bin" and a "lib" subfolder in the home directory to store the programs respectively their libraries. Then, unpack the binaries in the appropriate folders. Some packages need compilation, so when configuring them, pass the "--prefix *home*" option to change the output folder to the appropriate home directory. Make sure that the compiler and the linker finds the additionally source code headers and the previously installed libraries. The environment strings "LDFLAGS", "CCFLAGS" and "CPPFLAGS" generally pass options to the linker respectively compilers – just scan the "makefile" to find out how to specify additionally folders containing source code or libraries.

Finally, to run the software, the machines library loader needs to be informed about the new library location "*home*/lib". Just add this folder at the end of the library search path stored in the environment string "LD_LIBRARY_PATH" (Solaris OS 8).

Once installed in a user folder, other users can get the software simply by copying the "bin" and "lib" subfolders in their own home folder and by adapting the "LD_LIBRARY_PATH" environment string.

HDF5 graphics

Besides of the *Vis5D* software proposed in the *MPB* tutorial, the binary *HDF5* files can be read out and rewritten as text file by a Java program *NCSA H5View*. Hence, import into and data analysis with software like *MatLab* is much simplified.

B.3 Hints

MPB computes the reciprocal lattice by dropping the factor of 2π. Thus, to obtain the correct wave vectors, the *k*-values have to be multiplied by 2π.

Before launching a time consuming computation, just test your file by only initialising the computation with the command "init-params ...". Then, test if the specified *k*-points really match the right values. Note that *MPB* notes the wave vectors in units of the reciprocal lattice vectors.

B.4 Sample control files

"linear.ctl"

```
; x linear lattice of layers
;
; Layer thickness is R*Lattice size
; Layer permitivity is E*Eo
;
; +------------------+
; | Input parameters |
; +------------------+
;
(define-param E 8)
(define-param R 0.5)
;
; +----------------+
; | Setup geometry |
; +----------------+
;
(set! geometry (list
  (make block
          (center 0 0 0)
          (size (* 2 R) 1 1)
          (material (make dielectric (epsilon E)))
  )
))
;
; +--------------+
; | Quantization |
; +--------------+
;
(set! grid-size (vector3 32 1 1))
```

```
(set! k-points (list
  (vector3 0 0 0)         ; G
  (vector3 0.5 0 0)       ; X
))
(set! k-points (interpolate 32 k-points))  ; total of 34 points
(set! num-bands 20)
(run-te display-group-velocities
; (output-at-kpoint (vector3 0.5 0 0) fix-efield-phase output-efield-y)
)
```

"square.ctl"

```
; xy square lattice of cylindrical rods
;
; Rod radius is R*lattice size = R
; Rod permitivity is E*Eo
; Rod height is infinite
;
; +------------------+
; | Input parameters |
; +------------------+
;
(define-param E 11)
(define-param R 0.2)
;
; +----------------+
; | Setup geometry |
; +----------------+
;
(set! geometry (list (make cylinder (center 0 0 0)
                                    (radius R)
                                    (height infinity)
                                    (material (make dielectric (epsilon E)))
          )))
;
; +--------------+
; | Quantization |
; +--------------+
;
(set! grid-size (vector3 32 32 1))
(set! k-points (list (vector3 0.5 0.5 0)   ; M
                     (vector3 0 0 0)       ; G
                     (vector3 0.5 0 0)     ; X
                     (vector3 0.5 0.5 0) ; M
          ))
(set! k-points (interpolate 8 k-points))
(set! num-bands 20)
(run-te display-group-velocities
; (output-at-kpoint (vector3 0.5 0 0) fix-efield-phase output-efield)
)
(run-tm display-group-velocities
; (output-at-kpoint (vector3 0.5 0 0) fix-efield-phase output-efield)
)
```

"triangle.ctl"

```
; xy triangular lattice of cylindrical rods
;
; Rod radius is R*lattice size = R
; Rod permitivity is E*Eo
; Rod height is infinite
; Rod axis // z
; turned around x towards y by A
; turned around z towards y by B
;
; Plane wave basis NxNx1
;
; +------------------+
; | Input parameters |
; +------------------+
;
(define-param A 0)
(define-param B 0)
(define-param E 11)
(define-param N 32)
(define-param R 0.30)
```

```
;
; +------------------+
; | Lattice geometry |
; +------------------+
;
(set! geometry-lattice
  (make lattice
          (basis1 (sqrt 0.75) -0.5 0)
          (basis2 (sqrt 0.75) 0.5 0)
          (basis3 0 0 1)
  )
)
;
; +-----------------+
; | Rod axis vector |
; +-----------------+
;
(define axe (rotate-vector3 (vector3 0 0 1) (deg->rad B) (rotate-vector3 (vector3 (sqrt 0.75) -0.5
0) (deg->rad (- A)) (vector3 0 0 1))))
;
; +----------------+
; | Setup geometry |
; +----------------+
;
(set! geometry (list
  (make cylinder
          (axis axe)
          (center 0 0 0)
          (height infinity)
          (radius R)
          (material (make dielectric (epsilon E)))
  )
))
;
; +--------------+
; | Quantization |
; +--------------+
;
(set! grid-size (vector3 N N 1))
(set! k-points (list (vector3 (/ -3) (/ 3) 0)      ; K
                (vector3 0 0 0)             ; G
                (vector3 0 0.5 0)           ; M
                (vector3 (/ -3) (/ 3) 0)    ; K
          ))
(set! k-points (interpolate 8 k-points))
(set! num-bands 20)
(run-te display-group-velocities
; (output-at-kpoint (vector3 0 0.5 0) fix-efield-phase output-efield)
)
(run-tm display-group-velocities
; (output-at-kpoint (vector3 0 0.5 0) fix-efield-phase output-efield)
)
```

