Preconditioning Bandgap Eigenvalue Problems in Three Dimensional Photonic Crystals Simulations

Tsung-Ming Huang\textsuperscript{a}, Wei-Jen Chang\textsuperscript{b}, Yin-Liang Huang\textsuperscript{c}, Wen-Wei Lin\textsuperscript{d}, Wei-Cheng Wang\textsuperscript{e}, Weichung Wang\textsuperscript{f,}\textsuperscript{*}

\textsuperscript{a}Department of Mathematics, National Taiwan Normal University, Taipei 116, Taiwan. E-mail: min@math.ntnu.edu.tw.
\textsuperscript{b}Graduate Institute of Photonics and Optoelectronic, National Taiwan University, Taipei 106, Taiwan. E-mail: e4523744@gmail.com.
\textsuperscript{c}Department of Applied Mathematics, National University of Tainan, Tainan 700, Taiwan. E-mail: liang@mail.nutn.edu.tw.
\textsuperscript{d}Department of Mathematics, National Taiwan University, Taipei 106, Taiwan. E-mail: wulin@math.ntu.edu.tw.
\textsuperscript{e}Department of Mathematics, National Tsing Hua University, Hsinchu 300, Taiwan. E-mail: wangwc@math.nthu.edu.tw.
\textsuperscript{f}Department of Mathematics, National Taiwan University, Taipei 106, Taiwan. E-mail: wwang@math.ntu.edu.tw.

Abstract

To explore band structures of three-dimensional photonic crystals numerically, we need to solve the eigenvalue problems derived from the governing Maxwell equations. The solutions of these eigenvalue problems cannot be computed effectively unless a suitable combination of eigenvalue solver and preconditioner is chosen. Taking eigenvalue problems due to Yee’s scheme as examples, we propose using Krylov-Schur method and Jacobi-Davidson method to solve the resulting eigenvalue problems. For preconditioning, we derive several novel preconditioning schemes based on various preconditioners, including a preconditioner that can be solved by Fast Fourier Transform efficiently. We then conduct intensive numerical experiments for various combinations of eigenvalue solvers and preconditioning schemes. We find that the Krylov-Schur method associated with the Fast Fourier Transform based preconditioner is very efficient. It remarkably outperforms all other eigenvalue solvers with common preconditioners like Jacobi, Symmetric Successive Over Relaxation, and incomplete factorizations. This promising solver can benefit applications like photonic crystal structure optimization.

Key words: Three-dimensional photonic crystals, Maxwell’s equations, eigenvalue problems, preconditioning, Fast Fourier Transform, Krylov-Schur method, Jacobi-Davidson method, harmonic extraction.

*Corresponding author. Tel: +886-2-3366-2871; Fax +886-2-2391-4439
1. Introduction

Photonic crystals are periodic dielectric structures with a frequency region and specific band structures and they have been studied extensively from both theoretical and practical aspects. Numerical simulations that are based on the solutions of the time harmonic Maxwell’s equations have also played an important role among the studies. Some spatial discretization methods include the finite difference methods [11, 12, 32, 50], the finite volume methods [13, 14, 31], the finite element methods [7, 9, 10, 16, 27, 34], the Whitney form [6, 48], the covolume discretization [37], the mimetic discretization [26] and the edge element [35, 36, 38].

To determine the band structures of photonic crystals numerically, we need to find the first few branches of positive eigenvalues of the resulting generalized eigenvalue problems

\[ A\vec{e} = \lambda B\vec{e} \]  

(1)

where \( A \) is the matrix corresponding to the discretization of a certain operator and \( B \) is a mass matrix. Furthermore, due to the presence of a large null space associated with (1) [8, 12], these target eigenvalues are actually located in the interior of the spectrum of (1). The eigenvalue problem (1) can thus be solved by inverse power method [2, 11, 12, 21], various Lanczos [2, 40] or Arnoldi method [4]. These methods use shift-and-invert technique to compute interior eigenpairs and the computational cost for solving the corresponding linear systems can be excessive. On the other hand, we can also use Jacobi-Davidson method [1, 2, 3, 4, 15, 24, 41, 42] to find the interior target eigenvalues without using the shift-and-invert technique. However, Jacobi-Davidson method still needs to solve linear systems approximately in each of the iterations. In short, the main computational burden is to solve the resulting linear systems (iteratively, as the system is large) within the eigenvalue solvers.

One way to accelerate the processes for solving the linear systems is to find efficient preconditioners. However, it is not a trivial task, especially for three-dimensional photonic band structures. In [12], convergences of the conjugate gradient solver are compared with Jacobi, Successive Over Relaxation (SOR) or Symmetric Successive Over Relaxation (SSOR) preconditioner for computing photonic band structures. For two-dimensional photonic band structures, the SSOR preconditioner can accelerate convergence, while another two preconditioners are ineffective. For the three-dimensional cases, none of the three preconditioners are effective. In [24], a null space free Jacobi-Davidson method are proposed to compute three-dimensional photonic band structures.

To study how we may solve the eigenvalue problems arising in three-dimensional photonic crystals, we first propose using shift-and-invert Krylov-Schur method and Jacobi-Davidson method to solve the generalized eigenvalue problem (1). We then suggest several preconditioning schemes for the associated linear systems. The schemes not only incorporate various common preconditioners like Jacobi, SSOR, ILU (incomplete LU factorization), ICC (incomplete Cholesky factorization). In addition, we derive a preconditioner whose corresponding linear systems can be solved by FFT efficiently. To study the effects of all the
preconditioning schemes, we conduct intensive numerical experiments and compare their performance. We find that the combination of the shift-and-invert Krylov-Schur method with the FFT-based preconditioner can be very efficient for solving the target eigenvalue problems.

The paper is organized as follows. The Yee’s scheme used to discretize the Maxwell’s equations is introduced in Section 1.1. Two particular types of eigenvalue solvers are discussed in Section 2. Several preconditioning schemes are proposed in Section 3. Results of intensive numerical experiments regarding the proposed preconditioning schemes are presented and discussed in Section 4. Finally, we conclude the paper in Section 5.

1.1. Yee’s scheme and the model eigenvalue problems

To compute bandgap structures of three-dimensional photonic crystals, we consider the following Maxwell’s equations

\[
\begin{align*}
\nabla \times \mathbf{H} &= \varepsilon \partial_t \mathbf{E}, \\
\nabla \times \mathbf{E} &= -\mu_0 \partial_t \mathbf{H}, \\
\n\nabla \cdot (\varepsilon \mathbf{E}) &= 0, \\
\n\nabla \cdot \mathbf{H} &= 0.
\end{align*}
\]

(2)

By separating the time and space variables and eliminating the magnetic field \( \mathbf{H} \), the above differential equations become

\[
\begin{align*}
\nabla \times \nabla \times \mathbf{E} &= \lambda \varepsilon \mathbf{E} \\
\n\nabla \cdot (\varepsilon \mathbf{E}) &= 0
\end{align*}
\]

(3)

for the electric field \( \mathbf{E} \), where \( \lambda = \mu_0 f^2 \) is the unknown eigenvalue, \( \mu_0 \) and \( f \) stands for magnetic constant and frequency, respectively. The permittivity \( \varepsilon \) is a material dependent constant. Note that the degenerate elliptic operator \( \nabla \times \nabla \times \) is self-adjoint and non-negative and (3) is simply an elliptic interface eigenvalue problem.

To solve equation (3), we use Yee’s scheme to discretize the equation and then derive the resulting generalized eigenvalue problems. Yee’s scheme \([50]\) is briefly described here and detailed derivation can be found in, for example, \([24, 50]\).

Since the photonic crystals consist of dielectric materials fabricated in periodic structure, Bloch’s Theorem \([28]\) suggests that the eigenfunctions of (3) can be written as

\[
\mathbf{E}(\mathbf{x}) = e^{i \mathbf{k} \cdot \mathbf{x}} \mathbf{e}(\mathbf{x}),
\]

(4)

for a vector \( \mathbf{k} \) in the first Brillouin zone. The function \( \mathbf{e} \) satisfies the periodic condition

\[
\mathbf{e}(\mathbf{x} + \mathbf{a}_\ell) = \mathbf{e}(\mathbf{x})
\]

(5)

for \( \ell = 1, 2, 3 \). Here the lattice translation vectors \( \mathbf{a}_\ell \) span the primitive cell which extends periodically to form the photonic crystal. Now, we can rewrite Eq. (3) as

\[
e^{-i \mathbf{k} \cdot \mathbf{x}} \nabla \times \nabla \times e^{i \mathbf{k} \cdot \mathbf{x}} \mathbf{e} = \lambda \varepsilon \mathbf{e}.
\]

(6)
Letting
\[ h = \nabla \times e^{ik \cdot x} \]
and substituting (7) into (6), we get
\[ \nabla \times h = \lambda \varepsilon e^{ik \cdot x}. \]

In Yee’s scheme, equations (7) and (8) are discretized on centers of cell faces and centers of cell edges, respectively. First, we rewrite Eq. (7) as
\[
\begin{align*}
\partial_y e^{ik \cdot x_3} e_3 &= h_1, \\
\partial_z e^{ik \cdot x_2} e_2 &= h_2, \\
\partial_x e^{ik \cdot x_1} e_1 &= h_3
\end{align*}
\]
and discretize (9) by central finite differences at the face points. By further using the periodic property of \( e_1, e_2 \) and \( e_3 \), the discretizations of Eq. (9) can be represented as
\[ \vec{h} = G^* D \vec{e} \]
over the \( 3n \) grid points that \( n = n_1 n_2 n_3 \) and \( n_1, n_2, \) and \( n_3 \) are the number of grid points in the \( x, y, \) and \( z \) directions, respectively. Here \( G \) and \( D \) belong to \( \mathbb{C}^{3n \times 3n} \), \( G \) is an off-diagonal block matrix that each block is a sparse band matrix, and \( D \) is a diagonal matrix.

