4

Crank-Nicolson method

This chapter deals with the Crank-Nicolson method, which is arguably one of the most important building blocks in many numerical schemes. Its significance and applications go way beyond the field of beam propagation. Having illustrated in the previous sections the methods that exhibited serious drawbacks both in accuracy and stability, here we put those lessons together to construct an approach that

- is implicit, and unconditionally stable,
- is computationally effective because it relies on tri-diagonal systems,
- is 2-nd order accurate in both Δz and Δx
- can be utilized as a building block in more complex methods

The scheme was first developed by John Crank and Phyllis Nicolson, and published in 1947 as *"A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type"* in Proceedings of the Cambridge Philosophical Society. Since then it has been applied in many different contexts, and also re-invented numerous times under different names.

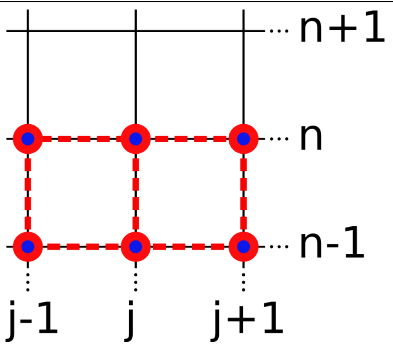
Specifically in the BPM field, methods are sometimes intentionally formulated such that sub-problems arise that can subsequently be treated by the Crank-Nicolson technique or something very similar. An operator-splitting approach is one such method discussed later in this course where the C-N approach can be utilized to calculate evolution due to components of the split evolution operator.

4.1 Crank-Nicolson method for one transverse dimension

4.1.1 Crank-Nicolson stencil

One way to look at the approach we are about to discuss, is to present it as an avergae of the explicit and implicit methods. However, a better way to intuit why the result of such averaging has the good properties it does is to ask for the most symmetric discretization stencil one can get with a reasonably small number of grid points involved. Symmetrization in the spatial discretization is already achieved by the three-point scheme for the Laplacian operator in both explicit and implicit method. But both also have an unwanted property that the discretization stencil *center* is different for the z-differences and x-differences, with the z-derivative being best approximated at *half*-step $n - 1/2$. This discrepancy can be removed by the averaging of the spatial derivative approximation between steps $n - 1$ and n. The result is a scheme which gives an approximation that is best exactly at a "virtual" mid-point $n - 1/2$.

The Crank-Nicolson scheme consists in discretizing temporal and spatial derivatives in a symmetric way such that their "reference point" (i.e. the center of the discretization stencil) is located half-way between two discrete steps. The stencil encompasses six grid points, as shown in the following figure:



Crank-Nicolson discretization stencil. Both the transverse (along x, label j) and longitudinal (along z, the propagation direction, labeled n) derivatives are approximated with respect to the geometric center of this stencil. The result is the second-order accuracy in both dimensions. Implicit nature of the scheme then gives the unconditional stability.

three of them at each discrete propagation step, here labeled n and $n+1$. The partial differential equation to be solved is approximated by finite differences with respect to the geometric center of this stencil.

1. The evolution operator ∂_z:

$$\partial_z \mathcal{E}(x = x_j, z = z_n + \Delta z/2) \approx \frac{1}{\Delta z}(E_j^{n+1} - E_j^n) \quad , \tag{4.1}$$

which is centered at step $n + 1/2$ and second-order accurate with respect to it.

2. One-dimensional Laplacian along the transverse dimension:

$$\partial_{xx} \mathcal{E}(x = x_j, z = z_n + \Delta z/2) \approx \frac{1}{\Delta x^2}\left(E_{j-1}^{n+1} - 2E_j^{n+1} + E_{j+1}^{n+1}\right) + \frac{1}{\Delta x^2}\left(E_{j-1}^n - 2E_j^n + E_{j+1}^n\right) \tag{4.2}$$

is centered at the same location, and is also second-order accurate.

4.1.2 Operator notation

As a rule, discretized expression similar to those above appear repeatedly in derivations and in final scheme formulations. That is why in the BPM literature such finite-difference expressions are often short-handed, and operator symbols are used to make everything (hopefully) more readable. For example, the one-dimensional Laplacian-related difference expression is represented by

$$\Delta_{ji} E_i^n \equiv E_{j-1}^n - 2E_j^n + E_{j+1}^n \tag{4.3}$$

where Δ_{ij} is understood as an operator (a tri-diagonal matrix in fact), acting on the vector E_j^n:

$$\Delta_{ij} E_j^n = \begin{pmatrix} -2 & v_0 & 0 & \cdots & \cdots & 0 & 0 \\ u_1 & -2 & v_1 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & u_j & -2 & v_j & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & u_{N-1} & -2 \end{pmatrix} \begin{pmatrix} E_0^n \\ E_1^n \\ \vdots \\ E_j^n \\ \vdots \\ E_{N-1}^n \end{pmatrix} \tag{4.4}$$

with $u_j = 1$ and $v_j = 1$. This will be modified later to treat also the radially symmetric case, when u_j and v_j become index-dependent. We will also use the symbolic "operator" notation for this finite difference operation,

$$\{E^n_{j-1} - 2E^n_j + E^n_{j+1}\} = \varDelta E^n \tag{4.5}$$

where the right hand side is understood as matrix-vector product.

4.1.3 Crank-Nicolson system of equations

Putting together pieces prepared in the previous subsections, Eq. (3.23) results in an implicit difference system

$$E^{n+1} - E^n = i\delta(\varDelta E^{n+1} + \varDelta E^n) \tag{4.6}$$

with $\delta = \varDelta z/4k_0(\varDelta r)^2$ — this shorthand will be used quite frequently. Objects in the above equation are to be understood as vectors (arrays) of electric field samples, and $\varDelta$ is the difference operator matrix (4.4).

A formal solution can be obtained by collecting E^{n+1},

$$(1 - i\delta\varDelta)E^{n+1} = (1 + i\delta\varDelta)E^n \tag{4.7}$$

and acting on this equation by the inverse of the operator that appears on the left:

$$E^{n+1} = (1 - i\delta\varDelta)^{-1}(1 + i\delta\varDelta)E^n, \tag{4.8}$$

One could worry if the inverse used here even exists – let us just say here it does, in the subspace of solutions, or functions representing propagating waves, as it will become clearer in what follows.

This form, in which the new propagated field E^{n+1} is expressed explicitly, is not the best for numerical evaluation. It is true that in Matlab it is a simple matter to construct a matrix for $\varDelta$, and then ask for the inverse operator we need. In the one-dimensional case, Matlab will readily return the inverse matrix and it could be used in calculations. This is, however, very inefficient. A better way to deal with inverse operators applied to vectors it to solve the corresponding linear system of equations, and that is how Crank-Nicolson method should be implemented.

But (4.8) was cast this way for a reason, namely that it manifestly shows that the scheme preserves the norm (energy) of the solution. If $\varDelta$ was not an operator (matrix) but a number, the ratio

$$\frac{1 + i\delta\varDelta}{1 - i\delta\varDelta} \tag{4.9}$$

would be a complex number with a unit modulus. That in turn would mean that the norm of the solution did not change. While $\varDelta$ is an operator, this reasoning still works as long as the vector on which it acts, is an eigenvector of $\varDelta$ — then the operator can be replaced by the corresponding eigenvalue. The set of plane-waves are such eigenvectors with real eigenvalues. Since this set is complete in the sense that it can serve as a basis, one just needs to repeat the above argument for each eigenvector. This approach will be applied to calculation of dispersion properties of the C-N scheme.