B.5 Unix shell script

```
#!/usr/bin/sh
#
mkdir ${1}
umask 027
for epsilon in ε-list;
do        for radius in r-list;
  do      echo "Processing: ${1} epsilon=${epsilon} radius=${radius}"
          mpb E=${epsilon} R=${radius} "${1}.ctl" > "${1}/E=${epsilon} R=${radius}"
#         mv epsilon.h5 "${1}/E=${epsilon} R=${radius}.h5"
  done
done
rm -f epsilon.h5
```

C. Data analysis

This section incorporates some *MatLab* functions used for data mining of the *MPB* text output files. These functions proofed useful for the presentations in this paper.

C.1 Data acquisition: "readBands.m"

```
%[data,bloc]=readBands(file)
%--------------------------
%
%Scans a "MIT Photonic bands" simulation output file for data
%sections and reads them in.
%
% data{}          cell vector
%     .file          full file name
%     .grid          grid size in [x;y;z] column
%     .size          lattice size as [x;y;z] column
%     .lattice       lattice vectors in [x y z] rows
%     .reciprocal    reciprocal lattice vectors in [x y z] rows
%     .geometry      unit cell geometry as cell strings
%     .polarity      band polarisation 'none | yeven | yodd | zeven(te) | zodd(tm)'
%     .vectors       wave vectors in [x y z k] rows
%     .bands         wave frequencies in [1 2 3 ...] rows
%     .ranges        band ranges in [from;to] columns
%     .xvelocity
%     .yvelocity
%     .zvelocity     x/y/z group velocity in [1 2 3 ...] rows
%     .yparity
%     .zparity       y/z-parity [-1...+1] in [1 2 3 ...] rows
%     .time          total elapsed time for computation in seconds
%
% bloc:  last cell of data in case of error
%
% file:  simulation file name
%
function [data,bloc]=readBands(file)
if ~nargin | isempty(file) | ~ischar(file)
   error('Unknown or undefined argument.');
end;
if ~nargout
   error('Unknown return value.');
end;
[fid,msg]=fopen(file,'rt');
if fid < 0
   error(msg);
end;
data={};
line=fgetl(fid);
while ischar(line)
   bloc.file=fopen(fid);         % file name
   bloc.grid=zeros(3,1);
   bloc.size=zeros(3,1);
   bloc.lattice=zeros(3);
   bloc.reciprocal=zeros(3);
   bloc.geometry={};
   bloc.polarity='';
   bloc.vectors=[];
   bloc.bands=[];
   bloc.ranges=[];
   bloc.xvelocity=[];
   bloc.yvelocity=[];
   bloc.zvelocity=[];
   bloc.yparity=[];
   bloc.zparity=[];
   bloc.time=0;
   while ~length(strmatch('Grid size',line))
      line=fgetl(fid);
      if ~ischar(line)        % grid size
         return;
      end;
   end;
   bloc.grid=sscanf(line,'Grid size is %d x %d x %d');
   line=fgetl(fid);           % lattice
   while ischar(line) & ~length(strmatch('Lattice vectors',line))
```

```
   line=fgetl(fid);
end;
for i=1:3
   line=fgetl(fid);
   if ~ischar(line)
      'Corrupt lattice definition.'
      return;
   end;
   bloc.lattice(i,:)=sscanf(line(findstr(line,'(':length(line)),'(%f, %f, %f)')';
   bloc.size(i)=norm(bloc.lattice(i,:));
end;
line=fgetl(fid);           % reciprocal lattice
while ischar(line) & ~length(strmatch('Reciprocal lattice vectors',line))
   line=fgetl(fid);
end;
for i=1:3
   line=fgetl(fid);
   if ~ischar(line)
      'Corrupt reciprocal lattice definition.'
      return;
   end;
   bloc.reciprocal(i,:)=sscanf(line(findstr(line,'(':length(line)),'(%f, %f, %f)')';
end;
line=fgetl(fid);           % unit cell geometry
while ischar(line) & ~length(strmatch('Geometric objects',line))
   line=fgetl(fid);
end;
line=fgetl(fid);
while ischar(line) & ~length(strmatch('Geometric object t',line))
   bloc.geometry{length(bloc.geometry)+1}=line;
   line=fgetl(fid);
end;
i=0;                       % number of wave vectors
n=[];
line=fgetl(fid);
while ischar(line) & isempty(n)
   n=sscanf(line,'%d k-points:');
   line=fgetl(fid);
end;
while ischar(line) & ~length(strmatch('Solving for band p',line))
   line=fgetl(fid);
end;
if ~ischar(line)           % band polarity
