Time-Space-Synchronized FDTD Method

Bob Limnor (bob@limnor.com)

Feb 18, 2018

Abstract

A new FDTD method is developed. It uses uniform grid instead of staggered grid; it uses curls at time edge instead of at time center; it uses higher order space curls to get higher order temporal derivatives. Comparing with state of the art FDTD algorithms it is advantageous in following aspects. 1. Fields are time space synchronized. 2. Its estimation order for time advancement is arbitrary; 3. Its estimation order for space curl is arbitrary but limited to the number of space grid. 4. Its computational complexity with respect to estimation orders is linear for both time and space, meaning that using high order is not costly. 5. Estimation order can be adjusted by simply changing an integer value; and can be adjusted at any simulation time step for any space point. 6. It makes field simulation run much faster and at the same time gets much more accurate results. 7. It uses a modular design to separate estimation algorithm from applying boundary conditions. Simulation data generated by the new algorithm and by a traditional FDTD confirmed these advantages.

Introduction

I am a retired software engineer. I like to play with computer coding. Among the coding I played is Schneider's teaching book "Understanding the FDTD Method" [1] (https://github.com/john-b-schneider/uFDTD). One thing I found strange was that the field simulation results provided by an FDTD method could not be used to do calculations such as field energy transfer and field divergence. I did a quick search over the internet and did not find an answer. Since I am just playing with FDTD, not using a grant to do research, I did not bother to do a thorough research but simply came up with my owner solution. My solution is a new algorithm. To my surprise the new algorithm is better than the current FDTD in many major aspects, as explained below.

1. It uses a non-staggered grid to give simulation results in a time and space synchronized manner. Therefore it is called a Time-Space-Synchronized FDTD algorithm (TSS FDTD, or in short, TSS). I believe it is the first algorithm that can do it. Please correct me if I am wrong. This feature is my sole motivation to develop this new algorithm. It makes it possible to calculate field divergence and energy transfer. If your project requires such calculations then my algorithm is a very good option.
2. Its estimation order for time advancement is arbitrary. By "arbitrary" I mean that choosing estimation order does not affect memory usages and calculation algorithm. For example, choosing 4-th order and choosing 40-th order, same memory and programming code is used, just different numbers of iterations is used. I believe it is also the first algorithm that can do it. Increasing estimation order greatly increases simulation accuracy.

3. The higher order of time advancement estimation is realized via getting higher order temporal derivatives from higher order space curls, NOT via time history of fields. I believe this is also the first algorithm doing time advancement estimation in this way. Using time history of fields CANNOT increase estimation accuracy because there is not new information involved. Using higher order space curls for higher order temporal derivatives can greatly enhance estimation accuracy because new information is utilized.
4. Its estimation order for space curl is arbitrary but limited to the number of space grid
5. Its computational complexity with regard to the estimation order is linear. That means increasing a little calculation amount can increase accuracy greatly. Therefore, large step sizes can be used, which in turn reduces greatly the calculation amount. The combined effect is that the TSS algorithm is much faster and much more accurate than a traditional FDTD.
6. A modular design is used for designing a simulation system. It separates field estimation algorithms from applying boundary conditions. Thus, it is very easy to apply different boundary conditions.
7. It does not impose adverse restrictions, limitations, conditions, etc., to its applications comparing to traditional FDTD methods. For example, it can be used for inhomogeneous material just like a traditional FDTD does. That is, given a calculation task, if a traditional FDTD can be used then TSS algorithm can also be used.

I am going to present TSS algorithm in this file. I am also going to share my source code as open source. You are very welcome to criticize the algorithm, to provide professional guidance, to contribute to the algorithm or to the coding. You can also take its basic ideas and merge it into your own projects, just show your courtesy by presenting a reference to their origin here.

Simulation Task

Fields to be simulated

A simulation task is defined below for the purpose of describing basic ideas of the new algorithm. Calculation details may be adjusted for specific simulation tasks.

Consider a source-free electric and magnetic fields described by the Maxwell's equations

$$\frac{\partial E}{\partial t} = \frac{1}{\varepsilon} \nabla \times H \tag{1}$$

$$\frac{\partial H}{\partial t} = -\frac{1}{\mu} \nabla \times E \tag{2}$$

Where t is time, $E\ and\ H$ are 3D vectors in Cartesian coordinates (x, y, z), representing an electric field and a magnetic field, respectively, as

$$E(x,y,z,t)=\begin{bmatrix}E_x(x,y,z,t)\\E_y(x,y,z,t)\\E_z(x,y,z,t)\end{bmatrix}, H(x,y,z,t)=\begin{bmatrix}H_x(x,y,z,t)\\H_y(x,y,z,t)\\H_z(x,y,z,t)\end{bmatrix} \quad (3)$$

ε is the electric permittivity and μ is the magnetic permeability, it is assumed that they are known values.

Initial fields are assumed known:

$$E(x,y,z,t)|_{t=0}=E^{\{0\}}(x,y,z) \quad (4)$$
$$H(x,y,z,t)|_{t=0}=H^{\{0\}}(x,y,z)$$

The task is to calculate the electric field and the magnetic field in a given geometry region for time larger than 0:

$$E(x,y,z,t), H(x,y,z,t), t>0 \quad (5)$$
$$|x|\le R, |y|\le R, |z|\le R, for\ some\ R>0$$

Digitization

The time t is digitized by $\Delta_t > 0$ to be represented by an integer q:

$$t=q\Delta_t;\ \Delta_t>0, q=0,1,2,\dots \quad (6)$$

The 3D space is digitized by $\Delta_s > 0$ to be represented by 3 integers m,n,p :

$$x=m\,\Delta_s, y=n\,\Delta_s, z=p\,\Delta_s; m,n,p=0,\pm1,\pm2,\dots,r_{max} \quad (7)$$
$$r_{max}=\min_{r\Delta_s\ge R}(r)$$

A simulation task is to estimate following discrete values:

$$E(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)=\begin{bmatrix}E_x(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)\\E_y(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)\\E_z(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)\end{bmatrix}$$
$$H(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)=\begin{bmatrix}H_x(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)\\H_y(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)\\H_z(m\Delta_s,n\Delta_s,p\Delta_s,q\Delta_t)\end{bmatrix} \quad (8)$$
$$q=1,2,3,\dots$$
$$m,n,p=0,\pm1,\pm2,\dots,r_{max}$$

I'll first present the TSS algorithm to perform the above simulation task, followed by sections for developing of the algorithm. Various numerical results will be presented to show advantages of the algorithm.

The algorithm is synchronized in space and time, and thus it is referred to as a Time-Space-Synchronized Finite Difference Time Domain algorithm (TSS FDTD), or in short, a TSS algorithm. It includes time advancement estimation and space curl estimation. A simulation system also includes boundary condition modules and initial value modules. Modular design of a simulation system is also explained.