The actual numerical implementation of the Crank-Nicolson method follows (4.7), which is written as

$$L^-_{ij}E^{n+1}_j = L^+_{ij}E^n_j \tag{4.10}$$

to define the following sparse matrices

$$L^+ \equiv 1 + i\delta\varDelta \qquad L^- \equiv 1 - i\delta\varDelta.$$

It should be emphasized that there is no need to allocate memory space and store these matrices. One only needs to

- calculate right-hand-side vector: $R_i = L^+_{i,j} E^n_j$,
- and solve the tri-diagonal system of linear equations: $L^-_{ij} E^{n+1}_j = R_i$

Tri-diagonal systems can be solved very efficiently. Unlike general systems of linear equations, the computational complexity only scales linearly with the number of grid points. This is at the root of the success Crank-Nicolson method has had. The other reason for its wide spread is favorable dispersion properties, discussed next.

4.1.4 Numerical dispersion and stability properties

For an experienced eye, much of the numerical properties of the scheme can be seen directly form the operator form of the discretized equation. Without any calculations at all, one can say the following:

- Operator Δ is real and symmetric, and thus Hermitian. This in turn means that a) the eigenvalues are real, and b) there exists a basis made up of eigenvectors of the operator — these are actually the familiar plane waves "propagating" in the two-dimensional space spanned by the transverse dimensions x, y.
- Operator $(1 - i\delta\Delta)^{-1}(1 + i\delta\Delta)$ is therefore unitary, which is to say that when it acts on a vector the result has the same norm as the input. This is an important property for beam propagation, thanks to which energy of a numerical solution remains conserved after each discrete propagation update.
- Because the update $E^n \to E^{n+1}$ is done by a norm-preserving operation independently of how long the integration step is, this **method is unconditionally stable** — it does not amplify undesirable numerical noise. It also does not artificially damp numerical waves (recall the simple implicit method which did).

Next, all of the above will be derived explicitly in an elementary way. Consider an infinite 1D-domain, and on that domain choose plane-waves specified by their spatial wave-number k:

$$\psi_j(k) = e^{ikx_j} \tag{4.11}$$

This exponential form exhibits or represents the translational symmetry in our one-dimensional transverse space. Similarly, there is translational symmetry along the propagation direction z, so it is natural to look for a solution in the form

$$E(z_n, x_j) = Ae^{iK_z z_n}\psi_j(k) \ . \tag{4.12}$$

Here, K_z is so far an unknown propagation constant that will depend on the chosen transverse wavenumber, $K_z = K_z(k)$.

Before inserting into C-N method formula (4.8), realize that ψ are eigenvectors of the finite-difference operator Δ:

$$\Delta_{ij}\psi_j = e^{ikx_{i-1}} - 2e^{ikx_i} + e^{ikx_{i+1}} = e^{ikx_i}(e^{-ik\Delta x} - 2 + e^{+ik\Delta x}) = 2(\cos k\Delta x - 1)\psi_i \tag{4.13}$$

Obviously, the corresponding eigenvalue is $2(\cos k\Delta x - 1)$, and one can see that it is real, precisely as expected. When the operators contained in the RHS of (4.8) act on the plane wave, the latter is preserved, but multiplied by this eigenvalue. Consequently,

$$(1 - i\delta\Delta)^{-1}(1 + i\delta\Delta)\psi = \frac{1 + 2i\delta(\cos k\Delta x - 1)}{1 - 2i\delta(\cos k\Delta x - 1)}\psi \tag{4.14}$$

This is used to insert the solution ansatz in (4.8), and requiring that the amplitude A is non-trivial yields the following solvability condition that relates K_z and k:

$$e^{iK_z\Delta z} = \frac{1+2i\delta(\cos k\Delta x-1)}{1-2i\delta(\cos k\Delta x-1)} = \frac{1+\frac{i\Delta z}{2k_0\Delta x^2}(\cos k\Delta x-1)}{1-\frac{i\Delta z}{2k_0\Delta x^2}(\cos k\Delta x-1)} \tag{4.15}$$

This is the numerical dispersion relation of the one-dimensional Crank-Nicolson scheme. One can immediately see that K_z is purely real, because the expression on the right has an absolute value equal to one. The fact that no matter what k is chosen a real-valued K_z is obtained implies unconditional stability; The amplitude of any plane wave does not increase or decrease, and since any solution can be expressed as a superposition of plane waves, its energy or, equivalently, norm also does not change. No matter how long the propagation step Δz is, this method will remain stable (of course, long steps will degrade accuracy).

Having established the stability of the scheme, let us look closer at the accuracy. Based on the way the paraxial beam propagation was discretized we know that the local accuracy is of second order in both the transverse and propagation directions. This, however, tells us little about how numerical wave propagation differs from that of real waves this method is supposed to describe. To illuminate this issue a bit more, express the propagation constant for the given transverse wavenumber k explicitly,

$$K_z(k) = \frac{-i}{\Delta z}\ln\left[\frac{1+\frac{i\Delta z}{2k_0\Delta x^2}(\cos k\Delta x-1)}{1-\frac{i\Delta z}{2k_0\Delta x^2}(\cos k\Delta x-1)}\right] , \tag{4.16}$$

and Taylor expand around small k:

$$K_z(k) \approx -\frac{k^2}{2k_0} + \frac{\Delta x^2 k^4}{24k_0} + \left(\frac{\Delta z^2}{96k_0^3} - \frac{\Delta x^4}{720k_0}\right)k^6 + O[k]^7 \tag{4.17}$$

The first term is the exact propagation constant of the continuum paraxial beam propagation equation, so the method has the correct continuum limit (this is by no means a surprise, anything else would only indicate error in the calculation — the continuum limit is obtained correctly in “any” method).

The remaining terms indicate how fast errors diminish when the transverse grid spacing Δx decreases. Note that the first correction does not depend on the integration step Δz, so the main factor responsible for the “gap” between the numerical and exact wave propagation is the resolution of the spatial grid. This error decreases with the second power of Δx indicating the second-order accuracy in transverse dimension. The third term contains Δz^2 which is in line with this scheme's second-order accuracy in the propagation direction.

Exercise:
Parametrize the transverse wavenumber as $k = 2\pi/(n\Delta x)$. Then n stands for the “transverse wavelength” (note that it is significantly larger than the vacuum wavelength, at least in typical BPM scenarios) in units of grid spacing. This is the “number of points per wavelength” parameter often used to characterize resolution in the context of Maxwell solvers. Show in a graph how $K_z(k)/K_z^{(continuum)}$ depends on n for several values of $\delta = \Delta z/(2k_0\Delta x^2)$ (the resulting expression should not depend on any other parameters). What should n be in order to approximate $K_z(k)$ with an accuracy of 1% ?

This exercise illustrates how inefficiently a finite-difference code utilizes the “bandwidth” allowed by the discrete grid. In other words, one must “allocate” a significant number of discrete sampling points per one oscillation of a waveform for its propagation properties to be realistic. This was much less of a problem in spectral methods discussed in the previous Section.