   'Corrupt polarity definition.'
   return;
end;
bloc.polarity=line(findstr(line,': ')+2:length(line)-1);
line=fgetl(fid);
while ischar(line) & ~length(strmatch('Band ',line))
   s=findstr(line,':, ');
   if length(s) == 1 & s > 5
      switch line(s-5:s-1)
      case 'freqs'         % band frequency
         if i
            v=sscanf(line(s+1:length(line)),', %f',inf)';
            if v(1) ~= i
               'Synchronization lost.'
            end;
            bloc.bands(i,:)=v(6:length(v));
            bloc.vectors(i,:)=v(2:5);
         else
            bloc.bands=zeros(n,length(findstr(line,','))-5);
         end;
         i=i+1;
      case 'ocity'         % x/y/z group velocity
         l=[' ' line];
         l=l(s-8);
         s=findstr(line,',');
         if sscanf(line(s(1):s(2)),', %d') ~= i-1
            'Synchronization lost.'
         end;
         switch l
         case {'x','y','z'}
            v=sscanf(line(s(2):length(line)),', %f',inf)';
         end;
```

```
               switch l
               case 'x'
                  if i == 2
                     bloc.xvelocity=zeros(n,length(v));
                  end;
                  bloc.xvelocity(i-1,:)=v;
               case 'y'
                  if i == 2
                     bloc.yvelocity=zeros(n,length(v));
                  end;
                  bloc.yvelocity(i-1,:)=v;
               case 'z'
                  if i == 2
                     bloc.zvelocity=zeros(n,length(v));
                  end;
                  bloc.zvelocity(i-1,:)=v;
               otherwise
                  v=sscanf(line(s(2):length(line)),', #(%f %f %f)',[3 inf]);
                  if i == 2
                     bloc.xvelocity=zeros(n,size(v,2));
                     bloc.yvelocity=bloc.xvelocity;
                     bloc.zvelocity=bloc.xvelocity;
                  end;
                  bloc.xvelocity(i-1,:)=v(1,:);
                  bloc.yvelocity(i-1,:)=v(2,:);
                  bloc.zvelocity(i-1,:)=v(3,:);
               end;
            case 'arity'         % y/z parity
               s=findstr(line,',');
               if sscanf(line(s(1):s(2)),', %d') ~= i-1
                  'Synchronization lost.'
               end;
               v=sscanf(line(s(2):length(line)),', %f',inf)';
               switch line(s-7)
               case 'y'
                  if i == 2
                     bloc.yparity=zeros(n,length(v));
                  end;
                  bloc.yparity(i-1,:)=v;
               case 'z'
                  if i == 2
                     bloc.zparity=zeros(n,length(v));
                  end;
                  bloc.zparity(i-1,:)=v;
               otherwise
                  'Unknown parity tag.'
               end;
            otherwise
               'Unknown data tag.'
            end;
         end;
         line=fgetl(fid);
      end;
      n=size(bloc.bands,2);      % band limits
      bloc.ranges=zeros(2,n);
      for i=1:n
         if ~ischar(line)
            return;
         end;
         if sscanf(line(6:10),'%d') ~= i
            'Synchronization lost.'
         end;
         bloc.ranges(:,i)=sscanf(line(findstr(line,': ')+1:length(line)),'%f at #(%*f %*f %*f) to
%f')';
         line=fgetl(fid);
      end;
      while ischar(line)            % total elapsed time
         if  strcmp(line(1:5),'total')
            line=line(findstr(line,':')+2:length(line));
            n=findstr(line,',');
            while length(n)
               bloc.time=bloc.time*60+sscanf(line,'%d');
               line=line(n(1)+2:length(line));
               n=findstr(line,',');
            end;
```

```
            bloc.time=bloc.time*60+sscanf(line,'%d');
            break;
         end;
         line=fgetl(fid);
      end;
      data{length(data)+1}=bloc;
      line=fgetl(fid);
   end;
   fclose(fid);
   bloc={};
```

