Workshop #2: Introduction to finite differencing

AP 217: Foundations of Modern Optics
Harvard University

January 15, 2024

Contents

1 Introduction

1.1 Motivation

Much of the lecture component of this course is dedicated to Maxwell's Equations: Their formal setup, their structure, and their analytical solution in a select few basic situations of optics. Since the "discovery" of Maxwell's Equations in the 1870s, much of optics has involved providing analytical expressions for their solution in ever more complex scenarios. These often herald progress in physics more generally — the plane wave solution established the nature of light itself, while the simple introduction of an interface between two media gave a rigorous basis to the empirically-observed Snell's Law. As a more complex example, the fully-analytical Mie solution to Maxwell's Equations (1908) is widely used to this day to analyze scattering from spherical particles.

Analytics is crucially important to both this course and electromagnetism as a whole. However, a change in the problem usually demands a new theory. In the realm of optical design, or any situation in which we would like to make some practical device or system using our understanding of optics, this is often unacceptable. As a designer, we would like to have complete freedom to vary geometry and material parameters as a part of our workflow, without having to begin anew from first-principles.

What we desire is a general-purpose method of solving Maxwell's Equations that is 1) Repeatable and simple to implement, and 2) Applicable to a wide variety of situations without significant modification.

This is why we want to develop numerical methods to solve Maxwell's Equations.

1.2 What do we mean by numerical solutions of Maxwell's Equations?

For the purpose of this course, numerically solving Maxwell's Equations will mean casting electromagnetic problems as matrix equations. This will come in two flavors:

1. Problems of the form
$$A\vec{x} = \vec{b}, \tag{1}$$
i.e., linear systems of equations.

2. Problems of the form
$$A\vec{x} = \lambda\vec{x} \tag{2}$$
i.e., eigenvalue problems with λ as an eigenvalue.

(capital letters, both Latin and Greek, in these notes represent matrix quantities)

In these workshops, we will manipulate Maxwell's Equations into this form. The operator $\boldsymbol{A}$ will contain the governing equations and the geometry of the problem, as well as its boundary conditions. The solution vector $\vec{x}$ contains the fields that solve Maxwell's Equations. When applicable, the vector $\vec{b}$ contains sources of electromagnetic radiation that excite the system under consideration.

In both cases, extremely well-developed numerical algorithms exist. For the former, techniques such as LU decomposition (i.e., row reduction) and QR decomposition are built-in methods of most major programming languages[1]. For the latter, techniques such as the shifted-inverse power method are often used. Beyond their existence, the details of these techniques will not concern us much here. Rest assured these tools are readily available and highly efficient, particularly so if the matrices involved are sparse. For a sampling of such methods, see the SciPy linear algebra package[2].

2 Finite differencing

Finite differencing refers to a casting of a differential equation on a discrete coordinate grid. The finite difference method numerically solves differential equations on these discrete grids.

Note that in these workshops we are, in some ways, taking a whirlwind tour of this subject area. Finite difference methods are a deep mathematical subject, including more subtle mathematical details around convergence and

[1] Notably, under no circumstance is the direct inverse of the matrix A^{-1}ever directly calculated.

[2] There are two such packages, one general-purpose (click here) and one for sparse matrices specifically (click here).

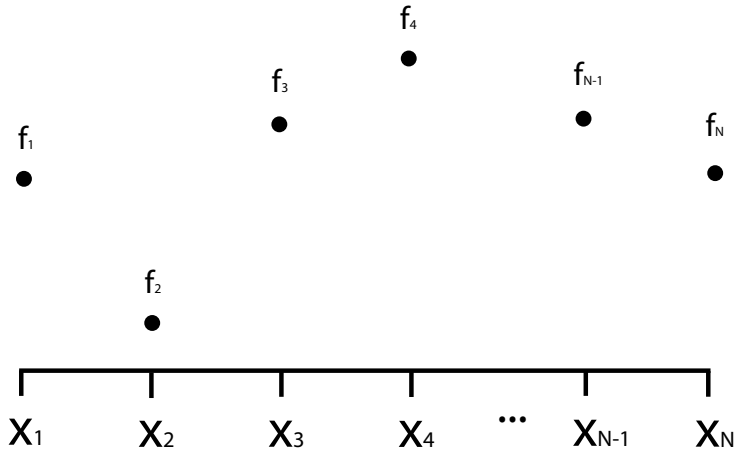

Figure 1: Discretizing a function $f(x)$ into sampled form as the vector $\vec{f}$.

