Purpose: This exercise illustrates numerical issues relevant for construction of initial conditions of Maxwell equations. Utilizing a simple Maxwell solver, dispersion and stability properties of numerical waves are examined and compared with theoretical predictions.

Take-away message:

- Unlike in BPM, initial conditions in Maxwell solvers require that the magnetic and electric fields are properly "orchestrated."
- Proper relation between the initial values of the electric and magnetic fields is determined by the direction of propagation of the initial waveform, and is influenced by the dispersion properties of the algorithm.
- The integration step of the numerical algorithm must be sufficiently small in order to avoid instability.
- Carefully designed numerical experiment is required to verify the theoretical numerical dispersion relation.

Tasks:

A) Implement one-dimensional Maxwell solver based on the Yee scheme. Code the discretized equation derived in the class. To keep the program as concise as possible, assume periodic boundary conditions (PBC). PBC mean that a point at the right-most end of the computational domain can be identified with the very first point at the left end of the domain:

$$E[N-1] = E[0] \quad , \quad H[N-1] = H[0] \ .$$

Here, N stands for the number of points in the one-dimensional computational "box," and c-like array indexing is assumed (i.e. the very first array index is zero). Choose the unit of length equal to the grid spacing, and the speed of light equal to one. With such a choice of units, the only free parameter characterizing the method is the Courant ratio $\Delta t/\Delta x$. For your initial runs, set the Courant ratio less or equal to 0.5.

To further simplify the programming, you can take advantage of the fact that the update of both, H and E field can be done in-place, utilizing the same array holders for the current and new field snapshots. However, keep in mind that this is not possible in higher dimensions and when simulated waves interact with the propagation medium.

B) Construct the initial condition in a form of a pulse characterized by a given wavelength. Give the pulse a Gaussian-shaped envelope. To ensure that the spectrum of the waveform is narrow, the envelope must encompass at least several wavelengths. Assume that the electric field component is given by the formula which you will implement. The question is how one needs to choose the magnetic field in order to initialize the solution such that the pulse propagates to the right?

C) Execute a short simulation and inspect the outcome carefully to see if the solution indeed propagates in the desired direction. Zoom in so that small features are not missed. Most likely, you will find that while the bulk of the pulse does propagate in one direction, there is a low-intensity "ghost" propagating in the opposite direction.

a) Explain the origin of the ghost

b) Discuss various options to achieve truly clean, one-way propagating initial conditions.

D) Verify that the simulation method becomes unstable for the Courant ratio larger or equal to one.

E) Design a method, and collect simulation data to verify the theoretic numerical dispersion relation derived in the class. Briefly describe the method of your choice, and discuss specifically what potential numerical artifacts could affect the results. Make sure that your measured numerical dispersion data span the whole region of wavelengths admissible for the discrete grid.

The following figure illustrates two cases of dispersion, and the accuracy you should achieve:

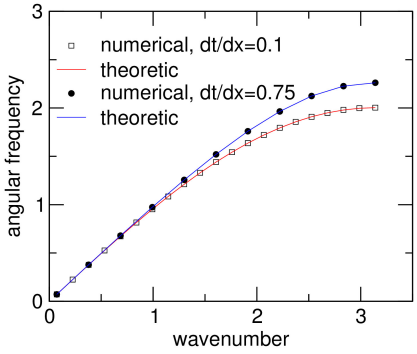


Figure 1. When an initial condition has a sharply-defined spatial wavenumber, the numerical integration algorithm produces a solution with a specific temporal angular frequency. This figure illustrates the dispersion relation which ties the spatial and temporal frequencies. Curves show the expected relation derived from the update scheme, and symbols are results of numerical simulations. Data for different Courant ratios indicate that the dispersion relation depends on $\Delta t/\Delta x$. The continuum limit corresponds to a straight line with unit slope.

Solution

Task A)

The implementation of the one-dimensional Maxwell integration step is straightforward. Here we use C-language to illustrate that the core of the procedure is nothing but a direct realization of the scheme derived in the class. A complete instructor's solution can be viewed in *OneDMaxwell.c*

Listing 1: 1D-Maxwell integration step implementation

```
1 void OneStep(double *E, double *H, double dtoverdx, int n) {
2   int i;
3
4   for(i=0;i<n-1;i++) E[i] = E[i] + dtoverdx*(H[i+1] - H[i  ]);
5   E[n-1] = E[0];
6
7   for(i=1;i<n  ;i++) H[i] = H[i] + dtoverdx*(E[i  ] - E[i-1]);
8   H[0]   = H[n-1];
9 }
```

Periodic boundary conditions are realized in lines 5 and 8 for electric and magnetic fields, respectively. The way they are implemented here requires one additional grid point where the field sample is a copy "slaved" to the value on the other side of the grid. An alternative method would utilize indexing which wraps the indices that "reach" beyond the array end like this:

$$H[i+1] \rightarrow H[(i+1)\%N]$$

This is computationally more expensive than the addition of an additional boundary grid point.