Boundary problem

Before presenting the TSS algorithm, I need to explain my understanding of the boundary problem and how a modular code design provides help and support to it.

Problem of open space

Equations (1), (2) and (4) form a Cauchy problem (initial value problem), the initial value is defined in whole space. A computer can only work on a limited space. By Kirchoff's formula, the solution at one space point can be calculated by a sphere surface integral on functions of initial values, the radius of the sphere is proportional to time. Therefore for any space point inside a simulation region, when time is long enough, the fields at that point will be determined by initial values outside of the simulation region and thus unavailable. That means that after certain simulation steps the fields at that point cannot be determined by any simulation algorithm because needed information is unavailable.

No matter how short a simulation time goes, fields at the simulation boundary are affected by initial values outside of the simulation region. Therefore, for any simulation algorithm, estimation errors due to missing information will occur at the boundary immediately on starting a simulation. That is, there is no way to get accurate estimation at the boundary.

It takes some simulation time steps before such errors affect spaces deep inside a simulation region.

Problem of close space

If the fields at or near the simulation boundary can be determined without using fields in unlimited spaces outside of the simulation region then it becomes a close space problem. It can be a physical boundary or an imaginary boundary. The open space problem is transformed into a problem of how to determine fields at or near the boundary. There are numerous "boundary conditions" described in literatures.

In a software engineer's modular way of thinking, the "boundary conditions" and the "space derivative estimation" should be two separate problems, if possible.

Modular design

Boundary condition module

Applying different kinds of boundary conditions is just setting fields at/near the boundary to different values.

We can use an interface to represent all kinds of boundary conditions, or say, to represent a "boundary condition module" in a simulation system. The simulation system is thus shield from the details of boundary conditions. The simulation system passes space points to the boundary condition module together with information such as how far the points are to the boundary; the boundary condition

module uses the information to set fields at the points according to the boundary condition algorithm the module implements.

Separating the boundary condition module from derivative estimation algorithm modules is possible because we are using non-staggered grid both in time and in space. It also makes it possible to use different boundary thicknesses, which can be difficult for staggered grids.

Space derivative estimation module

This module is responsible for space derivative estimation.

From a computer software engineer's point of view, it is to use available field data at space points to estimate space derivatives at each space point. The key idea to remember is that it doesn't matter how the available field data are made available. The field data may be made available through initial values, through applying boundary conditions, or excited by sources. How to make the field data available is a responsibility of other modules.

Therefore, by an approach of modular design, to design a space derivative estimation algorithm, we only need to consider the open space initial value problem, even though it is not a well-posed problem.

How to make the problem well-posed is the responsibility of boundary condition modules or other modules. No matter how other modules make the problem well-posed, it does not affect how this module works: use available field data to make best estimations of space derivatives.

Time advancement estimation module

This module uses higher order space curls to get higher order temporal derivatives to make time advancement estimation.

The "boundary condition module" needs to tell the simulation system how "thick" the boundary is. That is, the boundary consists of how many space steps. The simulation system tells this module to exclude how many space steps from the calculations. Once this module finishes moving the field one time step ahead, the simulation system calls the "boundary condition module" to set fields at/near the boundary.

Radius Indexing

To make it easy to plug in different boundary condition modules to perform a simulation, I developed a special space indexing system, using a concept of "pseudo radius". Using this coding technique is unnecessary in implementing the TSS algorithm. You can choose to use it or not. It is included in the source code.

Usually 3 nested loops are used to go through a 3D space, axis by axis. Suppose a 3D array A[I,j,k] stores field data. 3 nested loops can be "for(i=0; i<size;i++) {for(j=0;j<size;j++){for(k=0;k<size;k++){…use A[I,j,k]…}}}". If a 1D array A[index] is used to store field data, usually row-major format is used, still, 3 nested loops are used to go through the data, and the array index is calculated by index=k+size*(j+size*i). Nothing is wrong with this approach. But I feel not comfortable because it is not intuitive when dealing with boundaries.

I use a “radius indexing”. Instead of starting space indexing from simulation boundary, indexing starts from a center point inside the simulation region. Instead of going through space points axis by axis, it goes through space points by “radius”. For a geometry region defined by (7), the “radius” is defined by

$$r = \max(|m|, |n|, |p|)\,; m, n, p = 0, \pm 1, \pm 2, \dots, \pm r_{max}$$

In this arrangement, it is quite easy to determine the relationship of any space point with the boundary: the distance of a space point to the boundary is $r_{max} - r$.

Like in the case of row-major 3D array arrangement, where memory address is index=k+size*(j+size*i), there is also a function to get memory address from radius indexing m,n,p. There is also a function to get m,n, and p from a memory address.

An abstract C++ class is used for going through space points in radius indexing. It starts from radius 0, 1,2 ,,, and goes to the maximum radius. On changing a radius, a virtual function is called so that a derived class, such as a boundary condition module or an estimation module, knows on which radius the space points are. Then all space points on the same radius are passed to derived classes, one by one, for processing.

Time-Space-Synchronized Algorithm

Time advancement estimation

For notation simplicity in expressing TSS time advancement estimation, let’s omit the space variables:

$$\begin{aligned} E(q\Delta_t) &= E(m\Delta_s, n\Delta_s, p\Delta_s, q\Delta_t) \\ H(q\Delta_t) &= H(m\Delta_s, n\Delta_s, p\Delta_s, q\Delta_t) \end{aligned} \tag{9}$$

TSS time advancement algorithm is given by

$$E(q\Delta_t) = \sum_{k=0}^{k_{max}} \left(a_e^{[2k]} C_e^{\{2k\}} + a_h^{[2k+1]} C_h^{\{2k+1\}} \right) \tag{10}$$

$$H(q\Delta_t) = \sum_{k=0}^{k_{max}} \left(a_h^{[2k]} C_h^{\{2k\}} + a_e^{[2k+1]} C_e^{\{2k+1\}} \right) \tag{11}$$

where $k_{max} \geq 0$ is half estimation order for time advancement estimation.

$$time\ advancement\ estimation\ order = 2(k_{max} + 1)$$

Values in the right sides of the algorithm equations are calculated recursively as given below.

$a_e^{[k]}\ and\ a_h^{[k]}$ are two real numbers (or arrays, if inhomogeneous material is involved). $C_e^{\{k\}}\ and\ C_h^{\{k\}}$ are two 3D vectors. These values are calculated recursively by following formulas.

$$\beta_e = -\frac{\Delta_t}{\mu\Delta_s}, \beta_h = \frac{\Delta_t}{\varepsilon\Delta_s} \tag{12}$$

$$a_e^{[0]} = 1, a_h^{[0]} = 1 \tag{13}$$

$$C_e^{\{0\}} = E\big((q-1)\Delta_t\big), C_h^{\{0\}} = H\big((q-1)\Delta_t\big) \tag{14}$$

$$a_e^{[k]} = \frac{\beta_e}{k} a_h^{[k-1]}, a_h^{[k]} = \frac{\beta_h}{k} a_e^{[k-1]} \tag{15}$$

$$C_e^{\{k\}} = \Delta_s \nabla \times C_e^{\{k-1\}}, C_h^{\{k\}} = \Delta_s \nabla \times C_h^{\{k-1\}} \tag{16}$$

$$k = 1,2,\dots,2k_{max}+1$$

$$q = 1,2,3\dots$$

The curls in (16) are estimated by a TSS space curl estimation given in the next section.