Next, we rewrite (8) as
\[
\begin{align*}
\partial_y h_3 - \partial_z h_2 &= \lambda \varepsilon_1 e^{ik \cdot x_1} e_1, \\
\partial_z h_1 - \partial_x h_3 &= \lambda \varepsilon_2 e^{ik \cdot x_2} e_2, \\
\partial_x h_2 - \partial_y h_1 &= \lambda \varepsilon_3 e^{ik \cdot x_3} e_3,
\end{align*}
\]
and discretize (11) by central finite differences at edge points. Again, by using the periodic property of \( e_1, e_2 \) and \( e_3 \), the discretization of Eq. (11) can be written as
\[ D^* G \vec{h} = \lambda B \vec{e}, \]
where \( B \) is a positive diagonal matrix whose entries are the dielectric coefficients \( \varepsilon \) that are depends on the photonic crystal structure spacially.

Finally, by substituting (10) into (12), the discretization of (6) at edges forms the following target generalized eigenvalue problem
\[ A \vec{e} = \lambda B \vec{e}, \]
where
\[ A = D^* G G^* D. \]
A sample sparsity of the matrix \( A \) is shown in Figure 1. For more details regarding the discretization scheme, we refer readers to [24].

2. Eigenvalue Solvers

We propose using the Krylov-Schur method and the Jacobi-Davidson method to solve the resulting generalized Hermitian eigenvalue problem (13). In this section, we introduce these numerical schemes and provide references for detailed descriptions.
2.1. Krylov-Schur method

To solve the generalized Hermitian eigenvalue problem (13), we introduce a shift-and-invert Lanczos method and the associated restarting schemes. In particular, we suggest using the Krylov-Schur restarting procedure proposed in [45, 46] due to its stability and efficiency.

As the dimension of zero eigenspace of \((A, B)\) is one-third of dimension (i.e. \(n\)) [24] and we are interested in the smallest positive eigenvalues, we first rewrite (13) as

\[
(A - \sigma B)^{-1}B\vec{e} = (\lambda - \sigma)^{-1}\vec{e}
\]  

(14)

for a given target \(\sigma > 0\). The Lanczos method can then be applied to solve the eigenvalues that are close to \(\sigma\). The basic recursion for applying Lanczos method to (14) is

\[
(A - \sigma B)^{-1}BV_j = V_jT_j + \beta_jv_{j+1}e_j^T.
\]  

(15)

Here the basis \(V_j\) is \(B\)-orthogonal and the columns of \(V_j\) span a certain Krylov subspace. \(T_j\) is a real symmetric tridiagonal matrix defined by

\[
T_j = \begin{bmatrix}
\alpha_1 & \beta_1 \\
\beta_1 & \alpha_2 & \ddots \\
& \ddots & \ddots & \beta_{j-1} \\
& & \beta_{j-1} & \alpha_j
\end{bmatrix},
\]

or equivalently,

\[
(A - \sigma B)^{-1}Bv_j = \alpha_jv_j + \beta_jv_{j-1} + \beta_jv_{j+1}.
\]

By using the condition \(V_j^*BV_j = I_j\) and letting \(t_j = (A - \sigma B)^{-1}Bv_j - \alpha_jv_j - \beta_jv_{j-1} = \beta_jv_{j+1}\), we have

\[
\alpha_j = v_j^*B(A - \sigma B)^{-1}Bv_j \quad \text{and} \quad \beta_j^2 = t_j^*Bt_j.
\]

Further, by finding the eigenpair \((\theta_k, s_k)\) of \(T_j\), an approximate eigenpair \((\lambda_k, x_k)\) of \((A, B)\) can be computed by

\[
\lambda_k = \sigma + \frac{1}{\theta_k} \quad \text{and} \quad x_k = V_js_k.
\]

Furthermore, the corresponding residual is

\[
\begin{align*}
r_k &= AV_js_k - \lambda_kBV_js_k = (A - \sigma B)V_js_k - \theta_k^{-1}BV_js_k \\
&= -\theta_k^{-1}[BV_j - (A - \sigma B)V_jT_j]s_k \\
&= -\theta_k^{-1}(A - \sigma B)[(A - \sigma B)^{-1}BV_j - V_jT_j]s_k \\
&= -\theta_k^{-1}\beta_j(e_j^Ts_k)(A - \sigma B)v_{j+1}
\end{align*}
\]

which implies that the norm of \(r_k\) is small whenever \(|\beta_j(e_j^Ts_k)/\theta_k|\) is small.
In principle, we can keep expanding the Lanczos decomposition in (15) until the Ritz eigenpairs converge to the wanted eigenpairs. However, the feasibility of such expanding process is limited by the amount of memory to store $V_j$ and the increasing cost of orthogonalization as the index $j$ increases. The possibility of losing the numerical orthogonality in $V_j$ therefore, we restart the Lanczos process once the index $j$ becomes large. If the dimension of the Krylov subspace is larger than “ncv”, then the process of Lanczos decomposition will be restarted.

For the restarting process, we can use an explicit restart scheme [19], or we can use an implicit restart scheme [33, 43] that the Lanczos process is combined with the implicitly shifted QR algorithm. As the implicitly QR approach generates the Krylov subspace associated with all the approximate Schur vectors, the implicitly restart usually performs better than explicit restart. ARPACK [30] is a very successful implementation of the implicitly restarted Arnoldi algorithm. It has been applied in many applications and remains a popular choice for solving eigenvalue problems. However, these implicitly restart type schemes can suffer from numerical instability due to rounding errors. Consequently, such methods may compromise the convergence performance or even fail to obtain the wanted eigenvalues. As shown in Section 4, the target eigenvalue problems can be trapped by using such implicitly restart scheme. While remedies for the numerical difficulties in the purging and deflating processes of implicitly restart do exist [29], there is another improved restart option. Stewart proposes the Krylov-Schur method [18, 45, 46] that relaxes the need to preserve the structure of the Lanczos decomposition and therefore ease the complications of the purging and deflating. In addition, Wu and Simon consider Krylov-Schur method for symmetric eigenvalue problems [49]. By iteratively expanding and contracting a Krylov-Schur decomposition, the method incorporates an effective and robust restart scheme in a simple and efficient manner. The Krylov-Schur method is briefly described as follow.

First, we define the order $j + p$ Lanczos decomposition as

$$(A - \sigma B)^{-1}BV_{j+p} = V_{j+p}T_{j+p} + \beta_{j+p}v_{j+p+1}e_{j+p}^T$$

(16)

and let

$$T_{j+p} = U_{j+p}D_{j+p}U_{j+p}^T = \begin{bmatrix} U_j & U_p \end{bmatrix} \text{diag} (D_j, D_p) \begin{bmatrix} U_j^T \\ U_p^T \end{bmatrix}$$

(17)

be a Schur decomposition of $T_{j+p}$. The diagonal elements of $D_j$ and $D_p$ contain the $j$ wanted and $p$ unwanted Ritz values, respectively. Using (16) and (17), we have

$$(A - \sigma B)^{-1}B(V_{j+p}U_{j+p})$$

$$= (V_{j+p}U_{j+p})(U_{j+p}^T T_{j+p} U_{j+p}) + \beta_{j+p}v_{j+p+1}(e_{j+p}^T U_{j+p})$$

and thus a Krylov decomposition of order $j$ becomes

$$(A - \sigma B)^{-1}B\tilde{V}_j = \tilde{V}_jD_j + v_{j+p+1}t_j^T = \begin{bmatrix} \tilde{V}_j \\ \tilde{v}_{j+1} \end{bmatrix} \begin{bmatrix} D_j \\ t_j^T \end{bmatrix},$$