C.2 Plotting the band structure: "plotBands.m"

```
%hnd=plotBands(data, name, head, hnd)
%-----------------------------------
%
%Plot "MIT Photonic-Bands" band structure data.
%
% hnd:   figure handle
% data:  data structure (see readBands)
% name:  window title
% head:  axis title
%
function hnd=plotBands(data, name, head, hnd)
if ~nargin
   error('Unknown or undefined argument.');
end;
if nargin < 2 | ~ischar(name)
   name='MPB band structure';
end;
if ~isstruct(data)
   data=data{1};
end;
flag=invalid(data);
if flag > 7
   error('Data set corrupt.');
end;
if flag > 3
   'Data set damaged.'
end;
if nargin < 3 | ~ischar(head)
   head=[data.file ' polarity: ' data.polarity];
end;
if nargin < 4 | isempty(hnd) | ~ishandle(hnd)
   hnd=window(name,head);
else
   if ~strcmp(get(hnd,'Tag'),'Frequency')
      close(hnd);
      hnd=window(name,head);
   else
      figure(hnd);
      set(hnd,'Name',name);
      title(head);
   end;
end;
axe=get(hnd,'CurrentAxes');
delete(findobj(get(axe,'Children')));
data=rescale(data);        % scale and label the k axis
set(axe,'XLim',[data.scale(data.edges(1))
data.scale(data.edges(length(data.edges)))],'XTick',data.scale(data.edges),'XTickLabel',data.ticks
,'YLimMode','auto');
t=get(axe,'ColorOrder');
h=line(data.scale,data.bands,'Parent',axe,'Color',t(1,:));
t=get(axe,'YTick');
t=line(repmat(data.scale(data.edges(2:length(data.edges)-1)),1,2)',repmat([t(1)
t(length(t))],length(data.edges)-2,1)','Parent',axe,'Color',[0.75 0.75 0.75]);
set(axe,'Children',[h;t],'UserData',[1;t]);
%
% 'UserData' holds #plots and handles of the gray vertical lines
```

"private\rescale.m"

```
%data=rescale(data)
%------------------
%
```

```
%Rescales the k axis of a "MIT Photonic bands" data structure.
%Sorts band frequencies in ascending order.
%
% data          data structure completed with
%     .edges          corner indexes
%     .scale          k-axis scaling centered on first Gamma point
%     .ticks          corner labels
%
function data=rescale(data)
data.bands=sort(data.bands,2);
t=sum(data.lattice(1,:).*data.lattice(2,:))/prod(data.size(1:2));
if abs(t) < 0.000001        % rectangular lattice
   t={'G','X','X','Y','Y','M','M','M','M'};
   v=[ 0.0  0.0  0; ...     % Gamma point
       1/2  0.0  0; ...     % X point
      -1/2  0.0  0; ...
       0.0  1/2  0; ...     % Y point
       0.0 -1/2  0; ...
       1/2  1/2  0; ...     % M point
      -1/2  1/2  0; ...
       1/2 -1/2  0; ...
      -1/2 -1/2  0];
else
   if abs(t-0.5) < 0.000001 % triangular lattice
   t={'G','M','M','M','M','M','M','K','K','K','K','K','K'};
   v=[ 0.0  0.0  0; ...     % Gamma point
       1/2  0.0  0; ...     % M point
      -1/2  0.0  0; ...
       0.0  1/2  0; ...
       0.0 -1/2  0; ...
       1/2  1/2  0; ...
      -1/2 -1/2  0; ...
       1/3 -1/3  0; ...     % K point
      -1/3  1/3  0; ...
       2/3  1/3  0; ...
      -2/3 -1/3  0; ...
       1/3  2/3  0; ...
      -1/3 -2/3  0];
   else
      t={'G'};
      v=zeros(1,3);
   end;
end;
n=size(data.vectors,1);
e=data.vectors(2:n,1:3)-data.vectors(1:n-1,1:3);   % delta-k-vectors
d=sqrt(sum(e.*e,2));                               % delta-k-lengths
e=sum(e(2:n-1,:).*e(1:n-2,:),2)./(d(2:n-1).*d(1:n-2));
data.edges=[1;find(abs(e-1) > 0.000001)+1;n];
data.scale=zeros(n,1);
for i=1:n-1
   data.scale(i+1)=data.scale(i)+d(i);
end;
e=data.vectors(data.edges,1:3);                    % k-points
l={};
for i=1:length(data.edges)
   n=find(sum(abs(v-repmat(e(i,:),size(v,1),1)),2) < 0.000001);
   if isempty(n)
      l{i}='?';
   else
      l{i}=t{n};
      if t{n} == 'G' & ~data.scale(1)
         data.scale=data.scale-data.scale(data.edges(i));
      end;
   end;
end;
data.ticks=l;
```