Some discussions on the above algorithm:

1. Its estimation order can be of any even values: 2, 4, 8, …, and different estimation order can be used for each time step and each space point if so desired. It is just a matter of when to stop the recursion. Each recursion increases the estimation order by 2.
2. For $k_{max} = 0$, the TSS algorithm reduces to a 2-nd order estimation:
$$E(q\Delta_t) = E\big((q-1)\Delta_t\big) + \frac{\Delta_t}{\varepsilon}\nabla \times H\big((q-1)\Delta_t\big)$$
$$H(q\Delta_t) = H\big((q-1)\Delta_t\big) - \frac{\Delta_t}{\mu}\nabla \times E\big((q-1)\Delta_t\big)$$
The Yee's FDTD algorithm is ($\Delta_t is$ ***half*** $time\ step\ size\ for\ Yee's\ FDTD$)
$$E(q\Delta_t) = E\big((q-2)\Delta_t\big) + \frac{2\Delta_t}{\varepsilon}\nabla \times H\big((q-1)\Delta_t\big)$$
We can see that for the 2-nd order estimation, both Yee's FDTD and TSS use curl estimation at $(q-1)\Delta_t$ to represent first order temporal derivative. The difference is that for TSS, this temporal derivative value is assumed constant in a time interval of Δ_t ; for Yee's FDTD, this temporal derivative value is assumed constant in a time interval of $2\Delta_t$. Therefore, for the same 2-nd order estimation, TSS is more accurate and thus double time step size can be used. Usually time step size is proportional to the space step size. Increasing the time step size means that the space step size can be increased, this means that calculations can be reduced. It cancels the increased calculations caused by space synchronization. That is, TSS can get its benefits for free. Actually it is not only free, it is much rewarded in several aspects, including speed and precision, as shown later.
3. For the 2-nd order, 2 curl-estimations are used; an increasing of the order by 2 requires 4 more curl-estimations. So, it requires $4k_{max}+2$ curl-estimations. Thus, its computing complexity is linear to the estimation order.

4. Estimation orders for Yee's FDTD algorithms are fixed. That is why an FDTD algorithm is classified as FDTD (u, v), where u indicates the estimation order for time advancement and v indicates the estimation order for space derivative estimation. But for TSS, estimation orders are not fixed, for example, changing the estimation order for time advancement is as simple as changing an integer value of k_{max} .
5. The 2-nd order estimation needs 2 curl estimations, and the 4-th order needs 6 curl estimations. So, calculation amount needed for one 4-th order estimation can be used for 3 2-nd order estimations. Or, for the same amount of calculations, one third of the time step size can be used, resulting in more accurate estimations for the 2-nd order estimation. For a FDTD algorithm, if its estimation order is fixed then the only way to increase precision is to reduce step size. For TSS, another easy way of increasing precision is to increase the orders of the estimations, while utilizing the same amount of increased calculations. Comparing accuracy benefits of these two approaches:

$$benefit\ of\ 2nd\ order: \left(\frac{\Delta_t}{3}\right)^3 = \frac{1}{27}\Delta_t^3$$

$$benefit\ of\ 4th\ order: \Delta_t^5$$

Therefore, as long as

$$\Delta_t^5 < \frac{1}{27}\Delta_t^3 \rightarrow \Delta_t^2 < \frac{1}{27} \rightarrow \Delta_t < 0.2$$

, using higher estimation order with a larger step size gets more accurate results than using lower estimation order with a smaller step size. For a radio wave, if Δ_t is in a range of 10^{-5} then a higher order estimation is 10^{10} times more accurate than that of lower order estimation, while using the same amount of calculations and memory.

The development of the above algorithm will be given later in this file.

Inhomogeneous Material

In the above algorithm, $a_e^{[k]}\ and\ a_h^{[k]}$ are two real numbers because it is assumed that $\varepsilon\ and\ \mu$ are constants. If they are not constants then we need to use arrays for $a_e^{[k]}\ and\ a_h^{[k]}$ at each space point; the above algorithm still works.

Space Derivative Estimation

Notations

A unified notation is used to express the TSS derivative estimation algorithm:

$$v(w\Delta_s)\ is\ a\ function, w\ is\ an\ integer$$

$$v^{\{1\}}(w\Delta_s)\ is\ the\ derivative\ to\ be\ estimated$$

$$v(w\Delta_s + k\Delta_s), k = 0, \pm1, \pm2, \dots, are\ available\ function\ values$$

$$w = 0, \pm1, \pm2, \dots, \pm r_{max}$$

For example, if we want to estimate

$$\left.\frac{\partial E_x(x,y,z)}{\partial y}\right|_{x=m\Delta_s,y=n\Delta_s,z=p\Delta_s}$$

Then

$$w\Delta_s = n\Delta_s \rightarrow w = n$$

$$v(w\Delta_s + k\Delta_s) = E_x(m\Delta_s, n\Delta_s + k\Delta_s, p\Delta_s)$$

If we want to estimate

$$\left.\frac{\partial E_y(x,y,z)}{\partial z}\right|_{x=m\Delta_s,y=n\Delta_s,z=p\Delta_s}$$

Then

$$w\Delta_s = p\Delta_s \rightarrow w = p$$

$$v(w\Delta_s + k\Delta_s) = E_y(m\Delta_s, n\Delta_s, p\Delta_s + k\Delta_s)$$

Pseudo Code

The TSS derivative estimation algorithm is expressed in following pseudo code.

Suppose the estimation order is $2M, r_{max} \geq M > 0$

Step 1. Determine 3 integers *h, P, and N* by the value of w in following way.

$$if\ w \geq 0\ \{$$

$$h = r_{max} - w + 1, P = h - 1, N = -2M + h - 1$$

$$if\ h > M\ then\ h = 0, P = M, N = -M$$

$$\}$$

$$if\ w < 0\ \{$$

$$h = r_{max} + w + 1 + M, P = 2M - h + 1, N = -h + 1$$

$$if\ h > 2M\ then\ h = 0, P = M, N = -M$$

$$\}$$

Step 2. Calculate derivative estimation by a summation:

$$\Delta_s v^{\{1\}}(w\Delta_s) \approx \sum_{k=1}^{P} Q_M[h, k-1]\big(v(w\Delta_s + k\Delta_s) - v(w\Delta_s)\big) + \sum_{k=1}^{-N} Q_M[h, k+P-1]\big(v(w\Delta_s - k\Delta_s) - v(w\Delta_s)\big) \quad (17)$$

where Q_M is a $(2M+1) \times 2M$ constant matrix. This constant matrix is given in the next section. There are properties of Q_M which can be used for coding optimizations. For example, we know that

$$Q_M[0, k-1] = -Q_M[0, k+P-1]$$

And thus for h=0 the right side of (17) can be reduced to

$$\sum_{k=1}^{P} Q_M[0, k-1]\big(v(w\Delta_s + k\Delta_s) - v(w\Delta_s - k\Delta_s)\big)$$

We do not go deep into such topics to avoid distractions from the main focus of this paper.