(18)
where $\tilde{V}_j \equiv V_{j+p}U_j$, $\tilde{v}_{j+1} = v_{j+p+1}$, and $t_j^T \equiv \beta_{j+p} e_{j+p}^T U_j$. The new vectors $\tilde{v}_{j+2}, \ldots, \tilde{v}_{j+p+1}$ are computed sequentially starting from $\tilde{v}_{j+2}$ with

$$
(A - \sigma B)^{-1} B\tilde{V}_{j+1} = \begin{bmatrix} \tilde{V}_{j+1} & \tilde{v}_{j+2} \end{bmatrix} \begin{bmatrix} D_j & t_j \\ t_j^T & \tilde{\alpha}_{j+1} \\ 0 & \tilde{\beta}_{j+1} \end{bmatrix},
$$

or equivalently

$$
(A - \sigma B)^{-1} B\tilde{v}_{j+1} = \tilde{V}_j t_j + \tilde{\alpha}_{j+1} \tilde{v}_{j+1} + \tilde{\beta}_{j+1} \tilde{v}_{j+2},
$$

for

$$
\tilde{\alpha}_{j+1} = \tilde{v}_{j+1}^* B (A - \sigma B)^{-1} B \tilde{v}_{j+1}.
$$

Consequently a new Krylov decomposition of order $j + p$ can be generated by

$$
(A - \sigma B)^{-1} B\tilde{V}_{j+p} = \tilde{V}_{j+p} T_{j+p} + \tilde{\beta}_{j+p} \tilde{v}_{j+p+1} e_{j+p}^T,
$$

(19)

where

$$
\tilde{T}_{j+p} = \begin{bmatrix} D_j & t_j \\ t_j^T & \tilde{\alpha}_{j+1} & \tilde{\beta}_{j+1} \\ \tilde{\beta}_{j+1} & \tilde{\alpha}_{j+2} & \ddots \\ \vdots & \ddots & \tilde{\beta}_{j+p-1} \\ \tilde{\beta}_{j+p-1} & \tilde{\alpha}_{j+p} \end{bmatrix}.
$$

(20)

Eq. (19) has the same form as Eq. (16) and the above Krylov-Schur update process can be repeated until the desired eigenpairs are convergent.

We summarize this shift-and-invert Krylov-Schur method for solving the generalized Hermitian eigenvalue problem (13) in Algorithm 1. Note that the for-loop in lines 5-15 of Algorithm 1 extends Eq. (21) to a new Krylov-Schur decomposition of order $j + p$

$$
(A - \sigma B)^{-1} B\tilde{V}_{j+p} = \tilde{V}_{j+p} T_{j+p} + \tilde{\beta}_{j+p} \tilde{v}_{j+p+1} e_{j+p}^T,
$$

where $T_{j+p}$ is defined in (20). In addition, Algorithm 1 can be applied only to Hermitian-definite matrix pencils like (13), in which $A$ and $B$ are Hermitian and $B$ is positive definite.

### 2.2. Jacobi-Davidson methods

Another option to solve the eigenvalue problem (13) is the Jacobi-Davidson method (JD). JD is an iterative method that each iteration contains a Rayleigh-Ritz procedure. The Rayleigh-Ritz extraction is usually used to determine approximate eigenpair $(\theta, u)$ with $u \in V$ and $\|u\|_B = 1$ when the $B$-orthonormal basis $V$ of the searching subspace $\mathcal{V}$ is generated.

The Rayleigh-Ritz method works well for extraction of well-separated exterior eigenvalues of Hermitian matrices, but may fail for clustered or interior
Algorithm 1: Shift-and-invert Krylov-Schur method for solving $Ax = \lambda Bx$

**Input:** Coefficient matrices $A$ (Hermitian), $B$ (Hermitian and positive definite), the target $\sigma$, the number of desired eigenvalues $j$, and restarting number $\text{ncv}$ that $\text{ncv} = j + p$.

**Output:** The desired $j$ eigenpairs.

1: Build an initial Lanczos decomposition of order $j+p$:

$$(A - \sigma B)^{-1}BV_{j+p} = V_{j+p}T_{j+p} + \beta_{j+p}v_{j+p+1}e_{j+p}^T,$$

2: repeat

3: Compute Schur decomposition of $T_{j+p}$:

$$T_{j+p} = \begin{bmatrix} U_j & U_p \end{bmatrix} \begin{bmatrix} \text{diag}(D_j, D_p) & \text{diag}(D_j, D_p) \\ \text{diag}(D_p, D_p) & \text{diag}(D_p, D_p) \end{bmatrix} \begin{bmatrix} U_j^T \\ U_p^T \end{bmatrix},$$

where the diagonal elements of $D_j$ and $D_p$ contain the $j$ wanted and $p$ unwanted Ritz values, respectively.

4: Set $V_j := V_{j+p}U_j$, $v_{j+1} := v_{j+p+1}$ and $e_{j+p}^T := \beta_{j+p} e_{j+p}^T U_j$ to get truncated Krylov-Schur decomposition of order $j$:

$$(A - \sigma B)^{-1}BV_j = V_jD_j + v_{j+1}e_j^T.$$  \(21\)

5: for $k = j+1, \ldots, j+p$ do

6: Solve linear system

$$(A - \sigma B)q = Bv_k.$$  \(22\)

7: if $k = j+1$ then

8: Set $q := q - V_j t_j$.

9: else

10: Set $q := q - \beta_{k-1}v_{k-1}$.

11: end if

12: Compute $\alpha_k = e_k^T B q$ and reset $q := q - \alpha_k v_k$.

13: $B$-reorthogonalize $q$ to $v_1, \ldots, v_{k-1}$ if necessary.

14: Compute $\beta_k = \sqrt{q^T B q}$ and $v_{k+1} = q/\beta_k$.

15: end for

16: Decompose $T_{j+p} = S_{j+p} \Theta_{j+p} S_{j+p}^*$ to get the approximate eigenvalues.

17: Test for convergence.

18: until all the $j$ wanted eigenpairs are convergent
eigenvalues [22, 44]. For the generalized eigenvalue problem in (13), we are interested in the smallest positive eigenvalues which are interior eigenvalues as high-multiplicity zero eigenvalue exists. Consequently, we may use the harmonic Rayleigh-Ritz extraction [22] to circumvent the potential pitfalls and to accelerate the convergence. Main idea of the harmonic Rayleigh-Ritz with target $\sigma$ is outline as follows. Since $B$ in (13) is a positive diagonal matrix, the generalized eigenvalue problem in (13) can be rewritten as

$$\tilde{A} y \equiv (B^{-1/2} AB^{-1/2})(B^{1/2} \bar{e}) = \lambda y$$  \hspace{1cm} (23)

and

$$(\tilde{A} - \sigma I)^{-1} y = (\lambda - \sigma)^{-1} y$$

which implies that eigenvalues of $\tilde{A}$ near $\sigma$ are exterior eigenvalues of $(\tilde{A} - \sigma I)^{-1}$. Consider the Galerkin condition on the residual of the shift-and-inverse matrix, i.e., find $\bar{\theta} \in \mathbb{R}$, $\tilde{y} \in \tilde{V}$ to satisfy

$$\left( (\tilde{A} - \sigma I)^{-1} \tilde{y} - (\bar{\theta} - \sigma)^{-1} \tilde{y}, (\tilde{A} - \sigma I)^{*}(\tilde{A} - \sigma I) \tilde{v} \right) = 0, \forall \tilde{v} \in \tilde{V} \hspace{1cm} (24)$$

where $\tilde{V}$ is the search space. Let the columns of $\tilde{V}$ be an orthonormal basis of $\tilde{V}$. Then, Eq. (24) is equivalent to the following generalized eigenvalue problem

$$\tilde{V}^* (\tilde{A} - \sigma I)^* (\tilde{A} - \sigma I) \tilde{V} = (\bar{\theta} - \sigma) \tilde{V}^* (\tilde{A} - \sigma I)^* \tilde{V}, \quad \tilde{y} = \tilde{V} s. \hspace{1cm} (25)$$

By the definition of $\tilde{A}$ in (23), Eq. (25) can be represented as

$$(AV - \sigma BV)^* B^{-1} (AV - \sigma BV) s = (\bar{\theta} - \sigma) V^* (A^* - \sigma B) V s, \quad V = B^{-1/2} \tilde{V}.$$

We outline the JD equipped with the harmonic Rayleigh-Ritz in Algorithm 2. Main components of the algorithm are highlighted as follows.