stability that may at times be glossed over here. The overarching goal here is to lay the groundwork to implement and run your own optical simulations, and to understand what's "under-the-hood" of many commercially-available tools.

2.1 Derivatives on discrete coordinate grids

On a discrete coordinate grid, a function $f(x)$ can be represented by the values f_i it takes on at a discrete set of points $x \in \{x_i\}$. The function, then, can be represented in vector form as

$$\vec{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}. \tag{3}$$

The $\{x_i\}$ need not be evenly spaced but for the sake of this course, they will be. Suppose $x_i - x_{i-1} = \Delta x$. What about the derivative of $f(x)$, $\frac{df(x)}{dx}$? Suppose we are interested in its value at $x = x_i$. There is no one unique way that we can write down $\frac{df(x)}{dx}|_{x=x_i}$. One guess would be

$$\frac{df(x)}{dx}|_{x=x_i} = \frac{f_{i+1} - f_i}{\Delta x}. \tag{4}$$

This is known as the *forward difference*. But for that matter, why not the *backward difference* approximation to the derivative? That is given by

$$\frac{df(x)}{dx}|_{x=x_i} = \frac{f_i - f_{i-1}}{\Delta x}. \tag{5}$$

There is no inherent preference for the forward or backward differences. In fact, we can average the two to obtain the *central difference*, given by

$$\frac{df(x)}{dx}|_{x=x_i} = \frac{f_{i+1} - f_{i-1}}{2\Delta x}. \tag{6}$$

It can be shown that the central difference approximates the real, analytical derivative's value to $\mathcal{O}(\Delta x^2)$. Note that this central difference is based on a triplet of points, specifically x_i — where the derivative is being computed — and the set $\{x_{i-1}, x_{i+1}\}$ — where the function values are being sourced. Including a larger and larger domain of points around x_i in the computation of the derivative can drive the accuracy to higher orders of Δx, at the expense of a more complicated form of the derivative, and ultimately more complicated forms of the differential equations we want to solve numerically. This is a classic manifestation of the tradeoff between *accuracy* and *computational expense* that emerges in any numerical method.

For the purpose of this course, we will use the central difference derivative at a set of two points around the point of interest, as in Eq. 6. It can also be shown that the second derivative, using just these three points, is approximated by

$$\frac{d^2f(x)}{dx^2}|_{x=x_i} = \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta x)^2}. \tag{7}$$

Higher-order derivatives would require more than three points; luckily, in electromagnetism we never encounter spatial-derivatives of any higher order than 2.

One final point, that is more one of notation than substance: The forward difference Eq. 4 is itself a central difference, albeit for the fictitious point $x = x_{i+\frac{1}{2}}$. The backward difference Eq. 5 is also a central difference for $x = x_{i-\frac{1}{2}}$. As we will see later, this concept of a "half-grid" is often used in numerical treatments.

2.2 Boundary conditions

There is a problem with the central difference derivative as defined in Eq. 6: what happens at the boundaries? For instance, at $x = x_1$, we have

$$\frac{df(x)}{dx}|_{x=x_1} = \frac{f_2 - f_0}{2\Delta x}. \tag{8}$$

and at $x = x_N$,

$$\frac{df(x)}{dx}|_{x=x_N} = \frac{f_{N+1} - f_{N-1}}{2\Delta x}. \tag{9}$$

This is problematic, since both x_0 and $x_N + 1$ are outside the domain of the problem as defined in Eq. 3. The values f_0 and f_{N+1} are then undefined. This is handled by stating the problem's *boundary conditions*.