Let's define an operator $\mathcal{D}$ for the above estimation so that we can apply it later for curl and divergence estimations:

$$\mathcal{D}(v(w\Delta_s), w) = \sum_{k=1}^{P} Q_M[h, k-1]\big(v(w\Delta_s + k\Delta_s) - v(w\Delta_s)\big) + \sum_{k=1}^{-N} Q_M[h, k+P-1]\big(v(w\Delta_s - k\Delta_s) - v(w\Delta_s)\big) \quad (18)$$

Some discussions on the above algorithm:

1. Note that it estimates the derivative multiplied by space step size. If just the actual derivative value is needed then the estimation result needs to be divided by the space step size.
2. Its estimation order can be of any even values: 2, 4, 8, …, but it cannot be larger than $2r_{max}$, which is quite a large value for an estimation order. It is unlikely an estimation order near the value of $2r_{max}$ is needed in practice. Therefore, it is fair to say that TSS gives freedom in choosing estimation order for space derivative estimations. This choosing of estimation order can be point by point if so desired.
3. Because $P + (-N) = 2M$, at one space location, the derivative estimation requires $2M$ multiplications. Therefore, the computing complexity is linear to the estimation order. Increasing the estimation order by 2 requires 2 more multiplications. This is quite a small price to pay comparing to the precision increased by a higher order.

4. Deep inside the simulation region, $h = 0$, function values from positive deviations, $w\Delta_s + k\Delta_s$, and negative deviations, $w\Delta_s - k\Delta_s$, are used. Near the positive edge, some positive deviations are out of the simulation region, negative deviations are used to replace those unavailable points; near the negative edge, some negative deviations are not available and are replaced with positive deviations. This asymmetric use of function values is how this algorithm makes estimation with available information.

Estimation matrix

Given an integer $M > 0$, a matrix Q_M used in the above estimation algorithm is calculated below.

Q_M is a matrix of $2M + 1$ rows and $2M$ columns. It is calculated row by row:

$$Q_M = \begin{bmatrix} Q_{M,0} \\ Q_{M,1} \\ \vdots \\ Q_{M,M} \\ Q_{M,M+1} \\ \vdots \\ Q_{M,2M} \end{bmatrix}, Q_{M,i}\ is\ one\ row\ of\ 2M\ elements, i = 0,1,\dots,2M \tag{19}$$

Step 1. Calculate row 0, $Q_{M,0}$

Form a $2M \times 2M$ matrix $A_{M,0}$ by

$$A_{M,0} = \begin{bmatrix} 1 & 1/2! & \dots & 1/k! & \dots & 1/(2M-1)! & 1/(2M)! \\ 2 & 2^2/2! & \dots & 2^k/k! & \dots & 2^{2M-1}/(2M-1)! & 2^{2M}/(2M)! \\ \vdots & \vdots & \cdots & \eta^k/k! & \dots & \vdots & \vdots \\ M & M^2/2! & \dots & M^k/k! & \dots & M^{2M-1}/(2M-1)! & M^{2M}/(2M)! \\ -1 & 1/2! & \dots & (-1)^k/k! & \dots & (-1)^{2M-1}/(2M-1)! & (-1)^{2M}/(2M)! \\ -2 & 2^2/2! & \dots & (-2)^k/k! & \dots & (-2)^{2M-1}/(2M-1)! & (-2)^{2M}/(2M)! \\ \vdots & \vdots & \cdots & (-\eta)^k/k! & \dots & \vdots & \vdots \\ -M & M^2/2! & \dots & (-M)^k/k! & \dots & (-M)^{2M-1}/(2M-1)! & (-M)^{2M}/(2M)! \end{bmatrix} \tag{20}$$

$$Q_{M,0} = the\ first\ row\ of\ A_{M,0}^{-1} \tag{21}$$

Step 2. Calculate row 1 to row M.

Form a $2M \times 2M$ matrix $A_{M,h}$ by

$$A_{M,h} = \tag{22}$$

$$\begin{bmatrix} 1 & 1/2! & \dots & 1/k! & \dots & 1/(2M-1)! & 1/(2M)! \\ 2 & 2^2/2! & \dots & 2^k/k! & \dots & 2^{2M-1}/(2M-1)! & 2^{2M}/(2M)! \\ \vdots & \vdots & \cdots & \eta^k/k! & \dots & \vdots & \vdots \\ h-1 & (h-1)^2/2! & \dots & (h-1)^k/k! & \dots & (h-1)^{2M-1}/(2M-1)! & (h-1)^{2M}/(2M)! \\ -1 & 1/2! & \dots & (-1)^k/k! & \dots & (-1)^{2M-1}/(2M-1)! & (-1)^{2M}/(2M)! \\ -2 & 2^2/2! & \dots & (-2)^k/k! & \dots & (-2)^{2M-1}/(2M-1)! & (-2)^{2M}/(2M)! \\ \vdots & \vdots & \cdots & (-\eta)^k/k! & \dots & \vdots & \vdots \\ -M & M^2/2! & \dots & (-M)^k/k! & \dots & (-M)^{2M-1}/(2M-1)! & (-M)^{2M}/(2M)! \\ -(M+1) & (M+1)^2/2! & \dots & (-M-1)^k/k! & \dots & (-M-1)^{2M-1}/(2M-1)! & (-M-1)^{2M}/(2M)! \\ & & & \vdots & & & \\ -(2M-h+1) & (2M-h+1)^2/2! & \dots & (-2M+h-1)^k/k! & \dots & (-2M+h-1)^{2M-1}/(2M-1)! & (-2M+h-1)^{2M}/(2M)! \end{bmatrix}$$

$$Q_{M,h} = the\ first\ row\ of\ A_{M,h}^{-1} \quad h = 1,2,\dots,M \tag{23}$$

Cells of the first column of the first $h-1$ rows are positive values $1,2,\dots,h-1$. Cells of the first column of other rows are negative values. We call the first $h-1$ rows "positive rows". Note that for $h=1$ there is not a "positive row"; the first row starts with cell "-1".

Step 3. Calculate row M+1 to row 2M.