- Switches between the standard and the harmonic Rayleigh-Ritz are incorporated in line 4 to form the projected system, line 10 to compute of Ritz values, and line 24 to update of the projected system.
- The convergence of the eigenvalue system is checked in line 16.
- In the if-loop in line 17, an adaptive shift setting scheme suggested in [15] is adopted that the shift $\tau$ in the correction equation is kept fixed as the target value $\sigma$, until the residual is smaller than a user defined threshold $10\varepsilon_{shft}$.
- The correction equation is solved in line 22.
- The restarting is controlled in by the while-loop in line 9 (in terms of ncv) and then performed in line 31 if necessary.
- The locking scheme is performed in line 35.
In line 22 of Algorithm 2, the correction equation
\[
\left(I - \frac{pt^*}{u^*p}\right)(A - \tau B)\left(I - \frac{up^*}{u^*p}\right)t = -r \tag{26}
\]
is solved approximately in each iteration to expand the search subspace. As solving the correction equation is the most expensive part of the algorithm, various acceleration approaches, e.g. a shift technique [15], are used in practice. In addition, as suggested in [15, 41], (26) can be solved by using an iterative method with preconditioner
\[
\mathcal{M}_p \equiv \left(I - \frac{pt^*}{u^*p}\right) \mathcal{M} \left(I - \frac{up^*}{u^*p}\right),
\]
where \(\mathcal{M}\) is an approximation of \((A - \tau B)\). In each of the iterative steps for solving (26), it needs to solve the linear system
\[
\mathcal{M}_p z = y
\]
for a certain given vector \(y\) and \(z \perp Bu\). The solution \(z\) can be computed as
\[
z = \mathcal{M}^{-1} y - \zeta \mathcal{M}^{-1} Bu, \tag{27}
\]
where
\[
\zeta = \frac{u^*BM^{-1}y}{u^*BM^{-1}Bu}.
\]
Therefore, it has to solve only one linear system which is \(Md = y\) in each iterative step. The vector \(\mathcal{M}^{-1} Bu\) and inner product \(u^*BM^{-1}Bu\) are computed only once in all steps of the iterative process for solving (26). In addition, to compute the approximated solution of the correction equation (26), we adopt the heuristic strategies introduced in the appendix. If the parameters within the heuristic strategies are chosen suitably, the correction equation solver can be convergent and even be efficient.
3. Preconditioning Schemes

To solve the linear system (22) in the Krylov-Schur method or to solve the correction equation (26) in the Jacobi-Davidson method, we need to solve linear systems

\[(A - \tau B)z = b.\]  

(28)

If we can find a suitable preconditioner \(M \approx (A - \tau B)\), then the computational costs for finding the target eigenpairs can be reduced. In addition to Jacobi, SSOR, ILU and ICC preconditioners, in this section, we shall use two transformations to show how to utilize FFT to solve the preconditioning linear system corresponding to the three-dimensional Maxwell’s operator.

3.1. An FFT based preconditioner

The main idea of our preconditioning scheme for (28) is to take the preconditioner \(M = (A - \tau \varepsilon_0 I) \approx (A - \tau B)\) and then solve the preconditioned system

\[(A - \tau \varepsilon_0 I)z = b\]  

(29)

by using FFT. Here, we define a constant variable parameter \(\varepsilon_0\) by setting

\[\varepsilon_0 = \frac{1}{3n_1n_2n_3} \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \sum_{k=0}^{n_3-1} [\varepsilon_1(i,j,k) + \varepsilon_2(i,j,k) + \varepsilon_3(i,j,k)].\]

Since \(B\) is a diagonal matrix that contains nonzero elements \(\varepsilon_1(i,j,k), \varepsilon_2(i,j,k),\) and \(\varepsilon_3(i,j,k),\) the constant \(\varepsilon_0\) can be viewed as an average of the nonzero elements of \(B\). In other words, the matrix \(\varepsilon_0 I\) is an approximation of \(B\).

Now, we show how the linear system (29) can be solved efficiently by FFT. As the matrix \(A\) is actually the discretization of the operator \(e^{-i k \cdot x} \nabla \times \nabla \times e^{i k \cdot x}\), we can treat the linear system (29) as the discretization of the equation

\[(e^{-i k \cdot x} \nabla \times \nabla \times e^{i k \cdot x} - \varepsilon_0 \tau)z = b\]

or

\[\nabla \times \nabla \times e^{i k \cdot x}z - \varepsilon_0 \tau e^{i k \cdot x}z = e^{i k \cdot x}b.\]  

(30)

Using the fact that

\[\nabla \times (\nabla \times e^{i k \cdot x}z) = \nabla (\nabla \cdot e^{i k \cdot x}z) - \nabla^2 (e^{i k \cdot x}z),\]

the left hand side of Eq. (30) can be rewritten as

\[\nabla (\nabla \cdot e^{i k \cdot x}z) - \nabla^2 (e^{i k \cdot x}z) - \varepsilon_0 \tau e^{i k \cdot x}z,\]

or

\[(-\nabla^2 - \varepsilon_0 \tau) e^{i k \cdot x}z + \nabla (\nabla \cdot e^{i k \cdot x}z).\]  

(31)

The first term of (31) is composed by the Laplace operator \(\nabla^2\) and a constant \(\varepsilon_0 \tau\). It is well known that the linear system corresponding to the discretization of \((-\nabla^2 - \varepsilon_0 \tau)\) can be efficiently solved by using FFT.
Algorithm 2 Jacobi-Davidson method for solving $Ax = \lambda Bx$

**Input:** Coefficient matrices $A$ and $B$, the number of desired eigenvalues $\ell$, an initial vector with unit $B$-norm $V_1$, target $\sigma$, tolerances $\varepsilon_{\text{conv}}, \varepsilon_{\text{harm}}, \varepsilon_{\text{shft}}$, restarting number $\text{ncv}$, and number of Ritz vectors $m$.

**Output:** The desired eigenpairs $(\lambda_j, x_j)$ for $j = 1, \ldots, \ell$.

1. Set $V_\ell = [\ ]$, $k = 1$ and $r_0 = e_1$.
2. for $j = 1$ to $\ell$ do
3. repeat
4. if $(j = 1$ and $\|r_{k-1}\| \geq \varepsilon_{\text{harm}})$ then
5. Compute $W_k = AV_k$ and $M_k = V_k^* W_k$.
6. else
7. Compute $W_k = AV_k - \sigma BV_k$.
8. end if
9. while $(k \leq \text{ncv}$ and $\|r_{k-1}\| \geq \varepsilon_{\text{conv}})$ do
10. if $(j = 1$ and $\|r_{k-1}\| \geq \varepsilon_{\text{harm}})$ then
11. Compute the eigenpairs $(\theta_i, s_i)$ of $M_k s = \theta s$ with $\|s_i\| = 1$ and $\sigma < \theta_1 \leq \theta_2 \leq \cdots$.
12. else
13. Compute the eigenpairs $(\mu_i, s_i)$ of $W_k^* B^{-1} W_k s = \mu W_k^* V_k s$ and $\theta_1 = \sigma + \mu_i$ with $\|s_i\| = 1$ and $\sigma < \theta_1 \leq \theta_2 \leq \cdots$.
14. end if
15. Compute $u_k = V_k s_1$, $p_k = B u_k$, $r_k = (A - \theta_1 B) u_k$.
16. if $(\|r_k\| < \varepsilon_{\text{conv}})$ set $\lambda_j = \theta_1$, $x_j = u_k$, $k := k + 1$. Go to line 9.
17. if $(j = 1$ and $\|r_k\| \geq \varepsilon_{\text{shft}})$ or $(j \geq 2$ and $\|r_k\| \geq 10 \varepsilon_{\text{shft}})$ then
18. $\tau = \sigma$.
19. else
20. $\tau = \theta_1$.
21. end if
22. Solve the correction eq. $\left( I - \frac{p_k u_k^*}{u_k^* p_k} \right) \left( A - \tau B \right) \left( I - \frac{u_k p_k^*}{u_k^* p_k} \right) t_k = -r_k$
approximately for $t_k \perp_B u_k$.
23. $B$-orthogonalize $t_k$ against $[V_k, V]$; set $v_{k+1} = t_k / \|t_k\|_B$.
24. if $(j = 1$ and $\|r_k\| \geq \varepsilon_{\text{harm}})$ then
25. Compute $w_{k+1} = Av_{k+1}$, $M_{k+1} = \left[ \begin{array}{cc} M_k & V_k^* w_{k+1} \\ v_{k+1}^* \big( v_{k+1}^* w_{k+1} \big) & v_{k+1}^* w_{k+1} \end{array} \right]$.
26. else
27. Compute $w_{k+1} = Av_{k+1} - \sigma B v_{k+1}$.
28. end if
29. Expand $V_{k+1} = [V_k, v_{k+1}]$ and $W_{k+1} = [W_k, w_{k+1}]$. Set $k := k + 1$.
30. end while
31. if $(\|r_{k-1}\| \geq \varepsilon_{\text{conv}})$ then
32. Set $V_m := V_{k-1} [s_1, \cdots, s_m]$, $r_{m-1} := r_{k-1}$ and $k := m$.
33. end if
34. until wanted $j$th eigenpair is convergent
35. Set $V_j = [V_x, x_j]$, $V_{m-1} = V_{k-1} [s_2, \cdots, s_m]$, $k = m - 1$ and $r_{k-1} = e_1$.
36. end for
For the second term of (31), which involves the unknown \( z \), we can actually rewrite it in terms of \( b \) and then move it to the right hand side of Eq. (30). Using (30) and the fact \( \nabla \cdot (\nabla \times \mathbf{v}) = 0 \) for any \( \mathbf{v} \), we have

\[
\nabla \cdot (\nabla \times e^{ik \cdot x} z) - \varepsilon_0 \tau \nabla \cdot (e^{ik \cdot x} z) = \nabla \cdot (e^{ik \cdot x} b)
\]

and

\[
-\varepsilon_0 \tau \nabla \cdot (e^{ik \cdot x} z) = \nabla \cdot (e^{ik \cdot x} b).
\]

