MIT Photonic Bands 开发指南非官方版

文档说明：这是对 MPB 官方说明文档开发信息部分的简体中文的简单意译，在时间仓促和个人理解受限下，不担官方文档对应中文部分的保准确性和科学性。但求对任何对光子晶体计算和 MPB 程序感兴趣的个人和单位一起研究和维护，欢迎提供建议和指导！

Developer Information 开发信息

Here, we begin with a brief overview of what the program is computing, and then describe how the program and computation are broken up into different portions of the code.

在这里，我们开始对这个程序的计算原理进行简要的概述，之后对程序包不同文件(夹)包含代码的进行简单的介绍并说明其计算原理。

Forgive the primitive math typography below; this will be rectified when MathML is supported in a decent browser.

原谅下面的简陋的数学排版（译者：这里指代英文原版的数学公式排版）。MathML 是一个让浏览器支持数学公式排版的语言，在新文档发布中将使用到这里（翻译中对里面的公式进行 LaTeX 排版）。

The Mathematics of MPB MPB 的数学原理

This section provides a whirlwind tour of the mathematics of photonic band structure calculations and the algorithms that we employ. For more detailed information, see:

本节提供一种光子能带结构计算的原理和算法，我们运用数学公式进行快速而简单的描述。如需详细理解，请参阅一下文章：

Photonic Crystals: Molding the Flow of Light, by J. D. Joannopoulos, R. D. Meade, and J. N. Winn (Princeton, 1995).

Steven G. Johnson and J. D. Joannopoulos, "Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis," Optics Express 8, no. 3, 173-190 (2001).

The MIT Photonic-Bands Package takes a periodic dielectric structure and computes the eigenmodes of that structure, which are the electromagnetic waves that can propagate through the structure with a definite frequency. This corresponds to solving an eigenvalue problem M h = (w/c)^2 h, where h is the magnetic field, w is the frequency, and M is the Maxwell operator curl 1/epsilon curl. We also have an additional constraint, that div h be zero (the magnetic field must be "transverse").

运行MPB计算程序，需要输入一个周期性的介电（非金属）结构描述，并计算该结构的本征模式。这就是能够通过这一结构而传播且确定频率的电磁波。这相当于解决方程 M h =(w/c)^2 * H 的特征值问题，其中 H 是磁场，w 是频率，M 是麦克斯韦算符。我们也有一个外加条件的约束，即 H 的散度为零（磁场必须是横磁场）。

$$\begin{cases} \nabla \times \left[\frac{1}{\varepsilon_{(\mathbf{r})}} \nabla \times \mathbf{H}_{(\mathbf{r})} \right] = (\frac{\omega}{c})^2 \mathbf{H}_{(\mathbf{r})} \\ \nabla \cdot \mathbf{H}_{(\mathbf{r})} = 0 \end{cases}$$

Since the structure is periodic, we can also invoke Bloch's theorem to write the states in the form exp(i k*x) times a periodic function, where k is the Bloch wavevector. So, at each k-point (Bloch wavevector), we need to solve for a discrete set of eigenstates, the photonic bands of the structure.

由于晶体结构是周期的，我们也可以调用布洛赫定理的写成exp(i*k*x)的形式的时间周期函数，其中k是布洛赫波失。因此，在每个k点（布洛赫波失），我们需要解决的一组离散的本征态和目标晶体结构的光子能带。

To solve for the eigenstates on a computer, we must expand the magnetic field in some basis, where we truncate the basis to some finite number of points to discretize the problem. For example, we could use a traditional finite-element basis in which the field is taken on a finite number of mesh points and linearly interpolated in between. However, it is expensive to enforce the transversality constraint in this basis. Instead, we use a Fourier (spectral) basis, expanding the periodic part of the field in terms of exp(i G*x) planewaves. In this basis, the transversality constraint is easy to maintain, as it merely implies that the planewave amplitudes must be orthogonal to k + G.