Form a $2M \times 2M$ matrix $A_{M,h+M}$ by

$$A_{M,h+M} = \begin{bmatrix} 1 & 1/2! & \dots & 1/k! & \dots & 1/(2M-1)! & 1/(2M)! \\ 2 & 2^2/2! & \dots & 2^k/k! & \dots & 2^{2M-1}/(2M-1)! & 2^{2M}/(2M)! \\ \vdots & \vdots & \cdots & \eta^k/k! & \dots & \vdots & \vdots \\ M & M^2/2! & \dots & M^k/k! & \dots & M^{2M-1}/(2M-1)! & M^{2M}/(2M)! \\ M+1 & (M+1)^2/2! & \dots & (M+1)^k/k! & \dots & (M+1)^{2M-1}/(2M-1)! & (M+1)^{2M}/(2M)! \\ & & & \vdots & & & \\ 2M-h+1 & (2M-h+1)^2/2! & \dots & (2M-h+1)^k/k! & \dots & (2M-h+1)^{2M-1}/(2M-1)! & (2M-h+1)^{2M}/(2M)! \\ -1 & 1/2! & \dots & (-1)^k/k! & \dots & (-1)^{2M-1}/(2M-1)! & (-1)^{2M}/(2M)! \\ -2 & 2^2/2! & \dots & (-2)^k/k! & \dots & (-2)^{2M-1}/(2M-1)! & (-2)^{2M}/(2M)! \\ \vdots & \vdots & \cdots & (-\eta)^k/k! & \dots & \vdots & \vdots \\ -(h-1) & (h-1)^2/2! & \dots & (-h+1)^k/k! & \dots & (-h+1)^{2M-1}/(2M-1)! & (-h+1)^{2M}/(2M)! \end{bmatrix} \tag{24}$$

$$Q_{M,h+M} = the\ first\ row\ of\ A_{M,h+M}^{-1} \quad h = 1,2,\dots,M \tag{25}$$

Cells of the first column of the last $h-1$ rows are negative values $-1,-2,\dots,-(h-1)$. Cells of the first column of other rows are positive values. We call the last $h-1$ rows "negative rows". For $h=1$ there is not a "negative row"; the last row starts with cell "$2M$".

Some discussions on the above calculations:

1. Q_M is a constant matrix, its elements are constant real numbers. It is unrelated to a specific simulation task.

2. Before starting a simulation Q_M should be calculated or loaded from a file.
3. To enable adjusting estimation order at runtime, before starting a simulation, Q_p for $p = 1,2,\dots,M$ should be prepared.

Space curl estimation

For a 3D vector

$$V(x,y,z) = \begin{bmatrix} V_x(x,y,z) \\ V_y(x,y,z) \\ V_z(x,y,z) \end{bmatrix} \tag{26}$$

Its curl is calculated by

$$\nabla \times V = \begin{bmatrix} \frac{\partial V_z}{\partial y} - \frac{\partial V_y}{\partial z} \\ \frac{\partial V_x}{\partial z} - \frac{\partial V_z}{\partial x} \\ \frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y} \end{bmatrix} \tag{27}$$

In digitized space, the vector is represented by

$$V(m\Delta_s, n\Delta_s, p\Delta_s) = \begin{bmatrix} V_x(m\Delta_s, n\Delta_s, p\Delta_s) \\ V_y(m\Delta_s, n\Delta_s, p\Delta_s) \\ V_z(m\Delta_s, n\Delta_s, p\Delta_s) \end{bmatrix} \tag{28}$$

To simplify the notations, let's omit the space step size:

$$V(m,n,p) = \begin{bmatrix} V_x(m,n,p) \\ V_y(m,n,p) \\ V_z(m,n,p) \end{bmatrix} \tag{29}$$

Applying operator $\mathcal{D}$ defined in the last section to (27), we have curl estimation:

$$\Delta_s \nabla \times V(m,n,p) \approx \begin{bmatrix} \mathcal{D}(V_z(m,n,p),n) - \mathcal{D}\big(V_y(m,n,p),p\big) \\ \mathcal{D}(V_x(m,n,p),p) - \mathcal{D}(V_z(m,n,p),m) \\ \mathcal{D}\big(V_y(m,n,p),m\big) - \mathcal{D}(V_x(m,n,p),n) \end{bmatrix} \tag{30}$$

It is accomplished by 6 derivative estimations.

Define an operator $\mathcal{C}url$ for the above calculation:

$$\mathcal{C}url(V) = \begin{bmatrix} \mathcal{D}(V_z(m,n,p),n) - \mathcal{D}\big(V_y(m,n,p),p\big) \\ \mathcal{D}(V_x(m,n,p),p) - \mathcal{D}(V_z(m,n,p),m) \\ \mathcal{D}\big(V_y(m,n,p),m\big) - \mathcal{D}(V_x(m,n,p),n) \end{bmatrix} \tag{31}$$
$$m,n,p = 0,\pm1,\pm2,\dots,\pm r_{max}$$

Apply operator $\mathcal{C}url$ to the time advancement estimation (16), we have

$$C_e^{\{0\}} = E\big((q-1)\Delta_t\big), C_h^{\{0\}} = H\big((q-1)\Delta_t\big) \tag{32}$$

$$C_e^{\{k\}} = \mathcal{C}url\left(C_e^{\{k-1\}}\right), C_h^{\{k\}} = \mathcal{C}url\left(C_h^{\{k-1\}}\right)$$

$$k = 1,2,\dots,2k_{max}+1$$

Now all calculations in the TSS algorithm are defined.

Algorithm Accuracy Criterion

For a simulation algorithm, its accuracy should be a difference between simulated fields and the fields given by the solutions of the Maxwell's equations. Since we cannot get a solution of the Maxwell's equations, we cannot calculate the true simulation error. GAO et al [2] got "optimal error estimates", which were error bounds. We want very accurate comparison of algorithm accuracy. We cannot use error bounds to compare real precisions of different algorithms.

To evaluate different field simulation algorithms, in lieu of a better criterion, we use field divergence as a criterion for the precision of a simulation. A smaller divergence is considered to be more accurate. Note that this criterion is only valid for source free fields, not for fields excited by a source.

Using operator $\mathcal{D}$, field divergence is estimated by

$$Divergence\ of\ E(m,n,p) = \mathcal{D}\left(E_x(m,n,p),m\right) + \mathcal{D}\left(E_y(m,n,p),n\right) + \mathcal{D}\left(E_z(m,n,p),p\right)$$

$$Divergence\ of\ H(m,n,p) = \mathcal{D}\left(H_x(m,n,p),m\right) + \mathcal{D}\left(H_y(m,n,p),n\right) + \mathcal{D}\left(H_z(m,n,p),p\right)$$

We can do various statistics on the divergences to evaluate simulation accuracy. For example, use average values:

$$Average\ Divergence\ of\ E = \frac{sum(|Divergence\ of\ E(m,n,p)|, m,n,p = 0,\pm1,\dots,\pm r_{max})}{number\ of\ space\ points}$$

$$Average\ Divergence\ of\ H = \frac{sum(|Divergence\ of\ H(m,n,p)|, m,n,p = 0,\pm1,\dots,\pm r_{max})}{number\ of\ space\ points}$$

Development of the TSS algorithm

The TSS algorithm is presented in previous sections for you to implement it in your projects. In this section, the TSS algorithm is developed and proved.