By further using the results described in (31) and (32), we can rewrite Eq. (30) as

\[
(-\nabla^2 - \varepsilon_0 \tau) e^{ik \cdot x} z = e^{ik \cdot x} b + \frac{1}{\varepsilon_0 \tau} \nabla (\nabla \cdot e^{ik \cdot x} b).
\]

Discretizing Eq. (33) and letting \( \tilde{z} = e^{ik \cdot x} z \) and \( \tilde{b} = e^{ik \cdot x} b + \frac{1}{\varepsilon_0 \tau} \nabla (\nabla \cdot e^{ik \cdot x} b) \), we can obtain a new linear system

\[
(\tilde{A} - \tau \varepsilon_0 I) \tilde{z} = \tilde{b}
\]

that can be solved by FFT as the boundary condition of (33) is periodic. Note that \( \tilde{A} \) is the discretization method of \(-\nabla^2\) and \( \tilde{z} = Dz \), where \( D \) is the diagonal discretization matrix of \( e^{ik \cdot x} \).

Clearly, the solution of (34) can easily lead to the solution of target linear system (29). We conclude this section by two remarks: (i) solving (34) by FFT is as efficient as the complexity \( O(n \log(n)) \); (ii) \((\tilde{A} - \tau \varepsilon_0 I)\) is not singular as \( \varepsilon_0 \tau \) is not an eigenvalue of (33).

3.2. Eigenvalue solvers and preconditioners

Now, we describe how we may combine the Krylov-Schur method (KS) and Jacobi-Davidson method with various preconditioners, so that the target eigenvalue problem (13) can be solved efficiently. In particular, we need to find suitable preconditioners for the matrix in the form of \( (A - \tau B) \), where \( \tau = \sigma \) in Eq. (22) for KS and \( \tau \) is defined in Eq. (26) for JD.

\[
\tau = \begin{cases} 
\sigma \ (\text{i.e. the shift value in (22) for KS}), \\
\theta \ (\text{i.e. the Ritz value in (26) for JD}).
\end{cases}
\]

In addition to some common preconditioners like Jacobi, SSOR(\( \omega \)), ILU(\( \ell \)), and ICC(\( \ell \)), we are especially interested in exploring the performance of the preconditioning scheme proposed in Section 3.1. Note that \( \omega \) is a tunable parameter of SSOR that \( 0 < \omega < 2 \) and \( \ell \) is a tunable fill-in level used in ILU and ICC.

We can now combine the JD and KS eigenvalue solvers described in Section 2 with the following preconditioning schemes.
• **PS1:** using a preconditioner $\mathcal{M} \approx (A - \tau B)$ to solve (22) or (26) approximately. In this scheme, $\mathcal{M}$ may be chosen as Jacobi, SSOR($\omega$), ICC($\ell$), and ILU($\ell$) preconditioners.

• **PS2:** solving the $(\tilde{A} - \tau \varepsilon_0 I)$-preconditioned system (34) with lower complexity. We can also find an approximate solution of the preconditioned system (34) to reduce the computation cost. For example, we can solve the linear system $\tilde{\mathcal{M}}(Dz) = \tilde{b}$ by letting the preconditioner $\tilde{\mathcal{M}} \approx (\tilde{A} - \tau \varepsilon_0 I)$. Some possible choices of $\tilde{\mathcal{M}}$ include Jacobi, SSOR($\omega$), ICC($\ell$), and ILU($\ell$).

• **PS3:** solving the $(\tilde{A} - \tau \varepsilon_0 I)$-preconditioned system (34) directly by FFT with $O(n \log(n))$ complexity. When we use GMRES with the preconditioned system (34) to solve the linear systems (22), an accurate solution of (34) can be obtained by applying FFT.

To distinguish the preconditioners, we use the notation like PS1-SSOR(0.75) to indicate that the system $(A - \tau B)$ is preconditioned by SSOR with $\omega = 0.75$.

### 4. Numerical Results

We conduct numerical experiments to evaluate performances of the eigenvalue solvers and preconditioners described in Section 3. Figure 2 illustrates the target periodic dielectric structure within a primitive cubic cell that is considered in the numerical experiments. The structure consists of dielectric spheres with radius $r$ that are connected by circular cylinders with radius $s$. Here $r/a = 0.345, s/a = 0.11$, and $a$ is the edge length of the cube. Inside the structure is the dielectric material with permittivity contrast $\epsilon_i/\epsilon_o = 13$. We aim for solving the ten smallest positive eigenvalues $\lambda_1, \ldots, \lambda_{10}$ with $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{10}$ to determine the corresponding bandgap structures.

We code the JD solver and the FFT preconditioning scheme by using Fortran 95 programming language. Both of the standard Jacobi-Davidson (JD-S) [25, Figure 4] and the Jacobi-Davidson with harmonic extraction (JD-H) (i.e. Algorithm 2) are implemented and tested. Note that JD-S is similar to JD-H, except that (i) all of the $\theta_i$’s are computed by the Rayleigh-Ritz extraction in line 11 of Algorithm 2 and (ii) the shift $\tau$ in line 22 is set to be $\theta_1$ for all iterations. We use the KS solver included in the software package “Scalable Library for Eigenvalue Problem Computations” (SLEPc) [17, 20]. The preconditioners Jacobi, SSOR, ILU and ICC are integrated into these two eigenvalue solvers via the interfaces provided by “Portable, Extensible Toolkit for Scientific Computation” (PETSc) [5]. All the numerical experiments are conducted in a workstation that is equipped with two Intel Quad-Core Xeon E5355 2.66 GHz CPU, 16 Gigabyte main memory, and RedHat Linux operation system. Intel Fortran and C Compilers are used. Note that we use the sequential, rather than parallel, version of the codes in the numerical experiments. Several benchmark problems with matrix sizes $46,875$ ($3 \times 25^3$), $375,000$ ($3 \times 100^3$), and $40,000,000$ ($3 \times 200^3$) are generated by using Yee’s scheme.
4.1. Results for Krylov-Schur method

We use KS to solve the generalized eigenvalue problem (13) with \( k = (\pi, \pi, 0) \). The KS solver is stopped if the residual of the eigenvalue system is less than \( 10^{-8} \). We set the target \( \sigma = 0.1 \) and use GMRES to solve the associated linear systems defined in (22) with residual stopping criterion equals \( 10^{-10} \). In each iteration of GMRES, we solve the preconditioned system only once.

Figure 3 shows overall CPU timing results and outer iteration numbers for different preconditioning schemes that are used in KS. The restarting Krylov subspace dimension “ncv” which is equal to \( j + p \) in Algorithm 1 are chosen from 20 to 100. Furthermore, Table 1 lists the average CPU timing for each of the preconditioning schemes. Table 2 shows the GMRES iteration numbers associated with various preconditioners. Table 3 shows the GMRES iteration numbers associated with PS3-FFT for the benchmark problems with different number of grid points. In Table 4, we show timing results of KS with PS3-FFT to solve the eigenvalue problems with versus \( k \) and grid resolutions. Sizes of the matrices are given in the captions of the tables and figures.

Observing Figure 3 and Table 1, we can see that PS3-FFT significantly outperforms other preconditioning schemes for all of the ncv choices. The reason why PS3-FFT is so efficient can be understood by observing how the iterative solver (i.e. GMRES in our numerical experiments) converges while solving (22). As shown in Table 2, the GMRES iteration number associated with PS3-FFT is much less than the ones associated with other preconditioners. Consequently, even the computational cost for solving Eq. (34) by PS3-FFT is more expensive than that for solving \( \tilde{M}(Dz) = \tilde{b} \) by the preconditioners in PS2. The overall performance due to PS3-FFT is remarkably better than the performance due to the PS2 preconditioning schemes. Figure 3 also suggests that CPU timing results change insignificantly as ncv varies in each of the preconditioning schemes. This is because the total numbers of linear systems (22) that have been solved (not shown in the figure) change insignificantly for different ncv’s. In contrast, the number of outer iterations (the repeat-loop in line 2 of Algorithm 1) gradually decreases as ncv increases.