为了在计算机上的解决本征态计算，我们必须在基矢方向上展开磁场函数，我们在截断点的基础上对一些数量有限离散问题的磁场进行计算。例如，我们可以使用传统的有限元计算，其中一个领域是采取有限的网点数量和线性插值的方法。然而，这样的方法是耗费大量的计算资源，来执行在此基础上横截约束。相反，我们可以使用傅立叶（频谱）的基础上，对具有exp(i*G*x)周期性场的进行平面波展开。在此基础上，横截约束很容易处理的，因为它仅仅意味着平面波正交振幅一定对(k + G)作用。

In order to find the eigenfunctions, we could compute the elements of M explicitly in our basis, and then call LAPACK or some similar code to find the eigenvectors and eigenvalues. For a three-dimensional calculation, this could mean finding the eigenvectors of a matrix with hundreds of thousands of elements on a side--daunting merely to store, much less compute. Fortunately, we only want to know a few eigenvectors, not hundreds of thousands, so we can use much less expensive iterative methods that don't require us to store M explicitly.

为了寻找本征函数，我们可以在特定矢量方向上准确计算数学算符M，然后调用的LAPACK或某些类似的处理数学公式的源代码，找出特征向量和特征值。对于一个三维结构的计算，这可能意味着，在一个矩阵一个方向上将有成千上万的特征向量——匮乏的是内存的存储空间，却计算量很少。幸运的是，我们只是想知道一些特征向量，而不是几十万，所以我们可以用更高效率的方法——迭代——不要求我们的程序在计算过程准确存储M值。

Iterative eigensolvers require only that one supply a routine to operate M on a vector

(function). Starting with an initial guess for the eigenvector, they then converge quickly to the actual eigenvector, stopping when the desired tolerance is achieved. There are many iterative eigensolver methods; we use a preconditioned block minimization of the Rayleigh quotient which is further described in the file src/matrices/eigensolver.c. In the Fourier basis, applying M to a function is relatively easy: the curls become cross products with i (k + G); the multiplication by 1/epsilon is performed by using an FFT to transform to the spatial domain, multiplying, and then transforming back with an inverse FFT. For more information and references on iterative eigensolvers, see the paper cited above.

迭代的本征计算方法，只要求提供一个程序块在一个向量（函数）操作计算 M。开始时候给定一个特征向量初始估算值，然后快速收敛趋近实际特征向量，当达到所需精度就停止计算。迭代的本征计算有很多方法，我们使用的预条件最小化的 Rayleigh 法。关于这个方法，在 src/matrices/eigensolver.c 深入描述。在傅立叶的基础上，用 M 作用到一个函数是相对简单：旋度计算变成和 i(k + G)相乘；与 1/epsilon 的乘法计算，是通过使用的 FFT 转换到空域，相乘，然后使用逆 FFT 转换到时域。想了解更多关于迭代本征值的计算方法，请参看上面引用文献的内容。

We also support a "targeted" eigensolver. A typical iterative eigensolver finds the p lowest eigenvalues and eigenvectors. Instead, we can find the p eigenvalues closest to a given frequency w0 by solving for the eigenvalues of (M - (w0/c)^2)^2 instead of M. This new operator has the same eigenvectors as M, but its eigenvalues have been shifted to make those closest to w0 the smallest.

我们也支持具有“针对性”的本征计算方法。一个典型的迭代本征计算可以找到最低的 P 值和特征向量。相反，我们可以通过求解(M - (w0/c)^ 2)^2 来代替 M 的计算，来计算最接近频率 w0 的特征值 P。这一新的算符操作和 M 具有相同的特征向量，但它的特征值已被转移到让那些最接近 w0 值中最小的。

The eigensolver we use is preconditioned, which means that convergence can be greatly improved by suppling a good preconditioner matrix. Finding a good preconditioner involves making an approximate inverse of M, and is something of a black art with lots of trial and error.

我们使用的本征计算方法的预条件，这意味着给定一个好的预矩阵可以得到很大的收敛。而找到一个好的预条件涉及 M 的近似逆，是一个需要有大量试验、错误的黑色艺术。

Dielectric Function Computation 介电函数的计算

The initialization of the dielectric function deserves some additional discussion, both because it is crucial for good convergence, and because we use somewhat complicated algorithms for performance reasons.