4.1.5 Numerical solution

The most practical way to solve the C-N system was identified in a previous section without giving much detail; here we collect the necessary formulas in an explicit form to make the implementation of this algorithm straightforward.

One evolution step amounts to solving the implicit equation for E^{n+1}. In the operator form it reads

$$(1 - i\delta\mathit{\Delta})E^{n+1} = (1 + i\delta\mathit{\Delta})E^n \ . \tag{4.18}$$

Field samples E^{n+1} and E^n are represented in the computer as arrays, or vectors. Assume that the length of these arrays is N. Then, the above is nothing but a system of linear equations with unknowns $\{E_i^{n+1}\}_{i=0}^{N-1}$. Explicitly,

$$\sum_{j=0}^{N-1}(\delta_{ij} - i\delta\mathit{\Delta}_{ij})E_j^{n+1} = \sum_{k=0}^{N-1}(\delta_{ik} + i\delta\mathit{\Delta}_{ik})E_k^n \qquad , i = 0, 1, \ldots N-1 \tag{4.19}$$

where

$$\mathit{\Delta}_{ij}E_j^n = \begin{pmatrix} -2 & v_0 & 0 & \cdots & \cdots & 0 & 0 \\ u_1 & -2 & v_1 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & u_j & -2 & v_j & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & u_{N-1} & -2 \end{pmatrix} \begin{pmatrix} E_0^n \\ E_1^n \\ \vdots \\ E_j^n \\ \vdots \\ E_{N-1}^n \end{pmatrix} \tag{4.20}$$

with $u_j = 1$ and $v_j = 1$. Boundary points deserve a closer look. What is written above tacitly assumes that the electric field amplitude vanishes immediately outside of the computational domain proper. In other words, the field samples E_0 and E_{N-1} are updated the same way as their "inner" counterparts but with "ghost boundary-layer points" carrying zero field values:

$$E_{-1} = 0 \quad \text{and} \quad E_N = 0 \quad . \tag{4.21}$$

This is a boundary condition that represents **perfect electric conductor** (PEC) walls at $x_{j=-1}$ and $x_{j=N}$. It will cause perfect reflection of waves that propagate from the inside against the domain boundary. A whole section will be devoted to the question of how one can simulate free space outside of the computational domain.

The fact that the system of equations to solve has a tri-diagonal matrix is most important. This makes it possible to solve it in time linear in N. The algorithm used in a tri-diagonal matrix solver is called Thomas algorithm after a British atomic physicist. It is essentially Gaussian elimination executed in the straightforward order (i.e. without pivoting), followed by backward substitution. The important property is that the procedure is stable and thus applicable to large systems.

Exercise:
Periodic boundary conditions in a computational domain are more a nuisance than something one would really want. Let us pretend anyway that the above system should be solved with periodic boundary conditions. Modify the system accordingly (changes will appear in both the system‘s matrix and in the right-hand-side vector), and research an efficient way to solve it.

4.2 Practice track: Properties of the Crank-Nicolson method

Summary:

- Implementation of the Crank-Nicolson BPM scheme for a computational domain with single transverse dimension.
- Validation against the analytic one-dimensional Gaussian beam propagation.
- Convergence study: Numerical verification of the accuracy order.
- Advantages of the approach over the purely implicit discretization scheme become evident even in basic numerical experiments.

4.2.1 Implementation and validation

This practice track section concentrates on the implementation of the Crank-Nicolson method to solve the paraxial beam propagation equation, and on numerical experiments to study an verify its properties. We continue to utilize the "operator" notation adopted in the previous section dealing with the implicit update scheme. It should be become obvious that having written a beam simulator for the implicit method in the higher-level language, it only takes few lines of additional code to upgrade the simulator for the Crank-Nicolson approach.

Task 1: Method implementation
Inspect the implementation of the implicit method from the previous section and upgrade it to use the Crank-Nicolson scheme. In particular, the initialization of the two matrices (see lecture notes) $L^{\pm}$ that constitute the 'C-N operator pair' is similar: These operators can be initialized and stored as sparse matrices. The method of solution of the resulting linear system is also left up to Matlab. Note that this is not the most efficient approach (why?), and that a proper implementation using a tri-diagonal linear system solver will be discussed in the following exercise.

Solution:
Reader should utilize the program for the implicit method as a point of departure. There are essentially only two points in which the methods differ. The first is that besides $L^{(-)}$, $L^{(-)}$ must be also precalculated:

Listing 4.1. Preparation of the $L^{(\pm)}$ matrices

```

% prepare coefficient for LP, and LM matrices
  delta    = dz/(4*k0*dx^2);
  idelta   = 1i*delta;

% form diagonals...
  LP_diag = zeros(NX,1);
  LM_diag = zeros(NX,1);
  LM_diag(1:NX) = 1 + 2*idelta;
  LP_diag(1:NX) = 1 - 2*idelta;

% prepare array for upper and lower diagonals
  offdiag = ones(NX-1,1)*idelta;

```

```
% insert everything into sparse matrices:
  LM  =   sparse(1:NX-1,2:NX,   -offdiag,   NX,NX)+...
          sparse(1:NX,  1:NX,    LM_diag,   NX,NX)+...
          sparse(2:NX,  1:NX-1,-offdiag,   NX,NX);

  LP  =   sparse(1:NX-1,2:NX,   +offdiag,   NX,NX)+...
          sparse(1:NX,  1:NX,    LP_diag,   NX,NX)+...
          sparse(2:NX,  1:NX-1,+offdiag,   NX,NX);
```

The second modification lies in the main loop, where one needs to modify what becomes the rght-hand-side of the linear system to solve:

Listing 4.2. Main loop in the C-N method

```
%% LM and LP below are matrices/operators that define C-N method

% start form the initial condition
Eold = E0;

% outer loop is for periodic visualization
for k=1:Kmax
  for m=1:M
    B = LP*Eold;   % obtain rhs = LM Eold
    Enew = LM\B;   % solution to  LM Enew = B
    Eold = Enew;   % boundary guard may be applied here
  end

  % visualization
  Observer_Report
end
```

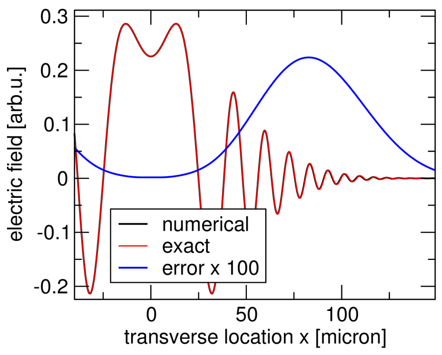
Instructor‘s solution is provided in files *Method.m* and *Main.m*.

Task 2: Validation against an analytic solution
Simulate propagation of a (one-dimensional) Gaussian beam and validate the result against the analytic formula. Select simulation parameters such that a diverging solution is obtained and examine deviations between the exact and numerical solution for increasing rate of beam divergence. You should be able to demonstrate that the error:

- responds to the integration step size Δz
- responds to the grid spacing Δx
- is most discernible in the peripheral sectors of the solution

Solution:

In order to enable an easy qualitative comparison with the performance of the implicit method from the previous section, the same numerical experiment is executed here. A narrow-waist Gaussian beam is let to diffract over a sufficient distance to observe significant evolution of the beam profile, and is then compared to the exact solution:



Numerical beam propagation result versus an analytic formula. An initially collimated Gaussian beam (1D) with a beam waist of 5 micron and wavelength $\lambda = 800$ nm, propagated over a distance of 500 micron. The real part of the field envelope is shown as a black (numerical) and red (exact) line. The two solutions are barely distinguishable. The error, $|E_{num} - E_{ana}|$, is of the order of 10^{-3}.

