Report

Finite Difference Mode Solver

Steev Mathew, Felix Wechsler, Yaqi Zhang

Group 3

May 27, 2020

Abbe School of Photonics

Friedrich-Schiller-Universität Jena

Contents

1 Introduction **3**

2 Physical Background **3**

3 Numerical Implementation **4**

3.1 Background 4

3.2 Implementation 6

4 Results **7**

4.1 Speed 9

5 Conclusion **10**

1 Introduction

In this report we explain how we can create a simple finite difference mode solver in `Python`. The physical problem we solve as an example, is the calculation of guided modes in 2D and 3D media. For simplification we assume low gradients in the dielectric permittivity and hence we solve the stationary scalar Helmholtz equation.

2 Physical Background

In this section we introduce the main physical concept. A more extensive discussion can be found in [PRK20]. Starting from Maxwell's equation, we can derive the general wave equation:

$$\nabla \times \nabla \times \mathbf{E}(\mathbf{r}, \omega) = \frac{\omega^2}{c^2} \varepsilon(\mathbf{r}, \omega) \mathbf{E}(\mathbf{r}, \omega) \tag{1}$$

Assuming that the change in ε is rather small, we can write:

$$\nabla \cdot \mathbf{D}(\mathbf{r}, \omega) = 0 \Rightarrow \varepsilon(\mathbf{r}, \omega) \nabla \cdot \mathbf{E}(\mathbf{r}, \omega) \approx 0 \tag{2}$$

This simplifies the wave equation to the Helmholtz equation:

$$\triangle \, \mathbf{E}(\mathbf{r}, \omega) + \frac{\omega^2}{c^2} \varepsilon(\mathbf{r}, \omega) \mathbf{E}(\mathbf{r}, \omega) = 0 \tag{3}$$

The general 3D ansatz functions for a wave propagation in z-direction for the Helmholtz equation are plane waves in the form of:

$$\mathbf{E}(\mathbf{r}, \omega) = \mathbf{E}_0(x, y) \exp\left[i\beta(\omega) z - i\omega t\right] \tag{4}$$

Inserting Equation 4 into Equation 3 provides the final result we need for the numerical calculation of the eigenmodes:

$$\frac{1}{k_0^2} \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) E_0(x, y, \omega) + \varepsilon(x, y, \omega) E_0(x, y, \omega) = \varepsilon_{\text{eff}}(\omega) E_0(x, \omega) \tag{5}$$

where $\varepsilon_{\text{eff}} = \left(\frac{\beta(\omega)}{k_0} \right)^2$ and $k_0 = \frac{2\pi}{\lambda}$ and λ being the wavelength of the electric field. On the left hand side of Equation 5 we see a second spatial derivative regarding x and y. The time dependency can be left out, since we are looking for stationary modes and the time dependency is just trivial.

3 Numerical Implementation

First we show the numerical background and afterwards we explain some details of our implementation.

3.1 Background

We directly explain the numerical discretization in 2D since 1D is only a special case of it. For simplicity we split the continuous space into a 2D grid with size $N_y \times N_x$. In Figure 1 we see such a discretization.

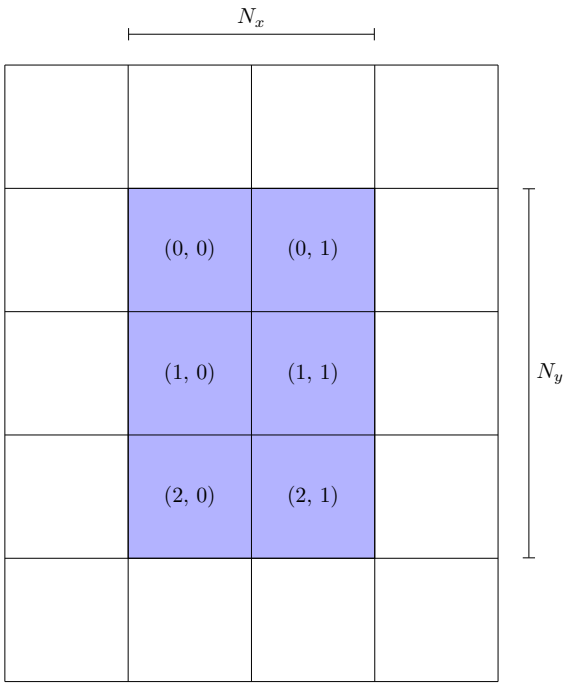

Figure 1: Discretization of the space

The function $f(x, y)$ is now represented by samples $f_{i,j}$. The central finite difference to calculate the second derivative is given by the following equation:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\bigg|_{x_i, y_j} \approx \frac{f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - 4f_{i,j}}{h^2} \tag{6}$$