介电函数的初始化是很值得进一步讨论，不但因为它是计算过程中良好收敛性至关重要的因素，而且是性能的十分重要的原因，因为我们使用比较复杂的算法。

To ameliorate the convergence problems caused in a planewave basis by a discontinuous dielectric function, the dielectric function is smoothed (averaged) at the resolution of the grid. Another way of thinking about it is that this brings the average dielectric constant (over the grid) closer to its true value. Since different polarizations of the field prefer different averaging methods, one has to construct an effective dielectric

tensor at the boundaries between dielectrics, as described by the paper referenced above.

为了改善在平面波的基矢中造成的不连续的介电函数的收敛性问题，介电函数在分辨率可变的网格上被平滑化 （平均）。 另一种对于平均介电常数的近似计算考虑（通过网格）可以更接近它的真实值。由于在场中不同的偏振方向，更趋于使用不同的平均方法。这是在电介质之间特变的边界处构建一个更加真实有效的介电张量。这方面在上面提及的文献有对应所述。

This averaging has two components. First, at each grid point the dielectric constant (epsilon) and its inverse are averaged over a uniform mesh extending halfway to the neighboring grid points. (The mesh resolution is controlled by the mesh-size user input variable.) Second, for grid points on the boundary between two dielectrics, we compute the vector normal to the dielectric interface; this is done by averaging the "dipole moment" of the dielectric function over a spherically-symmetric distribution of points. The normal vector and the two averages of epsilon are then combined into an effective dielectric tensor for the grid point.

这个平均方法有两个组成。首先，在每个网格点，介电常数(epsilon)和其逆被平均化，统一到每个网格点的网格和扩展到相邻的网格点。 （网格分辨率控制的网格大小，这是由用户输入变量设置）。第二，两个电介质之间的边界上的网格点，我们计算向量介质的界面；这是通过对一个球对称分布点的平均介电函数的偶极矩。把法向量和两个平均的介电常数合并成一个有效的介电张量的网格点。

All of this averaging is handled by a subroutine in src/maxwell/ (see below) that takes as input a function epsilon(r), which returns the dielectric constant for a given position r. This epsilon function must be as efficient as possible, because it is evaluated a large number of times: the size of the grid multiplied by mesh-size3 (in three dimensions).

所有的数值平均由 src/maxwell/ 文件夹下代码完成（参见下文）。输入一个关于距离 r 的介电函数 epsilon(r)，返回每个位置的介电数值。 此 epsilon 函数必须是正确有效，因为它计算是花费大量时间：占用空间的大小为网格点数的三倍（三个维度）。

To specify the geometry, the user provides a list of geometric objects (blocks, spheres, cylinders and so on). These are parsed into an efficient data structure and are used to to provide the epsilon function described above. (All of this is handled by the libctlgeom component of libctl, described below.) At the heart of the epsilon function is a routine to return the geometric object enclosing a given point, taking into account the fact that the objects are periodic in the lattice vectors. Our first algorithm for doing this was a simple linear search through the list of objects and their translations by the lattice vectors, but this proved to be too slow, especially in supercell calculations where there are many objects. We addressed the performance problem in two ways. First, for each object we construct a bounding box, with which point inclusion can be tested rapidly. Second, we build a hierarchical tree of bounding boxes, recursively partitioning the set of objects in the cell. This allows us to search for the object containing a point in a time logarithmic in the number of objects (instead of linear as before).