Comparison with the similar figure in Sect. XXX clearly suggest that the Crank-Nicolson method is significantly more accurate. The error is still largest in the off-axis region, but it is strongly suppressed.

The reader is invited to experiment with this simulation parameters to find out what it takes to obtain accurate solution, and for what parameters (mainly Δx and Δz) the result starts to deteriorate.

Instructor‘s test is in Test.m.

Task 3: Verification of the accuracy order

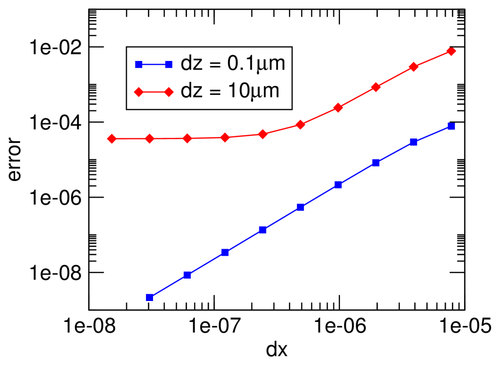
Design a method to verify the order of accuracy of the C-N method with respect to the spatial grid resolution. In a nutshell, a numerical solution should be generated, and an error with respect to a known exact solution be evaluated. One can plot the latter against the resolution on a log-log scale, and read the slope of the resulting curve.

Solution:

This exercise can be executed easily by wrapping the previous simulation in a an outer loop that scans a range of grid spacings, and records the maximum of the error. For each fixed grid spacing Δx, one starts from an initial condition given by an analytic formula for a one-dimensional Gaussian beam. The propagated numerical solution is compared with the Gaussian beam formula evaluated for the same propagation distance and the the error, defined as the maximum of the deviation between the two, is found.

Since the method is second order accurate with respect to the grid spacing, and we expect a power-law decrease of the error with the improving resolution, it is convenient to divide the grid spacing by the same factor, say two, at each iteration. The results, depicted on the log-log scale as in the following figure, should exhibit a straight-line portion with a slope corresponding to the accuracy order of the method.

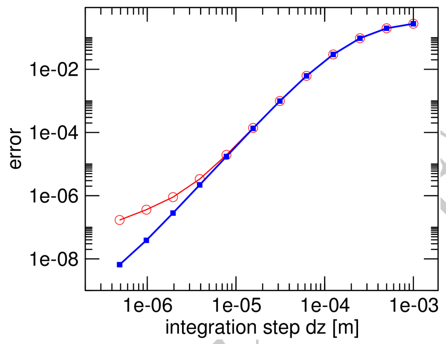
Recall that the statement about the order of accuracy concerns the local error. This is why the numerical solution should only be propagated for a short distance, but long enough for the solution to change significantly. In this illustration, the initial condition is taken to be a Gaussian beam with a waist of 5 micron, at a distance of 1 mm from its focus. This gives a beam profile with enough evolving structure to compare against the exact result. The numerical propagation distance is chosen to be just one propagation step. This way the error is dominated by the local error of the numerical method. The following figure shows an example of this accuracy test result:



Convergence study of a numerical beam propagation method. The data show how the error, which is defined as the maximum deviation between the numerical and analytic solutions, depends on the spatial grid spacing, Δx, and on the integration step, Δz. When, and only when the grid spacing is the quantity that limits the accuracy, the slope of the curve in this log-log plot corresponds to the order of accuracy of the discretization method, which is two in this case.

The picture shows that there indeed is a region in which the error decreases as Δx^2, which shows as a slope-two straight line in the log-log plot. When checking the implementation of a numerical method in a convergence study like this, it is important to realize that there are in general more factors that can limit the accuracy achieved in the numerical solution. First of all, the order of accuracy statement is about the *asymptotic* behavior of the error, meaning that Δx must be small. Indeed, a small deviation from the straight line are visible for large grid spacing. Second, the minimal error can be controlled by both the grid spacing and/or the integration step. If the integration step is large, the error will decrease with Δx initially, but it will level-off when the influence of Δz becomes dominant. This behavior is evident in the upper line shown in the figure. Thus, in general there is only a "window" in which the error behavior given by the accuracy order is unambiguously visible.

Very similar behavior can be seen for the convergence with respect to the integration step Δz. Since the Crank-Nicolson method is second order accurate in the evolution step, one expects to obtain a line with a slope of two in the log-log scale. Deviation form this ideal behavior are expected both for very long and very short steps. However, exactly what 'short' or 'long' means depends on the other parameters of the numerical simulation as is illustrated in this figure:



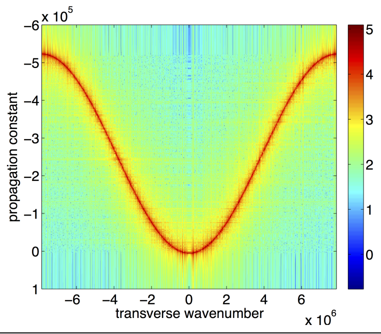
Convergence of the Crank-Nicolson beam propagation method with respect to the integration step. The slope of the black line corresponds to the second order accuracy of this method. The red line and symbols are obtained for a coarser grid, showing a saturation of accuracy for shortest integration steps, when the error is limited by the spatial grid resolution.

4.2.2 Measurement of the numerical dispersion

Once again, the method for measurement of the dispersion curve based on the numerical propagation of a noisy solution can be applied here. The result should be compared to the similar numerical experiment with the implicit discretization of beam propagation equation.

Task 1:
Modify the program used previously to measure the dispersion curve of the implicit beam propagation method, and use it to obtain a map of the numerical dispersion curve of the Crank-Nicolson based BPM.

Solution:
To enable better comparison, this dispersion experiment should be run with the same parameters as in the practice package devoted to the implicit method. The program modification is minimal, and hardly needs discussion. The result is shown in the following figure:



Numerical dispersion curve of the beam propagation method using the Crank-Nicolson method. This map shows the spectral energy concentration obtained for evolution of what was a white-noise initial condition. It is essentially a two-dimensional spatial spectrum of the solution $A(x,z)$ with $A(z, z = 0)$ being a random noise initial condition.

First of all, by the virtue of passing this test, it was verified that the Crank-Nicolson based BPM is indeed stable, at least for the given propagation step. But this is not unexpected. The two things that should be noted in the above result is that, first, the curve is well-defined for all transverse wavenumbers and, second, the curve is sharper than it its analogue for the implicit method. The first fact attests that there is no artificial damping in the C-N method. Energy of all waves, no matter how large transverse wavenumber the might have, is conserved. This means that the waves would have infinite lifetime in an infinite domain, and this is also because that curve appears sharper than in the implicit method where all waves suffer from numerical damping.