We assume that the boundary of the function f is zero. The white cells are 0 and the blue cells are non restricted. This simplifies the second derivative for blue cells which have neighbouring white cells. Following Equation 6 we can express this equation

via a matrix vector multiplication:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \approx \frac{1}{h^2} \begin{bmatrix} -4 & 1 & 1 & & & \\ 1 & -4 & 0 & 1 & & \\ 1 & 0 & -4 & 1 & 1 & \\ & 1 & 1 & -4 & 0 & 1 \\ & & 1 & 0 & -4 & 1 \\ & & & 1 & 1 & -4 \end{bmatrix} \cdot \begin{bmatrix} f_{0,0} \\ f_{0,1} \\ f_{1,0} \\ f_{1,1} \\ f_{2,0} \\ f_{2,1} \end{bmatrix} \tag{7}$$

h must be set to the discretization sampling. This scheme can be extended to a higher number of grid points. There are certain rules this matrix obeys. The diagonal elements are always of -4. The off diagonals $x_{m+1,m}$ with $m \in \{0, 1, \ldots, N_x + N_y - 2\}$ and $x_{n,n+1}$ with $n \in \{0, 1, \ldots, N_x + N_y - 2\}$ are always 1 expect when $(m+1) \equiv 0 \bmod N_x$ and $(n+1) \equiv 0 \bmod N_x$. This handles the case of the line break of the 2D grid. The vector consists of the x-rows of the grid. They have an offset of N_x to the main diagonal. This is again due to the line break. The elements are always 1. Having expressed the Laplacian as a matrix we can write down Equation 5 in this form:

$$\begin{bmatrix} -\frac{4}{k_0^2h^2} + \varepsilon_{0,0} & \frac{1}{k_0^2h^2} & \frac{1}{k_0^2h^2} & & & \\ \frac{1}{k_0^2h^2} & -\frac{4}{k_0^2h^2} + \varepsilon_{0,1} & 0 & \frac{1}{k_0^2h^2} & & \\ \frac{1}{k_0^2h^2} & 0 & -\frac{4}{k_0^2h^2} + \varepsilon_{1,0} & \frac{1}{k_0^2h^2} & \frac{1}{k_0^2h^2} & \\ & \frac{1}{k_0^2h^2} & \frac{1}{k_0^2h^2} & -\frac{4}{k_0^2h^2} + \varepsilon_{1,1} & 0 & \frac{1}{k_0^2h^2} \\ & & \frac{1}{k_0^2h^2} & 0 & -\frac{4}{k_0^2h^2} + \varepsilon_{2,1} & \frac{1}{k_0^2h^2} \\ & & & \frac{1}{k_0^2h^2} & \frac{1}{k_0^2h^2} & -\frac{4}{k_0^2h^2} + \varepsilon_{2,2} \end{bmatrix} \cdot \begin{bmatrix} E_{0,0} \\ E_{0,1} \\ E_{1,0} \\ E_{1,1} \\ E_{2,0} \\ E_{2,1} \end{bmatrix} = \varepsilon_{\text{eff}} \begin{bmatrix} E_{0,0} \\ E_{0,1} \\ E_{1,0} \\ E_{1,1} \\ E_{2,0} \\ E_{2,1} \end{bmatrix}.$$

To solve for the vector $E_{i,j}$ we need to solve an eigenvalue problem.

3.2 Implementation

In this part we explain important steps of our source code. The full code can be found in the attachment. Important to note is, that we use the function `diags` from the `scipy.sparse` package [Vir+20]. This allows us to create a sparse matrix with the diagonals stored in `data` and with offsets `[0, 1, −,1 −Nx, Nx]`. A dense matrix would not be possible to store for 3D problems. One important step is the creation of a mask in line 135. This is needed, as we have seen before that the off diagonals have zeros at certain positions. This is achieved by that mask. In line 137 we set the off diagonals to 0 where the mask is true.
Having that matrix finally created with line 144, we can call the function `get_eigen_matrix`. This function essentially calls `eigsh` from `scipy.sparse` to calculate the eigenvectors and eigenvalues of our matrix. Furthermore we take the real part of both the eigenvectors and eigenvalues. This is reasonable assuming that our material is lossless. Another issue arising from `eigsh` is, that the eigenvalues are not ordered. Therefore we perform an additional sorting of the eigenvectors and eigenvalues. The reason why we use `eigsh` instead of `eigs` is speed. The latter turns out to be slower on our type of problem. Since the matrix is symmetric, we can use the symmetric version `eigsh`.