C.3 Plotting the density of states: "plotDensity.m"

```
%hnd=plotDensity(data, name, head, hnd, n)
%----------------------------------------
%
%Plot "MIT Photonic-Bands" density of states.
%
% hnd:   figure handle
```

```
% data:  data structure (see readBands)
% name:  window title
% head:  axis title
% n      number of bins or zero for extrapolation
%           if not specified, extrapolation is choosen
%           whenever the velocity data is present, the
%           number of bins is set automatically else.
%
function hnd=plotDensity(data, name, head, hnd, n)
if ~nargin
   error('Unknown or undefined argument.');
end;
if nargin < 2 | ~ischar(name)
   name='MPB density of states';
end;
if ~isstruct(data)
   data=data{1};
end;
flag=invalid(data);
if flag > 7
   error('Data set corrupt.');
end;
if flag > 3
   'Data set damaged.'
end;
if nargin < 3 | ~ischar(head)
   head=[data.file ' polarity: ' data.polarity];
end;
if nargin < 4 | isempty(hnd) | ~ishandle(hnd)
   hnd=window(name,head);
   set(hnd,'Tag','Density');
else
   if ~strcmp(get(hnd,'Tag'),'Density')
      close(hnd);
      hnd=window(name,head);
      set(hnd,'Tag','Density');
   else
      figure(hnd);
      set(hnd,'Name',name);
   end;
end;
axe=get(hnd,'CurrentAxes');
delete(findobj(get(axe,'Children')));
if nargin < 5
   n=[];
end;
[data,n]=density(data,n);
if n
   h=barh(data.density(:,1),data.density(:,2),1);
   t=get(gca,'ColorOrder');
   set(h,'EdgeColor',t(1,:),'FaceColor',t(1,:));
   set(gca,'FontSize',15,'Position',[0.13 0.08 0.82 0.82]);
   ylabel('f [c/a]');
   title(head);
else
   j=10^ceil(log10(max(data.density(:,2))));
   i=10^floor(log10(max(0.001,min(data.density(:,2)))));
   set(axe,'XScale','log','XLim',[i j],'XTickMode','auto','XTickLabelMode','auto','Position',[0.13
0.1 0.82 0.8]);
   t=get(axe,'ColorOrder');
   line([repmat(i,size(data.density,1),1)
data.density(:,2)]',repmat(data.density(:,1),1,2)','Parent',axe,'Color',t(3,:));
end;
```

"private\density.m"

```
%[data,n]=density(data, n)
%-------------------------
%
%Computes the empirical density of states either by counting the
%number of modes per bin or by the free space formula (assuming
%spherical/circular k-vector surfaces of constant energy).
%
% data   data structure completed with
%     .density    density of states in [f d] rows
%
```

```
% n      number of bins or zero for extrapolation
%           if not specified, extrapolation is choosen
%           whenever the velocity data is present, the
%           number of bins is set automatically else.
%
function [data,n]=density(data, n)
if nargin < 2 | isempty(n) | ~isnumeric(n)
   if velocity(data)
      n=0;
   else
      n=ceil(sqrt(prod(size(data.bands))));
   end;
else
   if n <= 0
      if ~velocity(data)
         n=ceil(sqrt(prod(size(data.bands))));
      else
         n=0;
      end;
   else
      n=ceil(n);
   end;
end;
if n
   m=max(max(data.bands));    % frequency bins
   i=m/(2*n);
   data.density=[(i:2*i:m)' zeros(n,1)];
   for m=1:n
      data.density(m,2)=sum(sum(abs(data.bands-data.density(m,1)) < i));
   end;
else
   v=[];                      % group velocity
   if data.grid(1) > 1
      v=data.xvelocity.^2;
   end;
   if data.grid(2) > 1
      if isempty(v)
         v=data.yvelocity.^2;
      else
         v=v+data.yvelocity.^2;
      end;
   end;
   if data.grid(3) > 1
      if isempty(v)
         v=data.zvelocity.^2;
      else
         v=v+data.zvelocity.^2;
      end;
   end;
   v=sqrt(v);
   s=data.lattice(find(data.grid > 1),:);
   d=size(s,1);
   switch d                   % dimension
   case 1
      s=2*sqrt(norm(s))/pi;
   case 2
      s=sqrt(norm(cross(s(1,:),s(2,:))))/pi;
   case 3
      s=sqrt(norm(sum(s(3,:).*cross(s(1,:),s(2,:)))))/pi^2;
   otherwise
      warning('Point density skipped.');
      data.density=zeros(0,2);
      return;
   end;
   data.density=zeros(prod(size(v)),2);
   j=size(v,1);
   for i=1:size(v,2)          % vectorize & sort
      data.density(1+(i-1)*j:i*j,:)=[data.bands(:,i) v(:,i)];
   end;
   [data.density(:,1) i]=sort(data.density(:,1));
   data.density(:,2)=s*data.density(:,1).^(d-1)./max(0.000001,data.density(i,2).^d);
end;


%Checks if the extrapolation can be computed
```

```
%
function flag=velocity(data)
flag=0;
if data.grid(1) > 1 & (~isfield(data,'xvelocity') |
~isequal(size(data.bands),size(data.xvelocity)))
   return;
end;
if data.grid(2) > 1 & (~isfield(data,'yvelocity') |
~isequal(size(data.bands),size(data.yvelocity)))
   return;
end;
if data.grid(3) > 1 & (~isfield(data,'zvelocity') |
~isequal(size(data.bands),size(data.zvelocity)))
   return;
end;
flag=1;
```