Task 2:
The above dispersion measurement method is great for the visualization, but it is obvious that to hone its point-by-point accuracy would require extra effort. Design a more direct and quantitative method to obtain the dispersion curve point by point. Practice package dealing with the numerical dispersion in the Maxwell solver can provide useful inspiration for this exercise, which is left to the reader.

4.3 Crank-Nicolson method for axially symmetric beam propagation

4.3.1 Treatment of the coordinate singularity

The one-dimensional C-N method can be easily modified for two-dimensional beam propagation with axial symmetry (i.e. for situations that can also be handled by DHT-based FFT discussed earlier in this course). What needs a change is the discrete Laplacian operator. Its radially symmetric portion contains a *coordinate singularity* at $r = 0$:

$$\Delta E = \partial_{rr} E + \frac{1}{r}\partial_r E \ . \tag{4.22}$$

Away from $r = 0$, one can still use the following, second-order accurate finite-difference operator (see Section 3.2):

$$\Delta_{ji} E_j^n \equiv E_{j-1}^n - 2E_j^n + E_{j+1}^n + \frac{1}{2j}(E_{j+1}^n - E_{j-1}^n) \quad j > 0 \ , \tag{4.23}$$

The point in the center of the computational domain needs a special treatment, though. By pointing out the coordinate nature of the singularity we mean to say that it only exist due to what is locally unfortunate selection of coordinates. So the the numerical method should be designed in a way that "cleanly removes" this difficulty.

In some texts, one can encounter a less than ideal solution of this issue. Namely, one could argue that because of the radial symmetry of the solution, which we assume here to be the case, one can impose the condition $\partial_r E|_{r=0} = 0$, and use a discrete version of this constraint instead of the propagation equation for $r = 0$. This does work, but it shoudl be avoided because the outcome is an unnecessary loss of one order of accuracy in the vicinity of $r = 0$.

A better way is to treat the offending second term in the above expression (4.22) by the l'Hospital rule,

$$\frac{1}{r}\partial_r E \to \frac{\partial_{rr} E}{1} = \partial_{rr} E \ ,$$

so that the full discrete Laplacian becomes equal to $2\partial_{rr} E$ at $r = 0$, which is not singular anymore and can be discretized.

Yet another correct (and equivalent) alternative is to switch to Cartesian coordinates for $r = 0$ only, and then use the radial symmetry assumption. The discrete-grid realization utilizing a three-point scheme for each coordinate direction reads

$$\Delta E(x,y) = \partial_{xx} E + \partial_{yy} E \tag{4.24}$$

$$\approx \frac{E(-\Delta x,0) - 2E(0,0) + E(+\Delta x,0) + E(0,-\Delta x) - 2E(0,0) + E(0,+\Delta x)}{\Delta x^2} \tag{4.25}$$

$$= \frac{4E(+\Delta x,0) - 4E(0,0)}{\Delta x^2} \tag{4.26}$$

$$= \frac{4E_0 - 4E_1}{\Delta x^2} = \frac{4E_0 - 4E_1}{\Delta r^2} \tag{4.27}$$

Note that this approach remains valid for any choice of the stencil used in a single dimension. These modifications of the Laplacian operator discretization change the matrix Δ_{ij} as follows

$$\Delta_{ij} E_j^n = \begin{pmatrix} -4 & 4 & 0 & \cdots & 0 \\ \ddots & \ddots & \ddots & 0 & 0 \\ 0 & u_j & -2 & v_j & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ \cdots & \cdots & 0 & u_{N-1} & -2 \end{pmatrix} \begin{pmatrix} E_0^n \\ \vdots \\ E_j^n \\ \vdots \\ E_{N-1}^n \end{pmatrix} \tag{4.28}$$

where this time the off-diagonal terms are non-trivial, i.e. dependent on the location in the domain, $u_j = 1 - 1/(2j)$ and $v_j = 1 + 1/(2j)$, as they include contributions from the $1/r\partial_r$ portion of the Laplacian.

The rest of the method, and its solution remains exactly as in the one-dimensional case. It is therefore practical to implement the CN-based BPM in a way that separates the initialization of the linear system from its solution, so that parts of the code can be shared between the radial- and linear-domain methods.

4.3.2 Taking advantage of the tri-diagonal matrix structure

In the first numerical implementation of the finite-difference beam propagation we utilized a linear solver working with a sparse matrix. The structure of this matrix could have been arbitrary, and the solver would still come up with a solution. However, behind the scenes the Matlab solver probably recognized that that the matrix only had non-zero elements around the main diagonal. Because the numerical solution of such a linear system is both simpler and much faster than that of a general problem with an arbitrary matrix, in an actual BPM implementation, the programmer would make certain that an appropriate algorithm is used. Nowadays, one hardly ever needs or should program "home-written" linear algebra programs. Much better job than a casual implementator could do in a reasonable time has already been done and "canned" in many available linear algebra libraries. Nevertheless, the so-called Thomas algorithm should be regarded as an exception to this rule, and especially so in the beam-propagation method context. There are two reasons for this. The first is that writing and using ones own solver is so simple that it is probably easier than finding out how exactly to use a canned library routine. Second, the linear problems with three (or a few) non-zero diagonals arising in BPM methods have the special property that the numerical solution can be further simplified and thus made even faster. Specifically, there is no need to to look for the optimal matrix element to use as a pivot in the Gauss elimination process, and one does not need to guard against divisions by zero, which is something a general routine design could not risk.

Before going further, it should be emphasized that the utilizty of the Thomas algorithm is by no means tied to the radially symmetric problems treated in this Section; we merely take advantage of the appropriate point in this course‘s track to start using tri-diagonal linear solvers. In practice, many BPM approaches are specifically designed such that the whole solution is reduced to a possibly large series of independent tri-diagonal sub-steps. In other words, the simple modification of the realization of the Crank-Nicolson method discussed in the following Practice-track section is of universal importance.

4.4 Practice track: BPM with a tri-diagonal linear solver

Summary: This practice-track package departs from the 1D method implemented in the previous section, and illustrates the following:

- Modification for the radially symmetric beam solution.
- Validation against exact solutions, comparison with the Discrete Hankel Transform method.
- Solution with the help of the tri-diagonal matrix solver (Thomas algorithm).
- Packaging of the one-dimensional C-N solver for usage in more complex algorithms.

4.4.1 Thomas algorithm and radially symmetric, finite-difference BPM

Task 1:
Implement a solver for a linear system of equations with the left-hand-side given by a matrix that only has non-zero elements on the main diagonal and on the upper and lower sub-diagonal. Assume that the solver will only be used for problems that arise in the BPM context, with the implication that the Gauss elimination forward step does not need to check for non-zero pivot. Your goal is a fast, though specialized solver.

Solution:

The very first choise to make when implementing the solver is to decide how the input values will be passed. The most practical way is to pass to the solver function the size of the matrix, plus arrays (as pointers in C) that carry the lower, main, and upper diagonal. Of course, anther array is needed for the right-hand-side vector, and one more to store the solution.