Time advancement

For a 3D vector V , denote its multiple curls by

$$\nabla^{\{k\}} \times V = \underbrace{\nabla \times \nabla \times \ldots \times \nabla}_{k} \times V \tag{33}$$

That is,

$$\nabla^{\{0\}} \times V = V$$

$$\nabla^{\{1\}} \times V = \nabla \times V$$

$$\nabla^{\{2\}} \times V = \nabla \times \nabla \times V$$

And so on.

Extend (14) and (16) to a continuous varying time:

$$C_e^{\{0\}}(t) = E(t), C_h^{\{0\}}(t) = H(t) \tag{34}$$

$$C_e^{\{k\}}(t) = \Delta_s \nabla \times C_e^{\{k-1\}}(t), C_h^{\{k\}}(t) = \Delta_s \nabla \times C_h^{\{k-1\}}(t) \tag{35}$$

Using notation of (33), the above definitions can be written as

$$C_e^{\{k\}}(t) = \Delta_s^k \nabla^{\{k\}} \times E(t) \tag{36}$$

$$C_h^{\{k\}}(t) = \Delta_s^k \nabla^{\{k\}} \times H(t) \tag{37}$$

Assume the order of temporal derivative and space curls is exchangeable on $E(t) and\ H(t)$, such that

$$\frac{\partial \left(\nabla^{\{k\}} \times E(t)\right)}{\partial t} = \nabla^{\{k\}} \times \frac{\partial E(t)}{\partial t}$$

$$\frac{\partial \left(\nabla^{\{k\}} \times H(t)\right)}{\partial t} = \nabla^{\{k\}} \times \frac{\partial H(t)}{\partial t}$$

Applying (1) and (2) to (36) and (37), we have

$$\Delta_s^k \nabla^{\{k\}} \times \frac{\partial E(t)}{\partial t} = \Delta_s^k \nabla^{\{k\}} \times \frac{1}{\varepsilon} \nabla \times H = \frac{\Delta_s^k}{\varepsilon} \nabla^{\{k+1\}} \times H(t) = \frac{1}{\varepsilon \Delta_s} \Delta_s^{k+1} \nabla^{\{k+1\}} \times H(t) = \frac{1}{\varepsilon \Delta_s} C_h^{k+1}$$

$$\Delta_s^k \nabla^{\{k\}} \times \frac{\partial H(t)}{\partial t} = \Delta_s^k \nabla^{\{k\}} \times \left(-\frac{1}{\mu} \nabla \times E\right) = -\frac{\Delta_s^k}{\mu} \nabla^{\{k+1\}} \times E(t) = -\frac{1}{\mu \Delta_s} \Delta_s^{k+1} \nabla^{\{k+1\}} \times E(t)$$
$$= -\frac{1}{\mu \Delta_s} C_e^{k+1}$$

Thus, we have

$$\frac{\partial \left(C_e^{\{k\}}(t)\right)}{\partial t} = \frac{1}{\varepsilon \Delta_s} C_h^{k+1}(t) \tag{38}$$

$$\frac{\partial \left(C_h^{\{k\}}(t)\right)}{\partial t} = -\frac{1}{\mu \Delta_s} C_e^{k+1}(t) \tag{39}$$

With the above results, we can prove following lemma.

Lemma. For any integer $k \geq 0$

$$\left.\frac{\Delta_t^{2k}}{2k!} \frac{\partial^{2k} E(t)}{\partial t^{2k}}\right|_{t=(q-1)\Delta_t} = a_e^{[2k]} C_e^{\{2k\}} \tag{40}$$

$$\left.\frac{\Delta_t^{2k+1}}{(2k+1)!} \frac{\partial^{2k+1} E(t)}{\partial t^{2k+1}}\right|_{t=(q-1)\Delta_t} = a_h^{[2k+1]} C_h^{\{2k+1\}} \tag{40}$$

$$\left.\frac{\Delta_t^{2k}}{2k!} \frac{\partial^{2k} H(t)}{\partial t^{2k}}\right|_{t=(q-1)\Delta_t} = a_h^{[2k]} C_h^{\{2k\}} \tag{42}$$

$$\left.\frac{\Delta_t^{2k+1}}{(2k+1)!} \frac{\partial^{2k+1} H(t)}{\partial t^{2k+1}}\right|_{t=(q-1)\Delta_t} = a_e^{[2k+1]} C_e^{\{2k+1\}} \tag{41}$$

Proof.

For $k = 0$, (40) and (42) hold, because the left side of (40) and (42) become

$$\left.\frac{\Delta_t^{2k}}{2k!} \frac{\partial^{2k} E(t)}{\partial t^{2k}}\right|_{t=(q-1)\Delta_t} = E\big((q-1)\Delta_t\big) = a_e^{[0]} C_e^{\{0\}}$$

$$\left.\frac{\Delta_t^{2k}}{2k!} \frac{\partial^{2k} H(t)}{\partial t^{2k}}\right|_{t=(q-1)\Delta_t} = H\big((q-1)\Delta_t\big) = a_h^{[0]} C_h^{\{0\}}$$

By (1), (12), (13),(14),(15) and (16), for $k = 0$ the left side of (41) becomes

$$\frac{\Delta_t^{2k+1}}{(2k+1)!}\frac{\partial^{2k+1}E(t)}{\partial t^{2k+1}}\bigg|_{t=(q-1)\Delta_t} = \Delta_t \frac{\partial E(t)}{\partial t}\bigg|_{t=(q-1)\Delta_t} = \frac{\Delta_t}{\varepsilon}\nabla \times H\big((q-1)\Delta_t\big)$$

$$= \frac{\Delta_t}{\varepsilon\Delta_s}\Delta_s \nabla \times C_h^{\{0\}} = a_h^{[1]} C_h^{\{1\}}$$

By (2), (12), (13),(14),(15) and (16), for $k = 0$ the left side of (43) becomes

$$\frac{\Delta_t^{2k+1}}{(2k+1)!}\frac{\partial^{2k+1}H(t)}{\partial t^{2k+1}}\bigg|_{t=(q-1)\Delta_t} = \Delta_t \frac{\partial H(t)}{\partial t}\bigg|_{t=(q-1)\Delta_t} = -\frac{\Delta_t}{\mu}\nabla \times E\big((q-1)\Delta_t\big)$$

$$= -\frac{\Delta_t}{\mu\Delta_s}\Delta_s \nabla \times C_e^{\{0\}} = a_e^{[1]} C_e^{\{1\}}$$

Thus, for $k = 0$ the lemma holds.

Suppose for $k > 0$ the lemma holds.