对于要设置几何结构，用户要提供几何对象（块、球、圆柱等等） 的列表。这些设置被解析为有效的数据结构，用于提供上文所提到的 epsilon 的函数。（所有这一切都由 libctlgeom 组件的 libctl 库处理，下面接着介绍。epsilon 的核心函数，是返回包含给定对象周期的在格向量中考虑到的几何对象的程序。我们为了这样做，所以在第一次计算，算

法上是在列表中的设置几何对象和其转义的线性搜索，通过格子基矢。但这方法被实践证明十分缓慢，特别是有很多的对象的超元结构中。我们处理两种方法的性能问题。 第一，为每个对象构造进行快速测试，并且标示测试结果快的格子。第二，我们建立层次结构树的外框，以递归方式甄别单元格对象的集合。这使得我们可以搜索包含一个点，在一段时间中的对象数对数的对象（代替前面的线性计算）。

Code Organization 代码组织

The code is organized to keep the core computation independent of the user interface, and to keep the eigensolver routines independent of the operator they are computing the eigenvector of. The computational code is located in the src/ directory, with a few major subdirectories, described below. The Guile-based user interface is completely contained within the mpb-ctl/ directory.

程序代码组织，是为了保持核心计算独立于用户的接口，保持本征计算程序与计算在特征的向量的运算符独立。计算代码位于该 src/ 目录，与如下所述的几个主要的子目录。基于 Guile 的用户界面完全包含在 mpb-ctl/ 目录。

src/matrices/

This directory contains the eigensolver, in eigensolver.c, to which you pass an operator and it returns the eigenvectors. Eigenvectors are stored using the evectmatrix data structure, which holds p eigenvectors of length n, potentially distributed over n in MPI. See src/matrices/README for more information about the data structures. In particular, you should use the supplied functions (create_evectmatrix, etcetera) to create and manipulate the data structures, where possible.

此目录将包含本征计算算法，在 eigensolver.c 文件中。你通过一个运算符，并返回该特征向量。特征向量是存储在 evectmatrix 数据结构中，作为特征向量 p 的长度 n，可以分布在 MPI 上并行分布。 src/matrices/README 文件有关于数据结构的详细信息，请参阅。特别是，你应使用被提供的函数（create_evectmatrix，etcetera），在可能的情况下，去创建和操作数据的结构。

The type of the eigenvector elements is determined by scalar.h, which sets whether they are real or complex and single or double precision. This is, in turn, controlled by the --disable-complex and --enable-single parameters to the configure script at install-time. scalar.h contains macros to make it easier to support both real and complex numbers elsewhere in the code.

特征向量元素的类型取决于 scalar.h 文件，这个文件设置实数还是复数，单精度还是双精度。这在编译其间，在安装时配置脚本 configure --disable-complex 和--enable-single 控制。scalar.h 包含宏，以使它更易于支持在其他代码中的真实和复数。

Also in this directory is blasglue.c, a set of wrapper routines to make it convienient to call BLAS and LAPACK routines from C instead of Fortran.

也在此目录中是 blasglue.c，这个程序是对其调用 BLAS convienient 和 LAPACK 程序 C 语言而不是 Fortran 的封装。

src/util/

As its name implies, this is simply a number of utility routines for use elsewhere in the code. Of particular note is check.h, which defines a CHECK(condition, error-message) macro that is used extensively in the code to improve robustness. There are also debugging versions of malloc/free (which perform lots of paranoia tests, enabled by --enable-debug-malloc in configure), and MPI glue routines that allow the program to operate without the MPI libraries.

正如其名称所暗示，这是在其它地方在代码中使用的实用程序的集合。特别值得注意的是，广泛定义 check.h 条件错误消息宏，用于在代码中来提示代码的粗心和鲁莽。这也是生成调试版本的 分配/释放(在配置文件执行./configure --enable-debug-malloc），包括调用了 MPI 库，但也可以在不运行 MPI 下运行该程序。

src/matrixio

This section contains code to abstract I/O for eigenvectors and similar matrices, providing a simpler layer on top of the HDF5 interface. This could be modified to support HDF4 or other I/O formats.

本部分包含抽象的 I/O 特征向量及相似矩阵，在 HDF5 接口的基础上的代码，提供一个更简单的图层。这可以支持 HDF4 或其他 I/O 格式。

src/maxwell/

The maxwell/ directory contains all knowledge of Maxwell's equations used by the program. It implements functions to apply the Maxwell operator to a vector (in maxwell_op.c) and compute a good preconditioner (in maxwell_pre.c). These functions operate upon a representation of the fields in a transverse Fourier basis.