Note that there seems to be a slight asymmetry in the way electric and magnetic field arrays are indexed. This depends on the chosen correspondence between the index and the location in space it represents. In the present case $H[i]$ is located on the left from $E[i]$.

Task B)

In the beam propagation method, the initial condition is simply a specification of the electric (vector) field for $z = 0$. It is assumed tacitly that the corresponding values of the magnetic field are orchestrated such that the beam solution propagates in the forward direction. In most BPM versions, magnetic fields are never calculated.

In the case of direct Maxwell solver, the issue of the initial condition is slightly more subtle. It is up to the programmer to ensure that the whole (initial) electromagnetic field has desired properties. Often it is the requirement that the initial field represents a pulsed waveform propagating in the direction of positive z-axis.

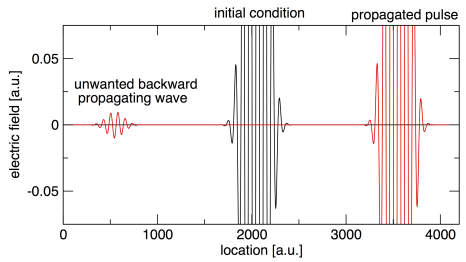
The principle that guides the construction of the initial condition is that the relation between the vector amplitudes of the electric and magnetic fields in the numerical solution should mimic that in real electromagnetic plane waves. Because only one-dimensional propagation is considered in this section, the relation simplifies. If one chooses the computational units such that the numerical field samples represent $E \to E_y$ and $H \to cB_z$ for propagation along x, then $H = \pm E$ corresponds to the amplitude relation in a left- and right propagating harmonic wave. The choice of the sign selects the propagation direction along the positive or negative x-axis direction.

So it seems that if one chooses $H(x, t = 0) = \pm E(x, t = 0)$ to define the magnetic field in the initial condition the resulting wavepacket should propagate either forward or backward as a whole. However, one must keep in mind that the numerical grid points for the two fields are staggered. This is reflected in the following example:

Listing 2: Initial condition, take one

```
1  for(i=0;i<N;i++) {
2  /*
3  We aim to orchestrate the magnetic field with the electric
4  such that the initial waveform will propagate to the right
5  Plane-wave properties dictate the opposite signs between
6  the electric field and magnetic field amplitudes
7  */
8
9   double x0 = 0.5*((double) N);
10  double xe = (double) i;
11
12 // same-index magnetic location shifted by half of lattice spacing:
13  double xh = xe - 0.5;
14
15  EF[i] = +InitialElectricField(xe, x0);
16  HF[i] = -InitialElectricField(xh, x0);
17
18 // ... which is almost working ... but not accurately enough ...
19 }
```

Here, $x0$ only marks the location where we want to place the center of the wavepacket. xe is a holder for the coordinate (derived from the array index i) corresponding to the spatial location of an electric field sample. xh plays the same role for the magnetic field sample. In line 10, it is shifted left by one half of the spatial grid spacing, and subsequently the same function is used to generate both electric and magnetic field values. The result is that the spatial profile of the electric and magnetic field is the same, as is expected for a one-way propagating electromagnetic pulse (in vacuum).



Inaccurate implementation of the one-way propagating initial condition in a one-dimensional Maxwell solver. The scale of the figure cuts the peaks of the waveform in the initial (black) and propagated (red) pulses, in order to emphasize the weak pulse in the left portion of this figure. This is a wavepacket that propagates in the opposite direction than the bulk of the pulse. It is an unwanted artifact that can not be removed by refining either grid resolution or the integration step.

Task C)

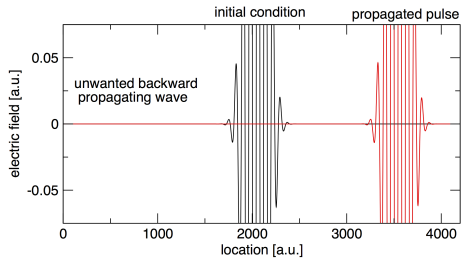
The figure above illustrates that this implementation of the initial does not work very well. It appears that the initial pulse splits into two identically shaped waveforms that propagate in the opposite directions. An undesired weak pulse is "ejected" from the initial condition. It is left to the reader to verify that it can not be eliminated by better grid resolution or by refining the integration step, because its amplitude decreases with smaller Δx and/or Δt but remains significant for all practically usable parameters.