The algorithm is a simplified bersion of the Gauss elimination. An example of how it could be written (in C) is shown next:

Listing 4.3. Thomas algorithm

```
void TDMsolver(int n,complex *a,complex *b,complex *c,complex *v,complex *x)
{
   /*
    * n - number of equations
    * a - lower-diagonal -- indices used = 1...n-1
    * b - the main diagonal
    * c - upper-diagonal -- indices used = 0...n-2
    * v - right hand side vector
    * x - solution holder
    * NOTE: array b will be DESTROYED!!!
    */

    /* Gauss elimination of the sub-diagonal elements */
    for (int i = 1; i < n; i++)
      {
       comple m = a[i]/b[i-1]; // no need to guard against b[i-1]=0
       b[i] = b[i] - m*c[i-1];
       v[i] = v[i] - m*v[i-1];
      }
20
21          /* Backward substitution */
22          x[n-1] = v[n-1]/b[n-1];
23
24          for (int i = n - 2; i >= 0; i--)
25          x[i]=(v[i]-c[i]*x[i+1])/b[i];
26 }
```

This listing show a simple program to realize what is the Thomas algorithm to solve a tri-diagonal system of linear algebraic equations. There are two properties that are worthwhile of a note. Fisrt is about the non-vanishing matrix element, $b[i-1]$ in line 16 used to eliminate the sub-diagonal value in line i. Normally, whenever one divides by a value, a check that it is not zero is in order. Fortunately, here we do not need to do such a check. The result is a faster running code, which the compiler optimzer has a better chance to improve in terms of performance. This is of utmost importance, because the algorithm shown above may be called hundreds of times during each propagation step in a beam propagation method, such as alternate direction implicit method.

But how do we actually know that there will be no division by zero in line 16? Clearly, it can not be true for a general tri-diagonal system. However, for a BPM application we can 'predict' this from what we know about physics of diffraction. Let us assume that a zero occurs at $b[i]$. It happens during the elimination of the sub-diagonal element in the same matrix row. As a result there is only one non-zero value in this particular row after this step, and that will imply that $x[i] = 0$. From here, one could start the backward substitution and evaluate all $x[j], j < i$. It would also mean that the solution vector splits at position i into two sections that "do not talk to each other," and can be caclulated independently. This is, however, unphysical because in a homogeneous material diffraction *must* communicate information across the point i from left to right and in the opposite direction. Thus, for a diffraction problem type, one does not expect to encounter a situtaion in which $b[i-1]$ vanishes.

Another important property of the Thomas algorithm is its stability. It means that numerical noise does not grow during the solution even if the size of the problem, i.e. length of the vector x is large. I other words, this simple algorithm can be applied in situtaions requiring large computational domains.

The favorable properties of this algorithm can be shown shown rigorously for the so-called diagonal dominant systems, in which

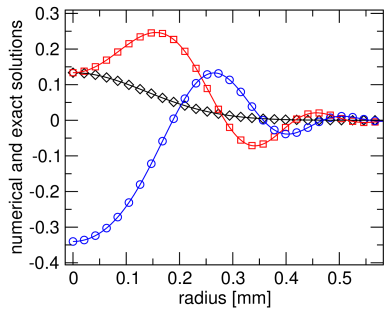
$$|b[i]| \geq |a[i]| + |c[i]| \; .$$

This constraint is just satisfied in a one-dimensional diffraction problem.

Task 2:
Implement a CN-based BPM simulator for radially symmetric solutions. Try to come up with a re-usable design, producing an object that will be utilized in the subsequent practice seesions of this course.

Solution:
The instructor's solution, written in C++, is placed in file *cn-bpm-radial.cc.* The reader is encouraged to utilize this program for inspiration, but write her/his own, perhaps in a different language. Since this and the one-dimensional Crank-Nicolson algortithm will be re-used multiple times during this course, the reader should atempt to design the CN solver as an object that encapsulates/hides all auxiliary variables and can be used as a bulding block for more complex algorithms.

Task 3: Validate your simulator in a comparison with the exact solution of a Gaussian beam. The solution to this task is completely analogous to a similar task for one-dimensional beam propagation practice session. The only difference lies in using the analytic formula for a Gaussian beam in two transverse dimensions. This should be a straightforward exercise and is left to the reader. The following figure can be used as a guide for what this exercise should produce in terms of numerical-vs-exact solution comparison



Testing the finite-difference BPM implementation. Initially collimated Gaussian beam was numerically propagated over a distance corresponding to several Rayleigh ranges — a length sufficient to produce significant change in the beam profile, so that the comparison is non-trivial. The numerical (symbols) solution is then plotted against the analytic (lines). One should plot both the real and imaginary parts of the amplitude togetehr with its modulus. The two kinds of solution should appear essentially identical on the scale of the figure.

4.4.2 Comparison of finite-difference and spectral methods

Summary:
This practice-track section compares two very different methods of numerical beam propagation in radially symmetric geometry

- Double-coding is an effective way to discover not only bugs, but also algorithmic problems in simulation codes. Here we compare results from *different algorithms*, which is a very strong test. It is practically impossible for the two methods used here to be implemented incorrectly yet giving the same numerical results.
- The physical context chosen is once again that of the Poisson‘s bright spot. This problem is a useful stress test for virtually all aspects of the given algorithm (resolution and numerical dispersion issues, stepping procedure, absorbing boundary guard implementation, etc)
- This exercise is an opportunity for the reader to hone intuition that often guides important design decisions in scientific computation practice. In particular, here one can get a good sense for how DHT-based and Crank-Nicolson based BPM compare in terms of numerical effort required to obtain similar-quality data.

The context chosen for this is the simulation of the Poisson‘s bright spot. The reason it is a good stress test for a beam propagation algorithm were appreciated in the earlier exercises. Here, we take advantage of the fact that a very simple geometry results in richly structured beam solutions which are ideally suitable for comparison of beam propagation methods.

Very often, especially at a beginning of a project that deals with development of simulation software, one has to ask: “Which method is better?” Of course, “better” means nothing without specification of what one needs to achieve. The proper question to ask would be, which method, and in our case we compare DHT-BPM versus the Crank-Nicolson based radial finite-difference method, is more suitable to produce a solution of the given problem (radial intensity profile behind the circular obstacle, in this case) with a given accuracy? This is the point of this practice track package.

Perhaps a good point to start from is to try to guess which of the two approaches will be faster to produce an accurate numerical solution. The answer may not be obvious. On one hand, DHT is well suited to this task since it is a spectral algorithm and it is free of numerical dispersion issues. On the other hand, Crank-Nicolson method has a more favorable complexity scaling; with the effort being proportional to the number N of grid points populating the computational domain. In theory it must outperform the DHT algorithm, complexity of which scales with the number of grid points squared. However, the outcome will depend on whether or not the N is sufficiently large for the complexity scaling to determine the answer...

Besides the verification of the correctness of the two codes, their perfomance comparison is an important aspect of this exercise. To compare apples to apples, one should select the problem in a way that actually make the computation speed comparison meaningfull:

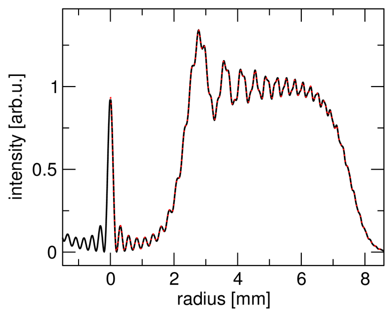
- To minimize the number of possible issues affecting the quality of the simulated beams, no boundary guard will be used. This also means that long propagation distances can not be simulated (Why?).
- To minimize the effects of high spatial frequencies (which behave in very different ways in the two method being compared), the initial condition is set up with a smooth screen edge. This is potentially dangerous, and one should check that important features of the solution are not affected.
- The Crank-Nicolson method is paraxial, while DHT-BPM can simulate both paraxial and non-paraxial regimes. To make the comparison quantitative, the paraxial propagator will be used in the DHT method.