For $k+1$, the left side of (40) becomes

$$\frac{\Delta_t^{2(k+1)}}{2(k+1)!}\frac{\partial^{2(k+1)}E(t)}{\partial t^{2(k+1)}}\bigg|_{t=(q-1)\Delta_t} = \frac{\Delta_t}{2K+2}\frac{\Delta_t^{2k+1}}{(2k+1)!}\frac{\partial\left(\frac{\partial^{2k+1}E(t)}{\partial t^{2k+1}}\right)}{\partial t}\bigg|_{t=(q-1)\Delta_t}$$

Because (41) holds for k , we have

$$\frac{\Delta_t}{2K+2}\frac{\Delta_t^{2k+1}}{(2k+1)!}\frac{\partial\left(\frac{\partial^{2k+1}E(t)}{\partial t^{2k+1}}\right)}{\partial t}\bigg|_{t=(q-1)\Delta_t} = \frac{\Delta_t}{2K+2}a_h^{[2k+1]}\frac{\partial\left(C_h^{\{2k+1\}}(t)\right)}{\partial t}\bigg|_{t=(q-1)\Delta_t}$$

By (39), (12), and (15), we have

$$= \frac{\Delta_t}{2K+2}a_h^{[2k+1]}\left(-\frac{1}{\mu\Delta_s}C_e^{\{2k+2\}}\right) = \frac{\beta_e}{2k+2}a_h^{[2k+1]}C_e^{\{2k+2\}}$$

$$= a_e^{[2k+2]}C_e^{\{2k+2\}}$$

Thus, (40) holds for $k+1$.

Use $k+1$ in left side of (41), we have

$$\frac{\Delta_t^{2(k+1)+1}}{(2(k+1)+1)!}\frac{\partial^{2(k+1)+1}E(t)}{\partial t^{2(k+1)+1}}\bigg|_{t=(q-1)\Delta_t} = \frac{\Delta_t}{2k+3}\frac{\Delta_t^{2k+2}}{(2k+2)!}\frac{\partial\left(\partial^{2k+2}E(t)/\partial t^{2k+2}\right)}{\partial t}\bigg|_{t=(q-1)\Delta_t}$$

Because (40) holds for $k+1$, the above becomes

$$= \frac{\Delta_t}{2k+3} \frac{\partial \left(a_e^{2k+2} C_e^{\{2k+2\}}(t) \right)}{\partial t} \Bigg|_{t=(q-1)\Delta_t} = \frac{\Delta_t}{2k+3} a_e^{2k+2} \frac{\partial \left(C_e^{\{2k+2\}}(t) \right)}{\partial t} \Bigg|_{t=(q-1)\Delta_t}$$

By (38), (12), and (15), the above becomes

$$= \frac{\Delta_t}{2k+3} a_e^{2k+2} \frac{1}{\varepsilon \Delta_s} C_h^{2k+3} = \frac{\beta_h}{2k+3} a_e^{2k+2} C_h^{2k+3} = a_h^{2k+3} C_h^{2k+3}$$

Thus, (41) holds for $k+1$.

Use $k+1$ in left side of (42), we have

$$\frac{\Delta_t^{2(k+1)}}{2(k+1)!} \frac{\partial^{2(k+1)} H(t)}{\partial t^{2(k+1)}} \Bigg|_{t=(q-1)\Delta_t} = \frac{\Delta_t}{2k+2} \frac{\Delta_t^{2k+1}}{(2k+1)!} \frac{\partial}{\partial t} \left(\frac{\partial^{2k+1} H(t)}{\partial t^{2k+1}} \right) \Bigg|_{t=(q-1)\Delta_t}$$

Because (43) holds for k, the above becomes

$$= \frac{\Delta_t}{2k+2} \frac{\partial}{\partial t} \left(a_e^{[2k+1]} C_e^{\{2k+1\}} \right) \Big|_{t=(q-1)\Delta_t}$$

By (38), (12) and (15), we have

$$= \frac{\Delta_t}{2k+2} a_e^{[2k+1]} \frac{1}{\varepsilon \Delta_s} C_h^{\{2k+2\}} = \frac{\beta_h}{2k+2} a_e^{[2k+1]} C_h^{\{2k+2\}} = a_h^{[2k+2]} C_h^{\{2k+2\}}$$

Thus, (42) holds for $k+1$.

Use $k+1$ in left side of (43), we have

$$\frac{\Delta_t^{2(k+1)+1}}{(2(k+1)+1)!} \frac{\partial^{2(k+1)+1} H(t)}{\partial t^{2(k+1)+1}} \Bigg|_{t=(q-1)\Delta_t} = \frac{\Delta_t}{2k+3} \frac{\Delta_t^{2k+2}}{(2k+2)!} \frac{\partial}{\partial t} \left(\frac{\partial^{2k+2} H(t)}{\partial t^{2k+2}} \right) \Bigg|_{t=(q-1)\Delta_t}$$

Because (42) holds for $k+1$, the above becomes

$$= \frac{\Delta_t}{2k+3} \frac{\partial}{\partial t} \left(a_h^{[2k+2]} C_h^{\{2k+2\}}(t) \right) \Big|_{t=(q-1)\Delta_t}$$

By (39), (12) and (15), the above becomes

$$= \frac{\Delta_t}{2k+3} a_h^{[2k+2]} \left(-\frac{1}{\mu \Delta_s} C_e^{2k+3} \right) = \frac{\beta_e}{2k+3} a_h^{[2k+2]} C_e^{\{2k+3\}} = a_e^{[2k+3]} C_e^{\{2k+3\}}$$

Thus, (43) holds for $k+1$.

Thus, the lemma holds for any integer $k \geq 0$.

QED.

Now we may derive TSS time advancement algorithm given in (10) and (11).

By Taylor series, we have

$$E(t+\Delta_t)=\sum_{n=0}^{\infty}\frac{\Delta_t^n}{n!}\frac{\partial^n E(t)}{\partial t^n} \tag{42}$$

$$H(t+\Delta_t)=\sum_{n=0}^{\infty}\frac{\Delta_t^n}{n!}\frac{\partial^n H(t)}{\partial t^n} \tag{43}$$

Rearrange the terms in (44) and (45),

$$E(t+\Delta_t)=\sum_{k=0}^{\infty}\left(\frac{\Delta_t^{2k}}{2k!}\frac{\partial^{2k}E(t)}{\partial t^{2k}}+\frac{\Delta_t^{2k+1}}{(2k+1)!}\frac{\partial^{2k+1}E(t)}{\partial t^{2k+1}}\right) \tag{44}$$

$$H(t+\Delta_t)=\sum_{k=0}^{\infty}\left(\frac{\Delta_t^{2k}}{2k!}\frac{\partial^{2k}H(t)}{\partial t^{2k}}+\frac{\Delta_t^{2k+1}}{(2k+1)!}\frac{\partial^{2k+1}H(t)}{\partial t^{2k+1}}\right) \tag{45}$$

Substitute (40) and (41) into (46), and let $t=(q-1)\Delta_t$, we have

$$E(q\Delta_t)=\sum_{k=0}^{\infty}\left(a_e^{[2k]}C_e^{\{2k\}}+a_h^{[2k+1]}C_h^{\{2k+1\}}\right) \tag{46}$$

Substitute (42) and (43) into (47), and let $t=(q-1)\Delta_t$, we have

$$H(q\Delta_t)=\sum_{k=0}^{\infty}\left(a_h^{[2k]}C_h^{\{2k\}}+a_e^{[2k+1]}C_e^{\{2k+1\}}\right) \tag{47}$$

Truncate at $k=0,1,2\ldots,k_{max;}\,k_{max}\geq 0$ in above summations, we have the estimation formulas (10) and (11).