在 maxwell/ 目录中包含的麦克斯韦方程程序使用的所有知识。它实现了将麦克斯韦运算符应用于一个矢量（maxwell_op.c），计算一个好的预条件（maxwell_pre.c）的功能。这些函数运行时表示横向的傅里叶矢量中的形式。

In order to use these functions, one must first initialize a maxwell_data structure with create_maxwell_data (defined in maxwell.c) and specify a k point with update_maxwell_data_k. One must also initialize the dielectric function using set_maxwell_dielectric by supplying a function that returns the dielectric constant for any given coordinate. You can also restrict yourself to TE or TM polarizations in two dimensions by calling set_maxwell_data_polarization.

为了将这些函数，其中之一必须首先用 create_maxwell_data 初始化 maxwell_data 结构的（在 maxwell.c 中定义），并在 update_maxwell_data_k 指定 k 点。其中一个还必须初始化介电函数，使用 set_maxwell_dielectric，通过提供一个函数，返回的任何给定的坐标的“介电常数”。你还可以通过调用 set_maxwell_data_polarization，在两个维度上，来限制 TE 或 TM 偏振态。

This directory also contains functions maxwell_compute_dfield, etcetera, to compute the position-space fields from the Fourier-transform representation returned by the

eigensolver.

此目录还包含函数 maxwell_compute_dfield 等，计算本征解所返回的傅里叶变换表示的位置空间数值。

mpb-ctl/

Here is the Guile-based user interface code for the eigensolver. Instead of using Guile directly, this code is built on top of the libctl library as described in previous sections. This means that the user-interface code (in mpb.c) is fairly short, consisting of a number of small functions that are callable by the user from Guile.

下面是基于 Guile 的用户界面的本征解代码。相反不是直接使用 Guile，此代码是建造在 libctl 库的基础上(在上一节中所述)。 这意味着用户界面代码（在 mpb.c）是相当短，由一系列小是由从 Guile 用户可调用的函数组成的。

The core of the user interface is the file mpb.scm, the specifications file for libctl as described in the libctl manual. Actually, mpb.scm is generated by configure from mpb.scm.in (in order to substitute in parameters like the location of the libctl library); you should only edit mpb.scm.in directly. (You can regenerate mpb.scm simply by running ./config.status instead of re-running configure.)

用户界面的核心配置是在的文件 mpb.scm 中，如 libctl 手册中所述的规格文件一样。实际上，mpb.scm 是从文件 mpb.scm.in 中生成配置（以替代像 libctl 库的位置参数）；你只应直接编辑 mpb.scm.in 文件。（只需通过运行./config.status，您可以重新生成 mpb.scm ，而不用重新运行配置。）

The specifications file defines the data structures and subroutines that are visible to the Guile user. It also defines a number of Scheme subroutines for the user to call directly, like (run). It is often simpler and more flexible to define functions like this in Scheme rather than in C.

配置文件所定义数据结构和子程序，对于 Guile 用户是可见的。它还定义了一大批组织好的子程序直接调用，像(run)函数。 在 Scheme 定义这样的函数，通常是更简单、更灵活，而不是在 C 中定义。

All of the code to handle the geometric objects resides in libctlgeom, a set of Scheme and C utility functions included with libctl (see the file utils/README in the libctl package). (These functions could also be useful in other programs, such as a time-domain Maxwell's equation simulator.)

所有处理几何对象的代码是在 libctlgeom 中，在一组中包含 Scheme 和 C 的实用程序函数库 libctl 中 （请查阅 libctl 代码中的文件 utils/README）。(这些功能也可以在其他程序如时间域麦克斯韦方程模拟器中有调用用。Libctl 在 GPLv2 下释放)

后记：关于 MPB 程序代码分析解说，更多更详细请参考 http://hi.baidu.com/ghxandsky/
欢迎广大的朋友加入 MPB 程序的学习和研究。互助分享交流，臻于善。