Task 1:

Set up two simulations, one using DHT-BPM and the other working with CN-BPM algorithm, to simulate the radial intensity profile behind an opaque circular obstacle illuminated by a collimated super-gaussian beam. Assume that the simulation results must represent the field with some reasonable sampling density up to certain maximal propagation distance. This means that the DHT method must be executed not in a single jump, but as a series of shorter propagation steps. Adjust this to what you deem to be a reasonable sampling along z, and compare running times in the two methods. Demonstrate that the calculated intensity profile agree with each other.

Solution:

Two instructor's solution programs have been set up in this directory that are implementing DHT-BPM (*p2DHT.m* and supporting sources), and C-N BPM (*cn-bpm-radial.cc*), respectively. Parameters in both are hard-coded such that a particular scenario is simulated with good quality, and a nice agreement between the results is demonstrated between the two methods. The following picture illustrates that the radial intensity of the propagated beams agree very well:



Numerical comparison and verification of two different numerical methods. Initially collimated super-gaussian beam diffracts around a circular screen, and the resulting radial intensity profile is simulated. The red dotted curve corresponds to the Crank-Nicolson method, and the black to the DHT-BPM (with data extended symmetrically to negative radii). On the scale of this figure, the agreement is perfect as one should expect.

Task 2:

Explore the effects of non-physical parameter variations departing from the instructor‘s solution template. The goal is to get a feeling about what is a still acceptable grid resolution, an appropriate integration step, and how do they affect running times, etc. This is an exploratory, open-ended exercise left to the reader.

Task 3:

Next, modify the smooth-edge initial condition (or fuzzy screen edge) in the Crank-Nicolson program to one with a sharp edge. Observe the consequences and make a few attempts to eliminate or minimize numerical artifacts that are caused in the solution by the sharp initial condition. Discuss the origin of these artifacts, and think about how would the *implicit* Euler method behave in these circumstances.

Solution:

Instructor‘s solution has been placed in the sub-directory *Task3.* This first figure shows result comparison between the base run (which uses a smooth initial condition) , and a simulation starting from an initial beam inident on a circular screen with a sharp edge.

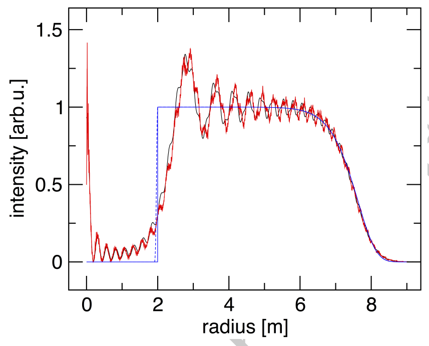
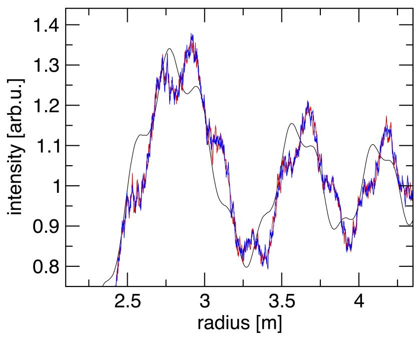


Illustration of the influence the high transverse wavenumbers, present in the initial condition, have on the propagated beam. The initial condition for the modified simulation run is shown as solid blue line. The base-run initial condition is shown in dashed and is barely distinguishable from the former on the scale of this fugure. Note that the only difference between the two is the sharp edge at $r = 2$ mm. The black and red lines represent the propagated beam intensity profieles for the fuzzy and sharp obstacle edge.

There are two kinds of differences between the base run and the new simulation with an initial condition with a sharp screen edge. The first is a “phase shift” between the beam intensity modulations. One can assume that this likely physical and correct, reflecting what is a slightly different effective size of the screen. The second difference is the appearance of high spatial frequencies in the new solution.

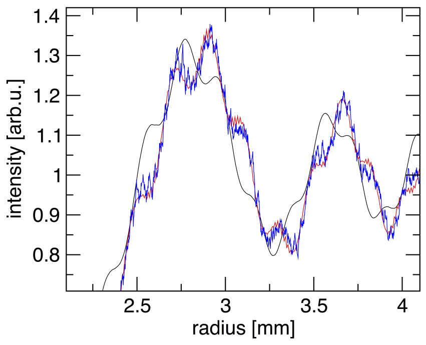
The waves with high spatial wavenumbers are naturally present in the initial condition with a sharp feature, while they are much more damped in the situation with a fuzzy screen. However, while the C-N method does not amplify them, and also does not damp these waves, these waves do have wrong dispersion properties. The consequence o this is that they diffract "out of sync" with other components of the beam's spatial spectrum. This is why even after a short propagation distance they appear as a "noise" superimposed on the background of a smooth solution.

The question is if it is possible to improve things by selecting better or coarser spatial grid resolution or perhaps shorter integration step. Let us look at the effect of a shorter integration step first:



Comparison of two simulations (red and green lines) with different integration steps. It is hardly possible to tell which represents the run with the finer integration step. This means that the integration step is not the parameter that "causes" the artifacts. The black curve shows the result of the base run for the reference.

Clearly, for this particular regime, decreasing Δz did not help; the strength of the noise is roughly the same. So, in the following picture we try a run with a coarser spatial grid:



The solution shown in red is the one obtained on a coarser grid. It is evident that the amplitude of the unwanted noise diminished, though the artifacts were not eliminated completely.

Comparison with the previous "noisy" solution shows that that high spatial frequency artifacts are somewhat less visible, but the result is certainly far from optimal. Decreasing spatial resolution even further is not good either, as the sampling of physical features present in the beam profile is already quite coarse. Moreover, it should not be difficult to realize that the only reason for less noise is that the waves with highest transverse wavenumbers have been removed from the picture. The behavior of other waves actually did not improve at all. This is therefore not really a viable way to deal with the problem.

In summary, the take-home lesson concerning the C-N approach is this:

- Spatially sharp features in the initial condition act a source of numerical noise which does not go away during simulations

- Decreasing the integration step does not completely eliminate the problem
- These artifacts are less severe on coarser grids, but can not be easily removed.

The numerical experimentation with the simulation in this package suggests that there is something "wrong" with the numerical waves that have large transverse wavenumbers, or equivalently, only encompass a small number of grid points per transverse wavelength. Of course, this issue should be familiar to us from the exploration of the Maxwell-solver properties in the first Chapter; in any finite-difference based approach, the extreme waves with fastest oscillation can not propagate properly. In fact, the waves with maximal possible wavenumber on a discrete grid exhibit plus-minus oscillations and as such they cannot distinguish between right and left propagation direction and that is why they do not propagate at all. Once they are created within the simulation domain, they will remain there. The remedy is to ensure that too high spatial frequencies are not generated in the first place. In most situations it is possible. In this example, one should work with a fuzzy edge, such such that the blurring is much smaller than one Fresnel zone width, *and* use very high spatial grid resolution so that the screen edge appears smooth on the scale of the grid spacing. Obviously, the penaltyis a singnificntly longer computation.