What is wrong with the above-described realization of the initial condition? The implementation did take into account that the electric and magnetic field grids are staggered in space, but did not account for the fact that they are also staggered in time. The temporal stagger means that the waveform of the magnetic field, which represents time earlier by $\Delta t/2$, must be shifted by the distance that the radiation travels during that time. This shift is accounted for in the following modified code:

Listing 3: Initial condition, take two

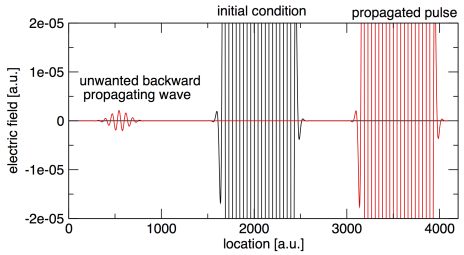
```
1    for(i=0;i<N;i++) {
2      // here we aim to orchestrate the magnetic field with the electric
3      // such that the initial waveform will propagate to the right
4      // this part is the same as before ...
5
6      double x0 = 0.5*((double) N);
7      double xe = (double) i;
8      double xh = xe - 0.5;
9
10     // ... but one has to account for different time:
11     xh -= 0.5*dtoverdx;
12
13     EF[i] = +InitialElectricField(xe, x0);
14     HF[i] = -InitialElectricField(xh, x0);
15   }
```

The addition is in line 11 where the the coordinate where the magnetic field is evaluated from the function that specifies the electric field initial condition is shifted by $1/2c\Delta t$ (with $c = 1$).



Better implementation of the one-way propagating initial condition in a one-dimensional Maxwell solver. The scale of the figure is the same as in the previous one. In this case the backward propagating pulse is not visible.

So it seems that the initial condition is properly constructed and does not produce any pulse propagation in the unwanted direction. However, as soon as one zooms into the figure (shown below) the artifact can be readily identified. We thus see that while the initial condition is much better, and the artifact is about three orders of magnitude weaker, it is still not perfect. At this point the origin of this undesirable feature may seem mysterious, but we shall soon understand that it originates in the numerical dispersion; Numerical dispersion is the reason why the propagation speed of the initial pulse is not exactly equal to $c = 1$. Because we have assumed just that in our correction of the initial condition, the latter is not completely one-way propagating.

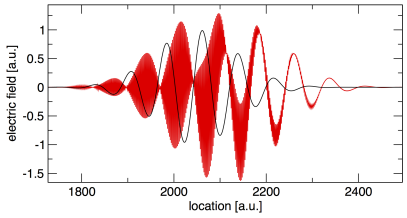


The same data shown at a finer vertical resolution reveals that a very weak pulse propagating in the wrong direction still exists...

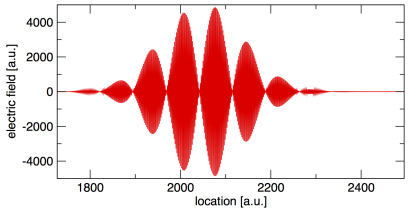
It should be emphasized that the initial condition as constructed in this exercise is sufficiently accurate for practical purposes. After all, in terms of pulse energy, only about one part in 10^{10} propagates in the wrong direction. Nevertheless, the appearance of the artifact, however minute, illustrates that the numerical dispersion, discussed in what follows, affects all aspects of the numerical algorithm.

Task D)

For this part of the exercise, we set the Courant ratio equal to, say, 1.01. It is most instructive to keep it quite close to unity, so that one has a chance to observe that instability develops gradually, albeit extremely fast. This is illustrated in the following figures.



Instability onset in Maxwell simulation. One hundred and twenty integration steps were executed with the Courant ratio chosen equal to 1.01, starting from the initial condition shown in solid black. This simulation was interrupted at a time chosen to reveal the very onset of numerical artifacts. As it is often the case, even-odd "oscillations" suddenly appear in the solution (red line) and continue to grow.



Only thirty additional integration steps are sufficient to amplify numerical artifacts by several orders of magnitude. Exponential growth of numerical noise is usually fast enough to make the solution to deteriorate so quickly that no gradual transition from a well-behaved, smooth solution can be noticed.

In practice, numerical instability is manifested through solutions that appear noisy and exhibit very large positive and negative values. It may also happen that an unstable solution appears smooth, but growths exponentially.