The advantage for using PS3-FFT is not limited to the above case. We solve other benchmark problems with different grid numbers by using PS3-FFT. In Table 3, we report the iteration numbers taken by GMRES in KS with PS3-FFT for solving the linear system (22) arising in the benchmark problems with different sizes. It is surprising that the resulting iteration numbers are only between 27-30 for the problems sizes ranging from 46,875 \((3 \times 25^3)\) to 24,000,000 \((3 \times 200^3)\). Obviously, comparing the problem sizes, these iteration numbers are not only quite small, they are also almost constant. To see why the PS3-FFT preconditioners perform so well, we plot the spectrum and condition numbers of \((A - \tau B)\) and \((A - \tau \varepsilon_0 I)^{-1}(A - \tau B)\) in Figures 4 and 5, respectively. The matrices sizes are equal to 10,125 \((3 \times 15^3)\) and \( \tau = 0.01 \). Figure 4 shows that the eigenvalues of the original matrix distribute widely in the interval \([-0.01, 2,681] \). In contrast, the eigenvalues of the matrix \((A - \tau \varepsilon_0 I)^{-1}(A - \tau \varepsilon_0 I)^{-1}(A - \tau \varepsilon_0 I)^{-1}(A - \tau \varepsilon_0 I)^{-1}\)
τB) are clustered within [0.807, 10.51]. Furthermore, Figure 5 shows that the condition numbers can be reduced from around $2.685 \times 10^{5}$ to 21.3 due to the PS3-FFT preconditioners for the matrices associated with all the $k$ vectors.

We have confined our numerical experiments for $k = (\pi, \pi, 0)$ so far. To explore the overall performance of KS with PS3-FFT, we use the method to solve the eigenvalue problems with different $k$ and different grid resolutions. The results are shown in Table 4. Other than the average timing for grid resolutions, the table also shows the corresponding standard deviations, ratios of average, and ratios of the FFT complexity. Here, the FFT complexity is $O(n \log(n))$. We can see the method performs in a stable manner as the standard deviation is small relatively. Furthermore, the average computational timing agrees with the ratio of the complexity of FFT, which suggests the GMRES iteration numbers remains (almost) constant for various $k$ and grid resolutions.

In short, PS3-FFT outperforms other preconditioning schemes remarkably in KS. This preconditioner is so efficient that the resulting iteration numbers for solving (22) are small and almost constant. Furthermore, the overall computational time for solving the eigenvalue problems (13) are independent of the grid resolutions for various $k$ choices.

Besides, other numerical characteristics observed from the numerical experiments are listed as follows. (i) For PS1, if $M$ is taken as the Jacobi or ILU preconditioner, KS fails to converge to the 10 eigenpairs after four hours. For SSOR, the parameter $\omega$ must be around 0.8 to obtain convergent results. For ICC, we consider only the case that $\ell = 1$. The choices for $\ell > 1$ are not recommended as a great amount of fill-ins can be induced (see Figure 1 for an example of the sparsity of $A$). (ii) For PS2, we ignore results for other values of $\omega$ in SSOR and $\ell$ in ICC and ILU as the choice of $\omega = 0.8$ and $\ell = 1$ perform the best among others in our experiments. (iii) Although only the first 10 linear systems associated with the building an initial Lanczos decomposition in line 1 of Algorithm 1 are shown in Table 2 and 3. Similar behaviors can be found for other linear systems associated with (22).

4.2. Remarks about Krylov method with explicit and implicit starting

As Lanczos and Arnoldi are two popular methods for solving eigenvalue problems, we also test these two methods to evaluate their performance for solving (13). In particular, we use the Lanczos method (which is explicitly restarted) and the ARPACK wrapper (which is an implicitly restarting Arnoldi method) provided by SLEPc to solve the eigenvalue problem (13). We use PS3-FFT as the preconditioning scheme while solving the associated linear systems.

In our numerical experiments, the Lanczos method spends more than five hours computational time and not all 10 eigenpairs are obtained. For the Arnoldi method, most of the computed eigenvalues are larger than $\lambda_{10}$ as shown in Table 5. Consequently, both of the Lanczos and the Arnoldi method are not suitable for the target eigenvalue problems.
4.3. Results for Jacobi-Davidson method

To investigate the preconditioning performance in JD method, we solve the correction equations (26) by Bi-CGSTAB or GMRES (see Algorithms 3 and 4 in the appendix) with different preconditioners \( M \) in JD-S. For PS1-Jacobi, PS1-SSOR, PS1-ICC, and PS3-FFT preconditioners, we have tuned the parameters \( \omega, \ell, \) and \( m_i \) for \( i = 1, \ldots, 5 \), which are described in the heuristic Algorithms 3 and 4, to achieve better performance. The vector \( \mathbf{k} = (\pi, \pi, 0) \). The total JD-S iteration numbers (i.e., the sum of outer iteration numbers used for finding all the eigenpairs of interest) and CPU time in seconds for computing the ten target eigenpairs are shown in Table 6 and 7 for matrix dimension of \( A \) equals 46,875 \((3 \times 25^3)\) or 375,000 \((3 \times 50^3)\), respectively.

Observing these two tables, we can see the performance of PS3-FFT preconditioner is significantly better than other preconditioners, including PS1-Jacobi, PS1-SSOR, and PS1-ICC. Furthermore, the JD-S iteration numbers due to the PS3-FFT preconditioner changes little for different benchmark problems and the best choice of \( m_i \) for the PS3-FFT preconditioner is independent to the discretization resolution. In contrast, for other preconditioners, the JD-S needs much larger iterations to converge for larger problems. The best choice of \( m_i \) also varies for different problems. Similar to the case of KS, we can thus conclude that the PS3-FFT-solve of (34) outperforms other preconditioning schemes while JD-S is used as the eigenvalue solver.

As high-multiplicity zero eigenvalue exists in the target eigenvalue problem and the wanted eigenvalues are interior, we also examine the performance of JD-S and JD-H. In Figures 6 and 7, we show the convergence history of the Ritz values for the ten eigenvalues that are of interest by JD-S and JD-H, respectively. Figures 6 shows that the Ritz values are dragged towards zero during the iterations. In contrast, Figure 7 shows that the (harmonic) Ritz values tend to decrease monotonically to the convergent eigenvalues.

The parameters used in the numerical experiments are that \( \varepsilon_{\text{conv}} = 10^{-8}, \varepsilon_{\text{harm}} = 0.1, \varepsilon_{\text{shft}} = 10, \sigma = 10^{-2}, \text{ncv} = 25, \) and \( m = 4 \).

4.4. Comparison of Krylov-Schur and Jacobi-Davidson methods

We have seen that the performance of PS3-FFT preconditioner is remarkably better than other preconditioners for both KS and JD. Now, we make a comparison of the two eigenvalue solvers for different \( k \) vectors. Both KS and JD use PS3-FFT preconditioning. We consider the \( k \) vectors in the first Brillouin zone for the benchmark problem whose dimension is 375,000 \((3 \times 50^3)\). The ten smallest positive eigenvalues are calculated.

First, we solve the eigenvalue problems associated with the \( k \)'s along the segments connecting \( \Gamma = (0,0,0), X = (\pi,0,0), M = (\pi,\pi,0), R = (\pi,\pi,\pi) \) and \( \Gamma \). In each of the segments, four uniform distributed sampling vectors \( k \) are chosen. In KS, all the target \( \sigma \) are set to be 0.1 and the \( \text{ncv} \) is taken as 25, 30, 35 and 40. Figure 8 illustrates the timing results and suggest the following. KS with PS3-FFT preconditioner outperforms JD-S and JD-H with PS3-FFT preconditioners for all \( k \)'s (except one case). JD-H is considerably quicker than
JD-S by taking about one-half of CPU time than that of JD-S. In addition, KS has smaller variation in timing for different $k$ and different $ncv$; in contrast, timing results of JD-S and JD-H vary significantly for different $k$.

The behavior of JD-S can be understood as follows. As the number of zero eigenvalues of the Hermitian generalized eigenvalue problem (13) is $n \ [24]$, where $3n$ is the matrix size, the Ritz value $\theta_k$ is constantly dragged towards zero during the subspace iterations due to the large null space. Consequently, the JD-S needs more iterations to react against the zero eigenvalue dragging and to converge to positive eigenvalues. In contrast, the zero eigenvalues affect JD-H less and JD-H performs 1.4 to 2.3 times faster than JD-S in these testing cases.

Second, we investigate the performance of different solvers for the $k$’s near $\Gamma = (0, 0, 0)$. In particular, we choose the sampling vectors $(2 \pi \kappa, 0, 0)$ and $(2 \pi \kappa, 2 \pi \kappa, 2 \pi \kappa)$ for $\kappa = 0.001, 0.005, 0.010, 0.025, 0.050, 0.075, \text{and } 0.100$. In KS, we choose $ncv = 25$. The overall timing for finding ten eigenvalues for each of the $k$’s are shown in Figure 9. Unlike the cases shown in Figure 8, JD-H is the most efficient solver for almost all the tested cases and KS is slower than other solvers as $k$ approaching zero.