C.4 Plotting gap maps: "plotGaps.m"

```
%hnd=plotGaps(e, name, head, hnd)
%--------------------------------
%
%Plots the te and tm band gaps for a set of files
%computed at epsilon e and r/a=0.02:0.02:0.48.
%
%hnd=plotBands(data, name, head, hnd)
%-----------------------------------
%
%Plot "MIT Photonic-Bands" band gaps for a set of
%files computed at epsilon e and r/a=0.02:0.02:0.48.
%
% hnd:   figure handle
% name:  window title
% head:  axis title
%
function hnd=plotGaps(e, name, head, hnd)
if ~nargin
   error('Unknown or undefined argument.');
end;
if nargin < 2 | ~ischar(name)
   name='MPB gap map';
end;
if nargin < 3 | ~ischar(head)
   head='';
end;
if nargin < 4 | isempty(hnd) | ~ishandle(hnd)
   hnd=reopen(name,head);
else
   if ~strcmp(get(hnd,'Tag'),'Gapmap')
      close(hnd);
      hnd=reopen(name,head);
   else
      figure(hnd);
      set(hnd,'Name',name);
      title(head);
   end;
end;
axe=get(hnd,'CurrentAxes');
delete(findobj(get(axe,'Children')));
if length(head)
   set(axe,'Position',[0.13 0.14 0.83 0.76]);
else
   set(axe,'Position',[0.13 0.14 0.83 0.82]);
end;
set(axe,'XLim',[0 0.5],'XTickMode','auto','XTickLabelMode','auto');
for r=1:24
   if r < 5
      data=sprintf('E=%d R=0.0%d',[e 2*r]);
   else
      data=sprintf('E=%d R=0.%d',[e 2*r]);
   end;
   data=readBands(data);
   for s=1:length(data)
      t=bandgaps(data{s});
      switch t.polarity
```

```
      case 'te'
         c=[1 0 0];        % red
         w=0.01;           % width
         o=0.01;           % x offset
      case 'tm'
         c=[0 0 1];        % blue
         w=0.01;
         o=0;
      otherwise
         c=[0 1 0];        % green
         w=0.02;
         o=0.01;
      end;
      t=t.gaps;
      for i=1:size(t,2)
         rectangle('EdgeColor','none','FaceColor',c,'Parent',axe,'Position',[0.02*r-o t(1,i)*(1-
t(2,i)/2) w prod(t(:,i))]);
      end;
   end;
end;

function hnd=reopen(name, head)
hnd=window(name,head);
set(hnd,'Tag','Gapmap');
xlabel('r [a]');
```

"private\bandgaps.m"

```
%data=bandgaps(data)
%-------------------
%
%Computes the band gaps and their relatif
%extent to the mean frequency.
%
% data   data structure completed with
%     .gaps    band gaps in [f e] rows
%
function data=bandgaps(data)
i=size(data.ranges,2);
i=find(data.ranges(1,2:i) > data.ranges(2,1:i-1));
f=(data.ranges(1,i+1)+data.ranges(2,i))/2;
e=(data.ranges(1,i+1)-data.ranges(2,i))./f;
i=find(e > 0.01);
data.gaps=[f(i);e(i)];        % store only gaps > 1%
```