Space derivative estimation

To simplify notations, for a function $v(s)$ define

$$v_i(w\Delta_s)=\Delta_s^i\frac{\partial^i v(s)}{\partial s^i}\Big|_{s=w\Delta_s}$$
$$w=0,\pm1,\pm2,\ldots,\pm r_{max} \tag{50}$$
$$i=0,1,2,\ldots$$

Given an integer $k\neq 0$, using the above notations in a Taylor's series gives

$$v(w\Delta_s + k\Delta_s) = \sum_{i=0}^{\infty} \frac{(k\Delta_s)^i}{i!} \frac{\partial^i v(s)}{\partial s^i}\Big|_{s=w\Delta_s} = \sum_{i=0}^{\infty} \frac{k^i}{i!} v_i(w\Delta_s) \quad \{51\}$$

Our purpose is to estimate $v_1(w\Delta_s)$ from values of $v(w\Delta_s + k\Delta_s)$. We want to use points closest to $w\Delta_s$, which means small $|k|$ values such as $k = \pm 1, \pm 2, \ldots$ under a constraint of $|w + k| \le r_{max}$.

Suppose we want to use $2M$ smallest $|k|$ values, $M > 0$. We use two integers $P\ and\ N$ to specify these k values corresponding to w.

$$P > 0, N < 0, P - N = 2M, P = 0,1,2, \ldots, 2M \quad \{52\}$$

k values are defined by $P\ and\ N$ as

$$1 \le k \le P, -1 \ge k \ge N \quad \{53\}$$

Note that for $P = 0$ there is not a positive k, it happens when $w = r_{max}$; for $P = 2M$ there is not a negative k, it happens when $w = -r_{max}$. P and N are determined by w to minimize $|k|$ values:

$$P = \begin{cases} r_{max} - w; w \ge r_{max} - M \\ M; M - r_{max} < w < r_{max} - M \\ 2M - r_{max} - w; w \le M - r_{max} \end{cases} \quad \{54\}$$

For each k in (51) we use 2M+1 terms in (51) for estimations. We get 2M linear equations for 2M unknowns. Arrange these equations into a linear mapping form:

$$A_{M,P} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{2M} \end{bmatrix} \approx \begin{bmatrix} v(w\Delta_s + \Delta_s) - v(w\Delta_s) \\ v(w\Delta_s + 2\Delta_s) - v(w\Delta_s) \\ \vdots \\ v(w\Delta_s + P\Delta_s) - v(w\Delta_s) \\ v(w\Delta_s - \Delta_s) - v(w\Delta_s) \\ v(w\Delta_s - 2\Delta_s) - v(w\Delta_s) \\ \vdots \\ v(w\Delta_s - (-N)\Delta_s) - v(w\Delta_s) \end{bmatrix} \quad \{55\}$$

where $A_{M,P}$ is a $2M\ by\ 2M$ matrix formed by coefficients in (51) from term 1 to term 2M: $\frac{k^i}{i!}, i = 1,2, \ldots, 2M$. k is the integer factor before Δ_s in (51):

$$A_{M,P} = \begin{bmatrix} 1 & 1/2! & \ldots & 1/k! & \ldots & 1/(2M-1)! & 1/(2M)! \\ 2 & 2^2/2! & \ldots & 2^k/k! & \ldots & 2^{2M-1}/(2M-1)! & 2^{2M}/(2M)! \\ \vdots & \vdots & \cdots & \eta^k/k! & \ldots & \vdots & \vdots \\ P & P^2/2! & \ldots & P^k/k! & \ldots & P^{2M-1}/(2M-1)! & P^{2M}/(2M)! \\ -1 & 1/2! & \ldots & (-1)^k/k! & \ldots & (-1)^{2M-1}/(2M-1)! & (-1)^{2M}/(2M)! \\ -2 & 2^2/2! & \ldots & (-2)^k/k! & \ldots & (-2)^{2M-1}/(2M-1)! & (-2)^{2M}/(2M)! \\ \vdots & \vdots & \cdots & (-\eta)^k/k! & \ldots & \vdots & \vdots \\ N & N^2/2! & \ldots & N^k/k! & \ldots & N^{2M-1}/(2M-1)! & N^{2M}/(2M)! \end{bmatrix} \quad \{56\}$$

Cells of the first column of $A_{M,P}$ are k values. The first P rows are for positive k values; let's call them "positive rows". The last $-N$ rows are for negative k values; let's call them "negative rows". If $P = 0$ then there is not a positive row. If $P = 2M$ then there is not a negative row.

Solving (55):

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{2M} \end{bmatrix} \approx A_{M,P}^{-1} \begin{bmatrix} v(w\Delta_s + \Delta_s) - v(w\Delta_s) \\ v(w\Delta_s + 2\Delta_s) - v(w\Delta_s) \\ \vdots \\ v(w\Delta_s + P\Delta_s) - v(w\Delta_s) \\ v(w\Delta_s - \Delta_s) - v(w\Delta_s) \\ v(w\Delta_s - 2\Delta_s) - v(w\Delta_s) \\ \vdots \\ v(w\Delta_s - (-N)\Delta_s) - v(w\Delta_s) \end{bmatrix} \quad \{57\}$$

We only need the solution for v_1 , therefore, we only need the first row of $A_{M,P}^{-1}$.

For each value of $P = 0,1,2,\ldots,2M$, form $A_{M,P}$ by (56), record the first row of $A_{M,P}^{-1}$. This preparation should be done before a simulation starts. During a simulation, (54) is used to decide which row to use to calculate a space derivative estimation. This is a trivial process; an example of this process is described previously in pseudo code.

Further documents

Following documents will be uploaded.

1. Software design specifications
2. Numerical Results
3. Source code in GitHub

References

[1] John B. Schneider, Understanding the Finite-Difference Time-Domain Method, www.eecs.wsu.edu/~schneidj/ufdtd, 2010.

[2] LIPING GAO AND BO ZHANG, "OPTIMAL ERROR ESTIMATES AND ENERGY CONSERVATION IDENTITIES OF THE ADI-FDTD SCHEME ON STAGGERED GRIDS FOR 3D MAXWELL'S EQUATIONS", Sci. China Math. (2013) 56: 1705