We examine why KS is slow for small $\kappa$’s and provide a simple remedy. As there are two small eigenvalues when the $k$’s near $\Gamma = (0, 0, 0) \ [23]$ (actually there are $n + 2$ zero eigenvalues when $k = \Gamma$), KS uses small targets so that it can find these two clustered eigenvalues in a reasonable time. In particular, KS sets $\sigma = 10^{-5}, 10^{-4}, \text{and } 10^{-3}$ for $\kappa = 0.001, 0.005, \text{and } 0.01$, respectively. However, such small $\sigma$’s lead to the side effects that the resulting coefficient matrices $(A - \sigma B)$ are close to singular, as $A$ is singular. KS thus takes more iterations and computational time for the other eight eigenvalues. For example, Figure 9 shows that KS is about three- and five-times slower than JD-S and JD-H for $\kappa = 0.001$, respectively. Besides, such ill-conditioned systems leads to residuals around $10^{-4}$ to $10^{0}$ for these eight eigenpairs when $\kappa = 0.001, 0.005, \text{and } 0.01$. A simple modification on choosing the target $\sigma$ can improve the performance of KS. We use the same smaller $\sigma$’s in KS for computing the first two eigenvalues. We then switch to a larger $\sigma$ for computing another eight eigenvalues. We set $\sigma = 0.1$ in our numerical experiments. Performance of such “two-stage” method is shown in Figure 9 by labeling as “KS-2”. The figure shows that KS-2 successfully improves the timing performance of KS. Nevertheless, JD-H performs the best for almost all the tested cases with small $\kappa$’s.

5. Conclusion

Aiming to effectively solve the eigenvalue problems arising in the numerical simulations of three-dimensional photonic crystals, we have studied how various preconditioning schemes can accelerate the computations. Based on Yee’s scheme, Maxwell’s equations are discretized to lead to the eigenvalue problems. We consider mainly the Krylov-Schur method and Jacobi-Davidson method as the eigenvalue solvers. After deriving the FFT-based preconditioner in detail, we have conducted intensive numerical experiments to evaluate the performance
of the preconditioners including Jacobi, SSOR, ILU, ICC, FFT-based for solving the corresponding linear systems arising in the eigenvalue solvers. We have found that the Krylov-Schur method with the FFT-based preconditioner is extraordinary efficient and outperforms than other combinations of eigenvalue solvers and preconditioners in almost all cases.

While we focus on finding efficient preconditioning schemes in this article, further timing performance improvement can be achieved via parallelization. In addition to the parallelizations of basic linear algebra and book-keeping operations, we anticipate that the main time saving comes from parallelization of the FFT-based preconditioned linear systems. Not only solving these linear systems is the most expensive part of both Krylov-Schur and Jacobi-Davidson methods, FFT itself is highly parallelizable by nature. To parallelize the proposed solvers on a Message Passing Interface (MPI) based computer cluster, we need to develop a parallel FFT-based preconditioner and then integrate it with parallel versions of KS, KS-2, and JD-H solvers. Obviously, the parallelizations of these solvers can be efficiently developed by taking advantage of the parallel computing capabilities of PETSc and SLEPc. In addition, growing usages of Graphics Processing Units (GPU) and hybrid CPU-GPU in scientific computing have suggested another potential computer architectures to accelerate the proposed approaches via parallelization. The key components include sparse matrix-vector multiplications and FFT, which are anticipated to be efficient on these configurations.

Appendix

To solve the linear systems involving the solution of the correction equation (26), we propose using the heuristic strategies presented in Algorithms 3 (for the first eigenvalue) and Algorithms 4 (for other eigenvalues) to achieve efficiency. Note that a simpler version of such heuristic is proposed in [25]. Several tunable parameters can be chosen in the heuristics, including the linear system solvers (GMRES [39] or Bi-CGSTAB [47]), preconditioning strategy, maximum iteration number, and residual stopping criterion. For example, \{GMRES, M, 30, 10^{-4}\} indicates that GMRES with the preconditioner \(M\) is used to solve a linear system, the maximum iteration number is 30, and the residual stopping criterion is \(10^{-4}\). To compute the first smallest positive eigenvalue (\(\lambda_1\)), Line 22 of Algorithm 2 is changed to the pseudo-code illustrated in Algorithm 3. In addition, to compute other positive eigenvalues (\(\lambda_2, \lambda_3, \ldots\)), Line 22 of Algorithm 2 is changed to the pseudo-code illustrated in Algorithm 4. Note that the variable \(i\) is set to be equal to 15 initially.

Acknowledgments

The authors are grateful to the anonymous referees for their useful comments and suggestion. This work is partially supported by the National Science Council, the Taida Institute of Mathematical Sciences, and the National Center for Theoretical Sciences in Taiwan.
Algorithm 3 The heuristic strategy for computing $\lambda_1$.
1: Solve Eq. (26) (approximately) to obtain a $t_k \perp_B u_k$ by the method determined below.
2: if ($k \leq 9$) then
3: Use $\{\text{BiCGSTAB, No precond., } 6, 10^{-4}\}$.
4: else
5: Use $\{\text{GMRES, } \mathcal{M}, m_1, 10^{-4}\}$.
6: end if

Algorithm 4 The heuristic strategy for computing $\{\lambda_2, \ldots, \lambda_\ell\}$.
1: Solve Eq. (26) (approximately) to obtain a $t_k \perp_B u_k$ by the method determined below.
2: if $k = 1$ then
3: Set flag = 1.
4: end if
5: if ($\|r_k\|_2 > 0.1$ and $k < 6$) then
6: Use $\{\text{BiCGSTAB, No precond., } 6, 10^{-4}\}$.
7: else if ($\|r_k\|_2 \geq 0.1$ and $k > 14$) then
8: Use $\{\text{GMRES, } \mathcal{M}, m_2, 10^{-4}\}$.
9: Set flag = 1.
10: else if (flag == 1) then
11: Set $i = m_3$ and flag = 0.
12: Use $\{\text{GMRES, } \mathcal{M}, i, 10^{-4}\}$.
13: else if ($\|r_k\|_2 < 0.1$ and $\|r_{k-1}\|_2/\|r_k\|_2 < 4$) then
14: Set $i = \min(m_4, i + m_5)$.
15: Use $\{\text{GMRES, } \mathcal{M}, i, 10^{-4}\}$.
16: else
17: Use $\{\text{GMRES, } \mathcal{M}, i, 10^{-4}\}$.
18: end if
References


Figure 1: A sparsity example of $A$ that is defined in (13).

Figure 2: The periodic dielectric structure within a primitive cell. Inside of the structure is the dielectric material and outside of the structure is air. Here $r/a = 0.345$, $s/a = 0.11$, and $\varepsilon_i/\varepsilon_o = 13.$
Figure 3: Overall CPU timing results and number of outer iterations of KS with various preconditioners and different restarting Krylov subspace dimension ncv. The sizes of the matrices are 46,875 (3 × 253).
<table>
<thead>
<tr>
<th>Preconditioning scheme</th>
<th>Average CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS1-SSOR(0.8)</td>
<td>6,558</td>
</tr>
<tr>
<td>PS1-ICC(1)</td>
<td>10,606</td>
</tr>
<tr>
<td>PS2-Jacobi</td>
<td>2,947</td>
</tr>
<tr>
<td>PS2-SSOR(0.8)</td>
<td>1,511</td>
</tr>
<tr>
<td>PS2-ICC(1)</td>
<td>1,847</td>
</tr>
<tr>
<td>PS2-ILU(1)</td>
<td>2,778</td>
</tr>
<tr>
<td><strong>PS3-FFT</strong></td>
<td><strong>75</strong></td>
</tr>
</tbody>
</table>

Table 1: Average CPU timing of KS associated with the preconditioning schemes and the ncv’s used in Figure 3. The sizes of the matrices are $46,875$ ($3 \times 25^3$).

<table>
<thead>
<tr>
<th>Index $k$</th>
<th>PS2-Jacobi</th>
<th>PS2-SSOR(0.8)</th>
<th>PS2-ICC(1)</th>
<th>PS2-ILU(1)</th>
<th>PS3-FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>852</td>
<td>493</td>
<td>296</td>
<td>273</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>853</td>
<td>492</td>
<td>296</td>
<td>273</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>1,008</td>
<td>462</td>
<td>287</td>
<td>284</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>956</td>
<td>510</td>
<td>279</td>
<td>282</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>1,026</td>
<td>507</td>
<td>300</td>
<td>310</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>954</td>
<td>497</td>
<td>289</td>
<td>297</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>949</td>
<td>499</td>
<td>289</td>
<td>297</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>1,156</td>
<td>546</td>
<td>298</td>
<td>303</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>891</td>
<td>504</td>
<td>298</td>
<td>300</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>1,086</td>
<td>516</td>
<td>307</td>
<td>303</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 2: The iteration numbers taken by GMRES in KS for solving the linear system (22) with different preconditioners. We present the results for solving the linear systems associated with constructing Krylov subspace $V_k$, for $k = 1, ..., 10$, in line 1 of Algorithm 1. The problem dimension is $3 \times 25^3$.