4 Results

In this section we present the obtained results. In Figure 2 we can see the numerical results for the 2D slab material. We only show five solutions with the highest effective ε_{eff}. It can be seen that the ground mode is associated with the highest effective permittivity. The normalization is chosen in such a way, that the peaks are 1.

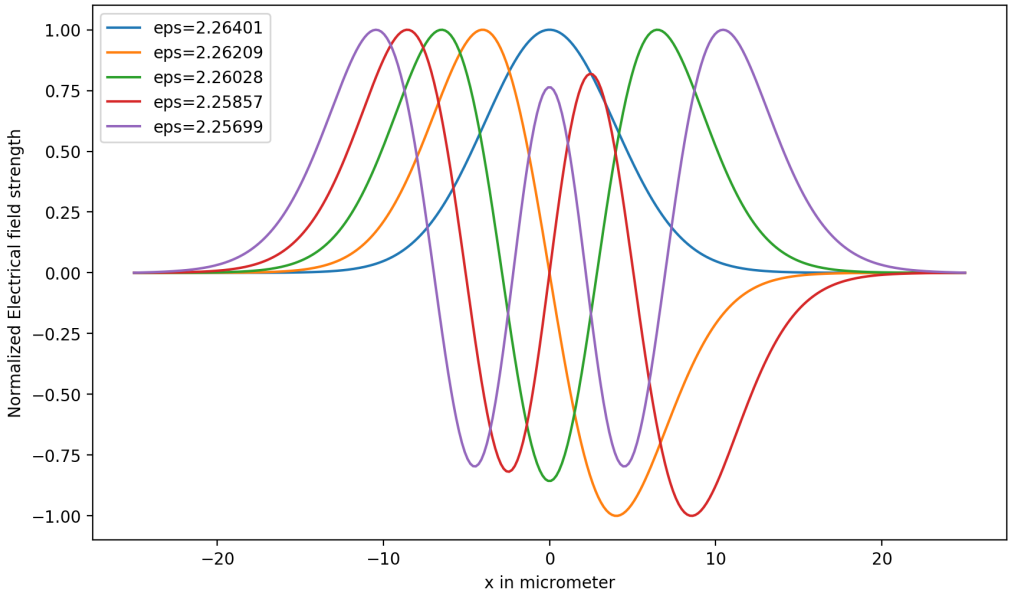

Figure 2: The first 5 modes in a slab waveguide.

Important for a correct numerical solution is, that the grid is large enough that the modes are already close to 0 at the boundary. A grid of $[-10\,\mu\text{m}, 10\,\mu\text{m}]$ for example would not be broad enough to ensure that the field is 0. Additionally, as it can be seen, higher order modes are spatially more extended.
In Figure 3 and Figure 4 we can see the 2D cross section of the normalized electrical field for the 3D waveguide. The 0th order is symmetric around the center. The 2nd order is not.

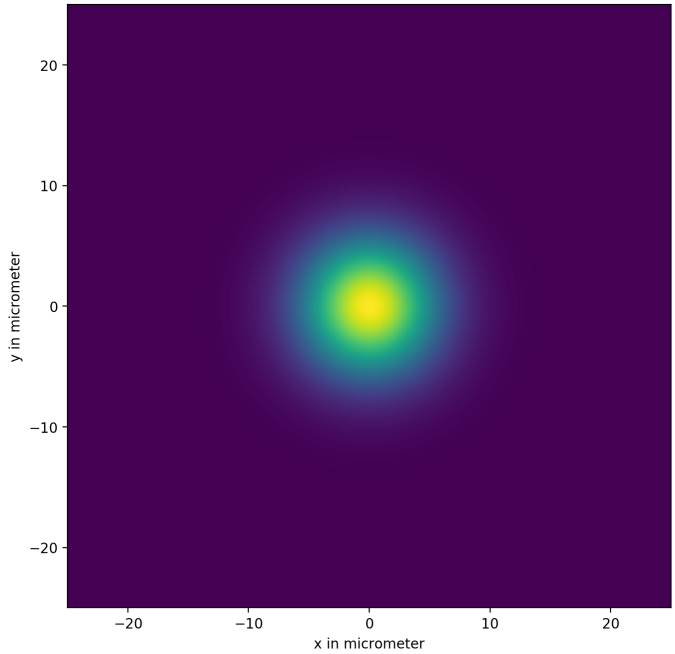

Figure 3: The first mode (0th order) in a strip waveguide.