E)

There are many valid ways to solve this problem. We will illustrate two approaches, and invite the reader to explore their variations.

The first issue one has to deal with is the setup of a solution with a well-defined wavenumber. The initial condition must ensure that the wavenumber is preserved during the simulation and the evolution will exhibit a single angular frequency — this will only happen if the wavenumber is chosen from the discrete set of values implied by the grid resolution and the computational domain size. Of course this relies on the periodic boundary conditions.

The second issue is how to deal with the fact that waves propagate in both directions. One must either create truly one-way propagating initial condition (as we did in the first part of this exercise) or, alternatively, one can take advantage of a symmetric situation in which the intensity of waves in opposite direction is strictly equal. The second option is simpler, as it only requires that the initial magnetic field is set to zero. Left-right symmetry of the update scheme then ensures that forward and backward component of the waveform is exactly the same. We will therefore initialize the simulation run with a standing wave with a chosen spatial wavenumber, and follow its evolution to see what angular frequency it will "produce."

The third problem is a method to extract the numerical value of the angular frequency that the algorithm produces. This can be done in many ways. For example, a direct though not very elegant method is to let the simulation running, and simply count zero-crossings at an arbitrary but fixed location in space. Together with the elapsed time this gives the information needed to calculate the value of ω. Another method is to compare solutions after a single step — for this approach one has to derive a formula that relates the solution at two times, and isolate ω from it.

Two versions of instructor's solutions are hidden out of sight in the sub-directory named *Solution*. We describe their main features next. It is strongly recommended that the reader tries to come up with his/her own approach before reading further.

Initialization

Listing 4: Initialization for dispersion measurement

```
1
2   // set the Courant ratio, dispersion will depend on this value
3   double dtoverdx = 0.1;
4
5   int el;     // scaled spatial wavenumber
6   double k0;  // spatial wavenumber - we measure omega for each
7
8   // this starts the main measurement loop
9   for(el=N/2-1;el>0;el-=100) {
10
11  // spatial wavenumber must have one of the grid-supported values
12  k0 = el*2.0*M_PI/((double) N - 1.0);
13
14  // the initial condition has zero magnetic field
15  for(int i=0;i<N;i++) {
16    double xe = (double) i;
17
18    EF[i] = +cos(xe*k0);
```

```
19      HF[i] = 0.0;
20    }
```

There are a couple of points to note in the above code snippet. First, $\Delta t/\Delta x$ is fixed such that the integration scheme is stable. The dispersion curve depends on this value, of course. Then the main loop, controlled by el, scans values of spatial wavenumbers for which the angular frequencies are to be measured. The wavenumber value $k0$ is carefully chosen in line 12 such that the corresponding plane wave smoothly wraps around the periodic boundary condition of the computational domain. The initial electric field is specified as a harmonic wave, and because the initial magnetic field vanishes, this simulation will evolve a standing wave with the spatial wavenumber $k0$.

Counting zero crossings

The next code section runs the simulation for a chosen value of $k0$, and counts zero crossing of the electric field at the grid point $i = 0$:

Listing 5: Counting zero-crossings

```
1
2     // this is the spatial point, i=0, where we follow temporal evolution
3     double EFold = EF[0];
4
5     int start  = 0;  // this variable to indicate the first zero-crossing was detected
6     int istart = 0;  // step at which the first crossing was detected
7     int count  = 0;  // zero-crossing counter
8
9     for(i=0;count<50;i++) {        // go until fifty crossings
10      OneStep(EF,HF,dtoverdx,N);   // integration step
11
12    // this is where we test for zero-crossing
13      if( EF[0]*EFold <= 0.0 ) {
14        if(start == 0) { start = 1; istart = i;count=0;}
15        else  count++;
16      }
17
18    // keep the previous field value to compare with the new one
19      EFold = EF[0];
20    }
21
22    // here, the angular frequency is approximated based on the number of detected
23    // zero crossings
24    i--;
25    double T = 2.0*dtoverdx*(i - istart)/((double) count); // estimated period
26    double omega = 2.0*M_PI/T;                             // estimated omega
27
28    // theoretical value to show in the output
29    double YeeValue = 2.0/dtoverdx*asin( dtoverdx*sin(k0/2.0));
30
```

```
31      printf("%E %E %E\n",k0,omega,YeeValue);
32      } // end of wave-number-scan loop
```

This program can measure the numerical relation between the spatial wavenumber and angular frequency with a good accuracy. There are a few obvious drawbacks. The accuracy of the temporal period estimate, T, is limited by the (hard-coded in line 9) number of periods to simulate. Thus, to increase the accuracy, length of the simulation run must be increased accordingly. Moreover, the duration of the measurement at low wavenumber (and low frequency) increases as $1/\omega$. Nevertheless, these are simulation parameters that can be easily controlled and chosen to reflect the accuracy one aims for. Because the one-dimensional Maxwell simulation does not require significant numerical effort, increase in simulation time is not a really serious issue.