C.5 General private functions

"private\invalid.m"

```
%flag=invalid(data)
%------------------
%
%Validates a photonic bands data structure.
%
%data    data structure
%flag    sum of all diagnostic flags
%     0  valid structure
%     1  file/geometry/polarity/time member invalid
%     2  velocity/parity member invalid
%     4  grid/size/lattice/reciprocal member invalid
%     8  vectors/bands/ranges member invalid
%
function flag=invalid(data)
flag=15;
if ~nargin | isempty(data) | ~isstruct(data)
   return;
end;
if isfield(data,'file') & isstr(data.file) ...
      & isfield(data,'geometry') & iscellstr(data.geometry) ...
      & isfield(data,'polarity') & isstr(data.polarity) ...
      & isfield(data,'time') & isnumeric(data.time)
   flag=flag-1;
end;
if (~isfield(data,'xvelocity') | isnumeric(data.xvelocity)) ...
      & (~isfield(data,'yvelocity') | isnumeric(data.yvelocity)) ...
```

```
      & (~isfield(data,'zvelocity') | isnumeric(data.zvelocity)) ...
      & (~isfield(data,'yparity') | isnumeric(data.yparity)) ...
      & (~isfield(data,'zparity') | isnumeric(data.zparity))
   flag=flag-2;
end;
if isfield(data,'grid') & isnumeric(data.grid) ...
      & isfield(data,'size') & isnumeric(data.size) ...
      & isequal(size(data.size),size(data.grid),[3 1]) ...
      & isfield(data,'lattice') & isnumeric(data.lattice) ...
      & isfield(data,'reciprocal') & isnumeric(data.reciprocal) ...
      & isequal(size(data.lattice),size(data.reciprocal),[3 3])
   flag=flag-4;
end;
if isfield(data,'vectors') & isnumeric(data.vectors) & size(data.vectors,2) == 4 ...
      & isfield(data,'bands') & isnumeric(data.bands) & size(data.vectors,1) == size(data.bands,1)
...
      & isfield(data,'ranges') & isnumeric(data.ranges) & isequal(size(data.ranges),[2
size(data.bands,2)])
   flag=flag-8;
end;
```

"private\window.m"

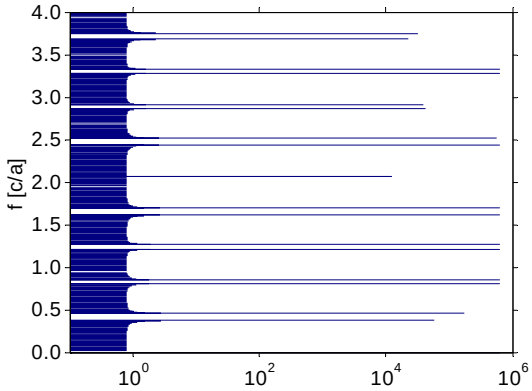
```
%hnd=window(name, head)
%----------------------
%
%Create a new "MIT Photonic-Bands" frequency window.
%
% hnd:   figure handle
% name:  window title
% head:  axis title
%
function hnd=window(name, head)
axe.Box='on';
axe.Color=[1 1 1];
axe.ColorOrder=1/255* ...
   [128   0   0; ...
      0 128   0; ...
      0   0 128; ...
      0   0   0; ...
    255   0   0; ...
      0 255   0; ...
      0   0 255; ...
      128 128 128];
axe.FontSize=15;
axe.Layer='top';
axe.LineStyleOrder='-';
axe.NextPlot='replacechildren';
axe.Position=[0.13 0.08 0.82 0.82];
axe.TickDir='out';
axe.TickDirMode='manual';
axe.XLimMode='manual';
axe.XTickLabelMode='manual';
axe.XTickMode='manual';
wnd.Colormap=1/255* ...
  [  0   0   0; ... %  0 black
   128 128 128; ... %  1 dark grey
   192 192 192; ... %  2 light grey
   255 255 255; ... %  3 white
   128   0   0; ... %  4 dark red
   192   0   0; ... %  5 red
   255   0   0; ... %  6 light red
     0 128   0; ... %  7 dark green
     0 192   0; ... %  8 green
     0 255   0; ... %  9 light green
     0   0 128; ... % 10 dark blue
     0   0 192; ... % 11 blue
     0   0 255; ... % 12 light blue
   255 255   0; ... % 13 yellow
   255   0 255; ... % 14 magenta
     0 255 255];    % 15 cyan
wnd.Dithermap=[];
wnd.DoubleBuffer='on';
wnd.IntegerHandle='off';
%wnd.MenuBar='none';
wnd.MinColormap=16;
```

```
if nargin & ischar(name) & length(name)
   wnd.Name=name;
else
   wnd.Name='MIT Photonic bands';
end;
wnd.NextPlot='replacechildren';
wnd.NumberTitle='off';
wnd.Position=[100 100 500 400];
wnd.Tag='Frequency';
hnd=figure(wnd);
axes(axe,'Parent',hnd);
ylabel('f [c/a]');
if nargin > 1 & ischar(head)
   title(head);
end;
```