Two common boundary conditions are:

1. **Dirichlet boundary conditions:** This is the zero boundary condition, a statement that outside the domain of the problem the function must decay to zero. Formally we state this as

$$f_0 = f_{N+1} = 0. \tag{10}$$

2. **Periodic boundary conditions:** This is the statement that the solution to the problem must repeat itself *ad infinitum*. This is formally stated as

$$f_0 = f_N \tag{11}$$

and

$$f_{N+1} = f_1. \tag{12}$$

2.3 Derivative matrices

The discrete form of the central difference derivative Eq. 6 can also be cast in matrix form in order to operate on vectors of the form of Eq. 3. For instance, if we assume Dirichlet boundary conditions as above, we can write

$$\Delta_x = \frac{1}{2\Delta x}\begin{bmatrix} 0 & +1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 0 & +1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 0 & +1 & \dots & 0 & 0 & 0 \\ & & & & & & & \\ \vdots & & & \ddots & \ddots & \ddots & & \vdots \\ & & & & & & & \\ 0 & & & \dots & & -1 & 0 & +1 \\ 0 & & & \dots & & & -1 & 0 \end{bmatrix} \tag{13}$$

From rows $i = 2$ to $i = N - 1$ of the derivative matrix Δ_x, there is a zero on the diagonal, a -1 just to the left the diagonal, and a $+1$ just to the right of the diagonal — a statement of Eq. 6. In row $i = 1$, there is no -1, a statement of the fact that the derivative here is just $f_2/(2\Delta x)$ owing to the zero boundary condition just outside the domain. The opposite is true in the last row, corresponding to the last point, where the derivative is simply $-f_{N-1}/(2\Delta x)$.

With periodic boundary conditions instead, the derivative matrix operator is modified to read

$$\Delta_x = \frac{1}{2\Delta x}\begin{bmatrix} 0 & +1 & 0 & 0 & \dots & 0 & 0 & -1 \\ -1 & 0 & +1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 0 & +1 & \dots & 0 & 0 & 0 \\ & & & & & & & \\ \vdots & & & \ddots & \ddots & \ddots & & \vdots \\ & & & & & & & \\ 0 & & & \dots & & -1 & 0 & +1 \\ +1 & & & \dots & & & -1 & 0 \end{bmatrix} \tag{14}$$

The only change here consists of the top right and bottom left elements of the matrix, which have become -1 and $+1$, respectively. This reflects the fact that the derivative at x_1 ropes in f_N, and the derivative at x_N makes use of f_1 owing to the periodic boundary conditions.

The second derivative can analogously be transformed into matrix form as well using Eq. 7 and the appropriate boundary conditions.

Importantly, notice that these derivative matrices are **sparse matrices**.

2.4 Converting differential equations to discrete operator form

As long as a differential equation is linear, we can cast its discrete version in matrix form and solve it using the standard techniques of numerical linear algebra.

An example of this is as follows. Consider the differential equation

$$\frac{df(x)}{dx} + \eta m(x) f(x) = \ell(x). \tag{15}$$

The first term, the derivative of the function, can already be cast as $\Delta_x \vec{f}$ with $\vec{f}$ the discrete form of $f(x)$ that represents the solution to the problem. The multiplicative constant η can be left as-is. The standalone function $\ell(x)$ on the right side can also be cast as a vector on the coordinate grid as $\vec{\ell}$. The multiplication of a known function $m(x)$ with $f(x)$ requires some special consideration. By this it is meant that the values of the two functions are multiplied together everywhere point-wise on the grid. We can again write $f(x)$ in vector form $\vec{f}$. This, however, requires that $m(x)$ is cast in matrix form as the diagonal operator M

$$M = \begin{bmatrix} m_1 & & & & \\ & m_2 & & & \\ & & m_3 & & \\ & & & \ddots & \\ & & & & m_N \end{bmatrix} \tag{16}$$

with the grid-sampled values of $m(x)$ sequentially along the diagonal (and zero everywhere else).