It is an easy fix to increase the accuracy of this measurement without increasing the numerical effort. With only fifty period long simulation the the accuracy is limited to a few percent. This can be easily improved by better localization of the first and last zero crossings. For example, one can use simple linear interpolation to estimate the true location of the zero by linear interpolation between the old and new field samples $EFold$ and $EF[0]$. This allows to improve the measurement of the time between zero crossing of the electric field to an accuracy better than Δt. As a result, the the quality of the measurement increases significantly. It is left to the reader to implement this improvement.

Extracting angular frequency from subsequent field snapshots

The previous method can be characterized as straightforward. It has a big advantage that the numerical measurement is direct, and does not require additions to the simulation code that would be based on "further development or considerations." If designed as a test of the algorithm implementation, this is exactly what one wants. On the other hand, the method can be hardly considered elegant. Next we show an alternative measurement of chromatic properties of the Yee algorithm. It only requires a couple of simulation step to execute, and does not suffer any accuracy issues. However, it requires to derive a formula to relate the simulated field after one step to the initial one in order to extract the numerical dispersion relation. Let us derive this formula.

Consider the electric field value, as evolved by the Yee algorithm, at a fixed location. The observation spot can be chosen arbitrarily with one condition that the initial field at this point is not zero. When set up as in the method described before, i.e. with the initial magnetic field equal to zero, the initial value of the electric field, denoted by E_0 corresponds to time $t = -\Delta t/2$. The next value at time $t = \Delta t/2$ is denoted by E_1, but it turns out to be equal to E_1 since the magnetic field vanishes during the very first update of the electric field. The subsequent step creates the field value denoted E_2, and it corresponds to time $t = 3/2\Delta t$. All three values are related, because they represent harmonic oscillation with an "unknown" amplitude A:

$$E_0 = A\cos\left(-\frac{1}{2}\omega\Delta t\right) \qquad E_1 = A\cos\left(+\frac{1}{2}\omega\Delta t\right) = E_0 \qquad E_2 = A\cos\left(+\frac{3}{2}\omega\Delta t\right) \tag{1}$$

Thus, it is sufficient to to execute two steps to generate E_2, and ω can be calculated from these equations through elimination of A. Probably the simplest way to to this is to use

$$\cos(a+b) = \cos a\cos b - \sin a\sin b \qquad \text{and} \qquad \sin(a+b) = \sin a\cos b + \cos a\sin b$$

to obtain

$$E_2 = A\left[4\cos^3\left(\frac{1}{2}\omega\Delta t\right) - 3\cos\left(\frac{1}{2}\omega\Delta t\right)\right] = E_0\left[4\cos^2\left(\frac{1}{2}\omega\Delta t\right) - 3\right] \tag{2}$$

Angular frequency ω can be expressed from the last relation as

$$\omega = \frac{2}{\Delta t} \arccos \left[\frac{1}{2} \sqrt{\frac{E_2 + 3E_0}{E_0}} \right] \tag{3}$$

Numerical measurement based on this formula can be implemented as shown in the following program listing

Listing 6: Angular frequency extracted from subsequent simulated field samples

```
1   // listing starts within the main wave-number-scan loop
2
3   // save the initial field
4   EFold = EF[0];
5
6   // execute two steps:
7   // this step does not change electric field due to zero H
8   OneStep(EF,HF,dtoverdx,N);
9   // and this step does...
10  OneStep(EF,HF,dtoverdx,N);
11
12  // formula XX
13  double omega=2.0/dtoverdx*acos(sqrt((EF[0]+3.0*EFold)/EFold)/2.0);
14
15  // theoretical formula value
16  double YeeValue = 2.0/dtoverdx*asin( dtoverdx*sin(k0/2.0));
17
18  // show results
19  printf("%E %E %E\n",k0,omega,YeeValue);
20  } // end of the wave-number-scan
```

Application of either of the two methods will produce data similar to that shown in Figure 1. Readers should experiment with different parameters for these simulations. In particular, it should become obvious that numerical dispersion becomes more and more pronounced for high-frequency waves. The deviation from the ideal, continuum-limit dispersion decreases with decreasing integration step, but numerical waves with the spatial frequency in the vicinity of the Nquist frequency never propagate as their real counterparts.