We notice that this 2nd order mode is also rotated slightly and is not aligned with the symmetry of the grid. This is reasonable since the modes are not oriented in a specific way. Also **`eigsh`** returns different orientations if it is called multiple times.

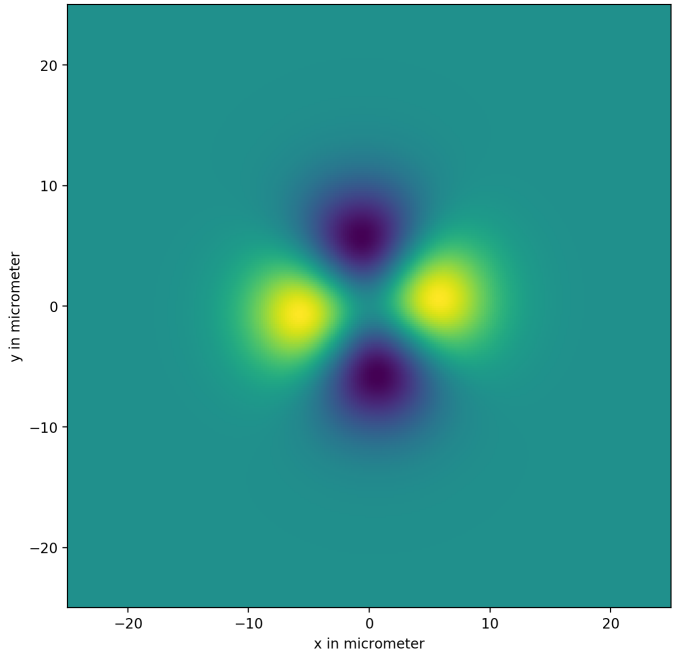

Figure 4: A higher order mode (2nd order) in a strip waveguide.

Our source code is also designed in a way to calculate non-rectangular grids. This is useful if one wants to solve systems which require more data points in a certain direction. An example is shown in Figure 5.

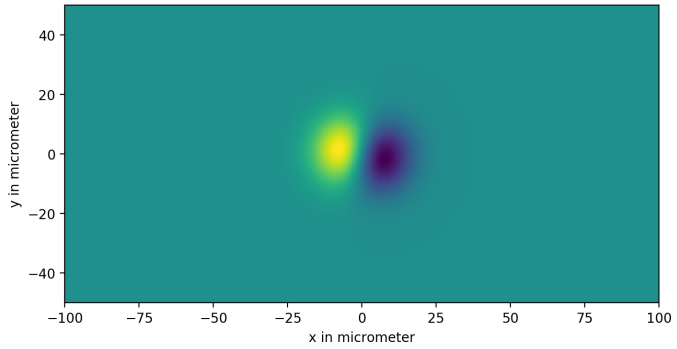

Figure 5: A higher order mode (1st order) in a strip waveguide on a non-rectangular grid.

4.1 Speed

We also measured the time needed to calculate the eigenmodes. Our setup for all measurements was Python 3.8.2, SciPy 1.3.2, NumPy 1.17.3 and a Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz. The full results can be found in Table 1. Whether using single or double precision did make a significant difference in the slab waveguide. Unexpected was, that single precision was actually slower. For a

	Time needed for `eigsh` in s	Total time needed in s
2D, single precision	0.2 ± 0.04	0.21 ± 0.04
2D, double precision	0.13 ± 0.02	0.14 ± 0.02
3D, single precision	2.64 ± 0.16	2.65 ± 0.16
3D, double precision	13.43 ± 0.98	13.44 ± 0.97

Table 1: Results for the computing time

3D material, the time difference between single and double precision was significant. In both cases the numerical results were up to machine precision identical. The exact chosen parameters for both runs can be found in the attached Jupyter Notebook. Another characteristic of the algorithm is, that it takes more time to compute more eigenmodes. To calculate the first mode, it is therefore usually enough to calculate one eigenvalue. This reduces the computational time further. In Table 1 we list

also the time needed in the function `eigsh`. This function call needs most of the computational time, which is also expected since it does all the solving part. This indicates that our source code performs well and is not the bottleneck.

5 Conclusion

In this report we presented a quick introduction into finite difference mode solver. Attached to this report the full source code can be found. Using the code, we were able to reproduce different guided modes. We have to show that single precision results are beneficial to use in the 3D case and give a significant speedup without loss of physical relevant accuracy.

References

[PRK20] T. Pertsch, C. Rockstuhl, and T. Kaiser. *Finite difference to solve guided eigenmodes in scalar approximation*. Lecture Notes on Computational Photonics. 2020.

[Vir+20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. 1. 0. Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2.