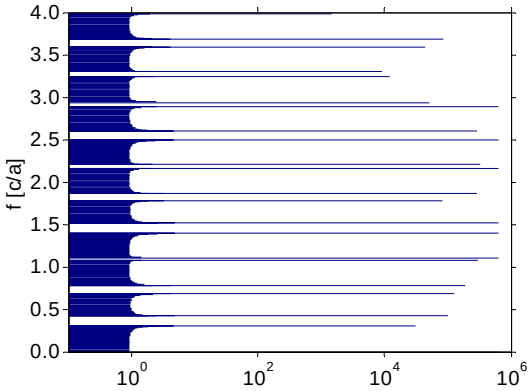
D. Results

D.1 Linear 1D lattice

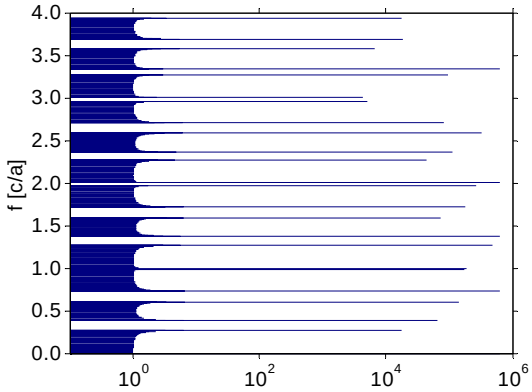
Linear 1D lattices with size *a*. The layers have equal thickness (50% air and 50% dielectric material).



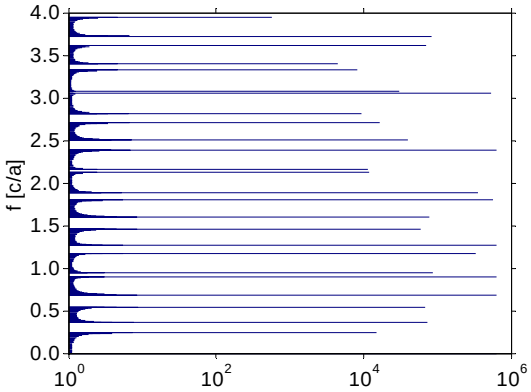
Graph 1: Density of states for $\varepsilon_r = 2$



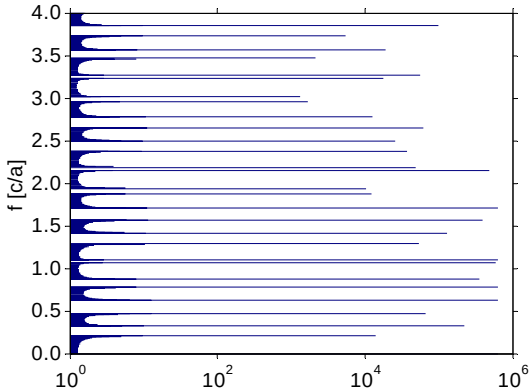
Graph 2: Density of states for $\varepsilon_r = 3$



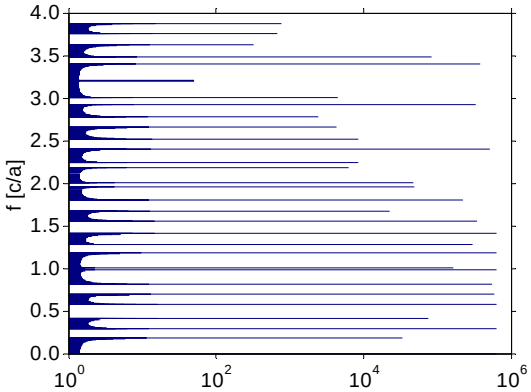
Graph 3: Density of states for $\varepsilon_r = 4$



Graph 4: Density of states for $\varepsilon_r = 5$



Graph 5: Density of states for $\varepsilon_r = 7$



Graph 6: Density of states for $\varepsilon_r = 9$

2002

D.3 *MPB* computation error

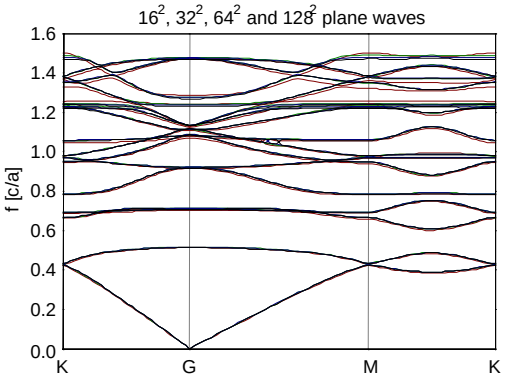
This analysis of the calculation error is based on the comparison of the result calculated with different basis sets of plane waves. It is expected that for an increasing number of plane waves, the results converge.

The analysis has been done for a triangular 2D lattice with a single cylindrical rod in each unit cell. The lattice size is a, the rod radius is $r = 0.25a$ and its dielectric constant is $\varepsilon_r = 10$. The calculation was done for $N = \{8,16,32,64,128\}$ yielding a basis set of N^2 plane waves. The mean relative differences for increasing N were computed to:

8.0%	(8→16)
0.85%	(16→32)
0.22%	(32→64)
0.25%	(64→128)

The pure computation error for N above 32 can be expected to be less than 1%. Note the computation times:

15s	(8)
76s	(16)
8´	(32)
51´	(64)
4h	(128) !



Graph 13: TE band structure for different N