Now, we can rewrite the differential equation as

$$\Delta_x \vec{f} + \eta M \vec{f} = \vec{\ell} \tag{17}$$

or, when factorized,

$$(\Delta_x + \eta M)\vec{f} = \vec{\ell} \tag{18}$$

which is in the form of a standard sparse linear system, $A\vec{x} = \vec{b}$ and can be solved by standard means.

2.5 Coupled differential equations

In many scenarios, we wish to solve coupled sets of differential equations for multiple functions. This is the case in electromagnetism: Maxwell's equations themselves are a coupled set of differential equations. In practice, including in electromagnetic problems, a trick is often undertaken with coupled equations. This is known as a staggered grid. This is best illustrated with an example.

Consider the set of coupled equations given by

$$\frac{df(x)}{dx} + n(x) g(x) = b_1(x) \tag{19}$$

and

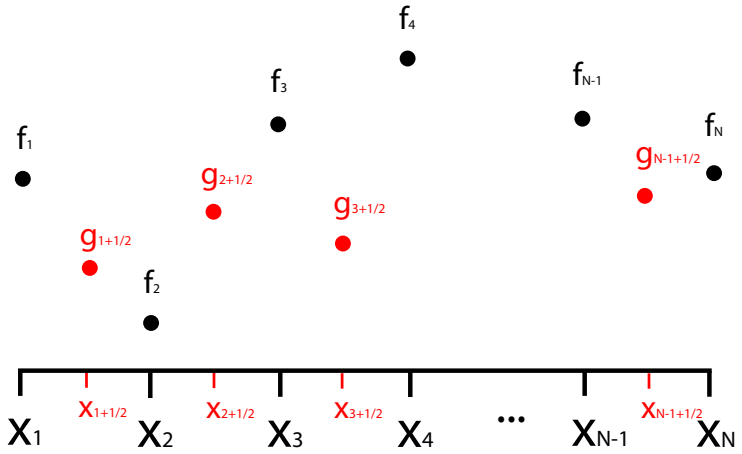

Figure 2: Defining two functions on a staggered grid. The function $f(x)$ is sampled on the even-grid (black), while the function $g(x)$ is sampled on the half-grid (red).

$$\frac{dg(x)}{dx} + m(x)f(x) = b_2(x). \tag{20}$$

The trick often taken is that a *staggered* coordinate system is used, as in Fig. 2. On this staggered coordinate system, the function $f(x)$ is discretized at even intervals of the grid (i.e., f_i) while the function $g(x)$ is sampled at half points (i.e., $g_{i+\frac{1}{2}}$, though we usually just say g_i understanding it to be sampled on the half-grid). In this staggered grid, the derivatives of f are understood to be computed on the half-grid, and the derivatives of g are understood to be computed on the even-grid. For example, we can write

$$\frac{df(x)}{dx}|_{x=x_{i+\frac{1}{2}}} = \frac{f_{i+1} - f_i}{\Delta x}. \tag{21}$$

This looks similar to the forward difference above, but it is actually a central difference for the point on the half-grid $x = x_{i+\frac{1}{2}}$. With the same memory footprint for f (we are storing the same number of samples of f overall), the accuracy of the derivative computed improves because the central difference derivative is computed over an x-span of Δx, rather than $2\Delta x$. This is the justification for using the staggered grid.