<table>
<thead>
<tr>
<th>Index $k$</th>
<th>$3 \times 25^3$</th>
<th>$3 \times 50^3$</th>
<th>$3 \times 75^3$</th>
<th>$3 \times 100^3$</th>
<th>$3 \times 200^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td>27</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>27</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>27</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>28</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>28</td>
<td>28</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>29</td>
<td>28</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>28</td>
<td>28</td>
<td>27</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>28</td>
<td>29</td>
<td>28</td>
<td>28</td>
<td>29</td>
</tr>
</tbody>
</table>

Table 3: The iteration numbers taken by GMRES in KS with PS3-FFT for solving the linear system (22) arising in the benchmark problems with different sizes. We present the results for solving the linear systems associated with constructing Krylov subspace $V_k$, for $k = 1, ..., 10$, in line 1 of Algorithm 1.
<table>
<thead>
<tr>
<th>k</th>
<th>Benchmark problem dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$3 \times 25^3$</td>
</tr>
<tr>
<td>$(\pi/4, 0, 0)$</td>
<td>65.0</td>
</tr>
<tr>
<td>$(\pi/2, 0, 0)$</td>
<td>70.8</td>
</tr>
<tr>
<td>$(3\pi/4, 0, 0)$</td>
<td>81.8</td>
</tr>
<tr>
<td>$(\pi, 0, 0)$</td>
<td>56.3</td>
</tr>
<tr>
<td>$(\pi, \pi/4, 0)$</td>
<td>61.6</td>
</tr>
<tr>
<td>$(\pi, \pi/2, 0)$</td>
<td>61.4</td>
</tr>
<tr>
<td>$(\pi, 3\pi/4, 0)$</td>
<td>48.6</td>
</tr>
<tr>
<td>$(\pi, \pi, 0)$</td>
<td>75.9</td>
</tr>
<tr>
<td>$(\pi, \pi, \pi/4)$</td>
<td>70.7</td>
</tr>
<tr>
<td>$(\pi, \pi, \pi/2)$</td>
<td>57.8</td>
</tr>
<tr>
<td>$(\pi, \pi, 3\pi/4)$</td>
<td>51.1</td>
</tr>
<tr>
<td>$(\pi, \pi, \pi)$</td>
<td>67.4</td>
</tr>
<tr>
<td>$(3\pi/4, 3\pi/4, 3\pi/4)$</td>
<td>69.2</td>
</tr>
<tr>
<td>$(\pi/2, \pi/2, \pi/2)$</td>
<td>67.9</td>
</tr>
<tr>
<td>$(\pi/4, \pi/4, \pi/4)$</td>
<td>72.9</td>
</tr>
<tr>
<td>Average timing</td>
<td>65.2</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>9.2</td>
</tr>
<tr>
<td>Ratio of average timing</td>
<td>-</td>
</tr>
<tr>
<td>Ratio of FFT complexity</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4: The CPU time (in seconds) used by KS with PS3-FFT to solve the eigenvalue problems with various $k$ and grid resolutions. Here we take $ncv=20$. The complexity of FFT $n \log(n)$. 

ncv iter CPU Time (s) nconv nwant nzero
20 9 525.3 2 0 2
25 7 609.7 8 4 0
30 3 464.8 7 5 0
35 4 688.7 10 4 0
40 3 656.4 10 4 0
45 3 765.7 8 2 0
50 2 636.3 8 4 0
55 2 708.9 8 2 0
60 2 785.9 9 2 0
65 2 866.1 9 2 0
70 2 945.7 9 2 0
75 1 587.5 7 3 0
80 1 627.0 8 4 0
85 1 666.9 9 3 0

Table 5: Numerical results for the implicitly restarting Arnoldi method with PS3-FFT preconditioner. In the table, \( \text{iter} \) denotes the number of restarting Krylov decomposition, \( \text{nconv} \) denotes the number of convergent eigenvalues obtained. Among these \( \text{nconv} \) eigenvalues, \( \text{nwant} \) denotes the numbers of wanted eigenvalues \( \{\lambda_1, \ldots, \lambda_{10}\} \) that have been found and \( \text{nzero} \) denotes the number of zero eigenvalues.

<table>
<thead>
<tr>
<th>M</th>
<th>( m_1 )</th>
<th>( m_2 )</th>
<th>( m_3 )</th>
<th>( m_4 )</th>
<th>( m_5 )</th>
<th>Iter</th>
<th>CPU Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS1-Jacobi</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>150</td>
<td>50</td>
<td>555</td>
<td>1,006</td>
</tr>
<tr>
<td>PS1-SSOR(0.75)</td>
<td>50</td>
<td>55</td>
<td>75</td>
<td>65</td>
<td>10</td>
<td>671</td>
<td>935</td>
</tr>
<tr>
<td>PS1-SSOR(0.9)</td>
<td>50</td>
<td>55</td>
<td>75</td>
<td>65</td>
<td>10</td>
<td>643</td>
<td>940</td>
</tr>
<tr>
<td>PS1-ICC(1)</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>50</td>
<td>10</td>
<td>904</td>
<td>1,757</td>
</tr>
<tr>
<td>PS1-ICC(2)</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>50</td>
<td>10</td>
<td>1,093</td>
<td>3,593</td>
</tr>
<tr>
<td>PS1-ICC(3)</td>
<td>100</td>
<td>150</td>
<td>100</td>
<td>100</td>
<td>10</td>
<td>1,709</td>
<td>7,087</td>
</tr>
<tr>
<td>PS3-FFT</td>
<td>25</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>2</td>
<td>236</td>
<td>104</td>
</tr>
</tbody>
</table>

Table 6: Numerical results of JD-S with different preconditioner. The total JD-S iteration numbers and CPU time in seconds for computing the ten target eigenpairs are shown. The matrix size is equal to 46,875 (\( 3 \times 25^3 \)).

<table>
<thead>
<tr>
<th>M</th>
<th>( m_1 )</th>
<th>( m_2 )</th>
<th>( m_3 )</th>
<th>( m_4 )</th>
<th>( m_5 )</th>
<th>Iter</th>
<th>CPU Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS1-Jacobi</td>
<td>150</td>
<td>150</td>
<td>250</td>
<td>150</td>
<td>100</td>
<td>1,105</td>
<td>32,429</td>
</tr>
<tr>
<td>PS1-SSOR(0.8)</td>
<td>50</td>
<td>55</td>
<td>75</td>
<td>60</td>
<td>2</td>
<td>1,278</td>
<td>20,647</td>
</tr>
<tr>
<td>PS1-SSOR(1.0)</td>
<td>45</td>
<td>45</td>
<td>65</td>
<td>55</td>
<td>2</td>
<td>1,418</td>
<td>20,262</td>
</tr>
<tr>
<td>PS1-ICC(0)</td>
<td>350</td>
<td>350</td>
<td>450</td>
<td>300</td>
<td>100</td>
<td>1,200</td>
<td>94,347</td>
</tr>
<tr>
<td>PS1-ICC(1)</td>
<td>100</td>
<td>100</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>1,066</td>
<td>38,227</td>
</tr>
<tr>
<td>PS3-FFT</td>
<td>25</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>2</td>
<td>243</td>
<td>1,187</td>
</tr>
</tbody>
</table>

Table 7: Numerical results of JD-S with different preconditioners. The total JD-S iteration numbers and CPU time in seconds for computing the ten target eigenpairs are shown. The matrix size is equal to 375,000 (\( 3 \times 50^3 \)).
Figure 4: Spectrum of (a) the matrix \((A - \tau B)\) and (b) the matrix \((A - \tau \epsilon_0 I)^{-1}(A - \tau B)\). The matrices sizes are equal to 10,125 \((3 \times 15^3)\), \(k = (\pi, \pi, 0)\), and \(\tau = 0.01\).

Figure 5: Condition numbers of (a) the matrix \((A - \tau B)\) and (b) the matrix \((A - \tau \epsilon_0 I)^{-1}(A - \tau B)\). The matrices sizes are equal to 10,125 \((3 \times 15^3)\) and \(\tau = 0.01\).
Figure 6: Convergence history of the Ritz values for $\lambda_1, \ldots, \lambda_{10}$ by using JD-S.

Figure 7: Convergence history of the harmonic Ritz values for $\lambda_1, \ldots, \lambda_{10}$ by using JD-H.
Figure 8: CPU time in seconds of JD-S, JD-H, and KS with PS3-FFT preconditioner versus $k$. The dimension of the test problem is $375,000 \times 50^3$.

Figure 9: CPU time of JD-S, JD-H, KS, and KS-2 with PS3-FFT preconditioner versus $k$ that are close to $\Gamma = (0, 0, 0)$. The dimension of the test problem is $375,000 \times 50^3$. 