Similarly, the derivative of the function $g(x)$ comes out on the even-grid as

$$\frac{dg(x)}{dx}|_{x=x_i} = \frac{g_{i+\frac{1}{2}} - g_{i-\frac{1}{2}}}{\Delta x}. \tag{22}$$

Notably, we can again write the derivatives for the two functions in matrix form as

$$\Delta_x^f = \begin{bmatrix} -1 & +1 & & & & & \\ & -1 & +1 & & & & \\ & & -1 & +1 & & & \\ & & & \ddots & \ddots & & \\ & & & & & +1 & \\ & & & & & -1 & \end{bmatrix} \tag{23}$$

and

$$\Delta_x^g = \begin{bmatrix} +1 & & & & & & \\ -1 & +1 & & & & & \\ & -1 & +1 & & & & \\ & & \ddots & \ddots & & & \\ & & & & & & \\ & & & & & & \\ & & & & & -1 & +1 \end{bmatrix}. \tag{24}$$

The notation Δ_x^f means that the matrix is an x-derivative intended to act on quantities (such as $\vec{f}$) that are sampled on the even-grid. Both Eq. 23 nad 24 are written with Dirichlet (zero) boundary conditions.

Observe that

$$\Delta_x^f = -(\Delta_x^g)^T \tag{25}$$

with T representing a matrix transpose. Eq. 25 holds true for many (but not all) boundary conditions, including Dirichlet and periodic.

Now, writing Eqs. 21 and 22 in matrix form, we have

$$\Delta_x^f \vec{f} + N\vec{g} = \vec{b}_1 \tag{26}$$

and

$$\Delta_x^g \vec{g} + M\vec{f} = \vec{b}_2 \tag{27}$$

respectively, where N and G as in Eq. 16 are diagonal matrices including sampled values of the functions $n(x)$ and $g(x)$.

The staggered grid approach works here so long as we write Eqs. 26 and 27 with the understanding that everything in Eq. 26 is sampled on the *half*-grid, and everything in Eq. 27 is sampled on the *even*-grid. In the first case, the vector $\vec{g}$ and the derivative $\Delta_x^f \vec{f}$ are given by definition on the half-grid, so in constructing N from the continuous form of $n(x)$ and $\vec{b}_1$ from $b_1(x)$, sampling points should correspond to the half grid. The opposite is true in Eq. 27.

The trick played and advantage gained with this staggered grid approach is only possible because neither of the equations has $\vec{f}$ or $\vec{g}$ appearing together directly, which are on incompatible coordinate systems. One only appears with the derivative of the other, permitting consistency.

This too is the case in Maxwell's Equations, where electric fields only appear alongside the *derivative* of magnetic fields, and vice versa.

To actually solve the matrix equations Eqs. 26 and 27, Eq. 26 can be solved for $\vec{g}$ and substituted into Eq. 27 to generate a linear system for $\vec{f}$ alone. When

solved, $\vec{g}$ can also be generated from the result. A simple computational example is provided in the iPython notebook for this workshop.

2.6 Generalizing to higher dimensions

So far we have only talked about 1D problems. Everything said so far generalizes to two, three, or higher dimensions, provided proper bookkeeping is exercised. In higher dimensions, multi-dimensional discrete coordinate grids must be flattened to one-dimensional vectors. It is particularly important that one pays attention to the order in which this is done, to avoid unintended inconsistencies.

Here too an example of this in the Python language is given in the iPython notebook accompanying this workshop. In Python, one may wish to store, for instance, values sampled on a two- or three-dimensional grid as a two- or three-dimensional NumPy array. However, in order to interact with this data in numerically-defined versions of differential equations, the data is often flattened into 1D arrays using the `flatten` command of a NumPy `ndarray`. This can be done in either `F` (named for the Fortran language) or `C` (named for C) ordering. `F` ordering is so-called "column-major", which means that unwrapping iterates through rows first, then columns (in higher than two-dimensions, `F` ordering would mean that the unwrapping iterates through an arrays dimensions from the first listed, then the second, etc.). `C` ordering is so-called "row-major", which means that unwrapping iterates through columns first, then rows (in higher than two-dimensions, `F` ordering would mean that the unwrapping iterates through an arrays dimensions from the last listed, then the second-to-last, etc.).

Neither `F` nor `C` ordering is intrinsically better, but it is important to be consistent. The materials in these notes use `F` ordering for unwrapping/re-wrapping of arrays.