4.5 Crank-Nicolson method for two transverse dimensions

In this section we will consider beam propagation in two transverse dimensions, and see how the techniques developed so far can be modified for what is in fact the most common geometry for BPM applications. One of the attractive properties in the previously discussed numerical method(s) will be lost; the resulting matrix for the discrete Laplacian operator will not be tri-diagonal any more. Consequently, the solution of the underlying linear system of equations will require a general-purpose algorithm, and in particular one that can handle systems with many degrees of freedom that are represented by sparse matrices. However, the second-order accuracy in all directions, together with the unconditional stability will be preserved. The consequences of increasing the dimension of the simulation domain will be explored in the subsequent Practice track section.

4.5.1 Discretization

Looking back at how one arrives at the discretization of the beam propagation algorithm for a single transverse dimension, we have the following representation of the Crank-Nicolson approach:

$$(1 - i\delta\Delta)E^{n+1} = (1 + i\delta\Delta)E^n \ . \tag{4.29}$$

Without any specification of how is the Laplacian Δ discretized, this equation emphasizes the most important feature, which is that whatever the spatial discretization scheme might be, in the calculation of the z-derivative everything is averaged between the nth and $(n+1)$th integration step.

Concrete grid geometry must be selected to specify Δ and its discretization. Square grid is assumed next, with equal grid spacing Δx in both transverse directions and the Laplacian will be

$$\Delta E = \partial_{xx}E + \partial_{yy}E \ , \tag{4.30}$$

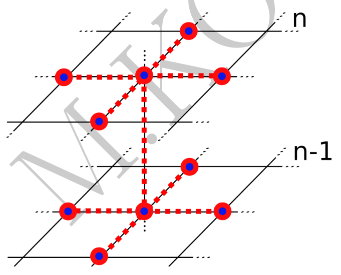
where each term can be approximated by the same symmetric second-order accurate scheme we have used in one dimension. The result is

$$(\Delta E)_{ij} \approx \frac{1}{\Delta x^2}\left[E_{ij-1} - 2Eij + E_{ij+1} + E_{i-1j} - 2E_{ij} + E_{i+1j}\right] \tag{4.31}$$

Note that the above can be also written as

$$(\Delta E)_{ij} \approx \frac{1}{\Delta x^2}\sum_n (E_n - E_c) \ , \tag{4.32}$$

where the sum runs aver all nearest neighbors E_n of the central site E_c. This formula is in fact true for all (isotropic) hyper-cubic lattices. With the same scheme applied to the discretized beam amplitude at the already calculated step n and to the to-be-calculated step $n+1$, the discretization stencil looks like this:



Discretization stencil for the Crank-Nicolson method in two transverse dimensions. The numerical evolution update proceeds from the previously calculated beam-amplitude profile, at the z-integration step labeled $n-1$, to the to-be-calculated amplitude at z-step n. In contrast to the one-dimensional case, the tri-diagonal “shape” of the matrix representing the discrete Laplacian operator is lost. Importantly, this method retains the properties of stability and accuracy.

4.5.2 Numerical solution, direct and iterative solvers

The system of equations describing one integration step is now obtained by inserting the chosen Laplacian discretization into the Crank-Nicolson general template (4.29):

$$E_{ij}^{n+1} - i\delta(E_{ij-1}^{n+1} + E_{ij+1}^{n+1} + E_{i-1j}^{n+1} + E_{i+1j}^{n+1} - 4E_{ij}^{n+1}) = E_{ij}^{n} - i\delta(E_{ij-1}^{n} + E_{ij+1}^{n} + E_{i-1j}^{n} + E_{i+1j}^{n} - 4E_{ij}^{n}) \tag{4.33}$$

where the double index ij runs over all locations in the two-dimensional grid sampling the beam cross-section. Let N be the number of grid points along bot the x-axis and along y-axis. This will simplify our notation, while the generalization for an arbitrary grid should be obvious. In order to solve this linear system, it must be translated into "language" that linear algebra libraries use. In other words, the solution vector is a set of values E_{ij}^{n+1} which we have to order in some way into a linear (one-dimensional) array. This means to map the double index ij into a single index. Once this mapping is chosen, the right hand side of the system can be straightforwardly computed and stored in a vector passed to the linear system solver. The left hand side, of the above system needs to be characterized in terms of its matrix. The concrete representation of the matrix will depend on the mapping $ij \to M(ij)$, and we will look at this in detail in the following Practice-Track section. For now let us assume that we have chosen to map double indices into single index as in

$$M(ij) = i * N + j - N \qquad i, j = 1, 2, \ldots, N \ , \tag{4.34}$$

which assumes that indexing is as in Matlab, i.e. starting from one. This or similar conversion between two-dimensional and linear indices will allow to represent the above system in the form

$$L^{-} E^{n+1} = R \ , \tag{4.35}$$

which will be passed to a solver subroutine to calculate E^{n+1}. Here L^{-} is the matrix of the systems, and R is the right-hand-side. Explicitly,

$$R_{M(ij)} = E_{ij}^{n} - i\delta(E_{ij-1}^{n} + E_{ij+1}^{n} + E_{i-1j}^{n} + E_{i+1j}^{n} - 4E_{ij}^{n}) \quad i, j = 1, \ldots N \tag{4.36}$$

with N standing for the number of grid points in the transverse direction (assumed to be equal for x and y) is vector of length N^2.

For the system matrix, which has the dimension $N^2 \times N^2$, it is enough to enumerate the non-zero elements as follows. The diagonal of the matrix is given by

$$L^{-}_{M(ij),M(ij)} = 1 + 4i\delta \qquad i, j = 1, \ldots N \ . \tag{4.37}$$

The off-diagonal elements are read off from the discretization scheme formula. We have

$$L^{-}_{M(ij),M(ij+1)} = L^{-}_{M(ij),M(ij-1)} = L^{-}_{M(i+1j),M(ij)} = L^{-}_{M(i-1j),M(ij)} = -i\delta \qquad i, j = 1, \ldots N \ . \tag{4.38}$$

Positions in the matrix not enumerated above carry zero values. It is clear that there are about $5 \times N^2$ nonzero elements in this matrix, which is very little in comparison with the total of N^4 of all elements. This is why it is called *sparse* matrix. Typically, matrices that arise in the solution of partial differential equations are sparse, and libraries are available for work with such matrices.

The choice before us now is which algorithm to use for the solution. There are two options, roughly speaking; iterative and direct solvers. The latter derive their name from the fact that they work with the matrix of the system and transform it to a triangular form (or a product of triangular matrices) in the process of the solution. These algorithms are guaranteed to obtain *some* solution (which may suffer from accuracy issues). In contrast, the iterative solvers do not work over the matrix, but use a series of approximate solutions in the process of iteration. In general there is no guarantee that it will converge, which is a major disadvantage of iterative solvers in general. On the other hand, they have a big advantage in that they actually do not need to store the matrix in the computer memory. All that is needed is an algorithm that specifies the results of the matrix multiplying a given vector. As a result the memory usage is limited, and extremely large linear systems can be solved in this way.

Both approaches are in frequent use in the beam propagation modeling, and this course will provide opportunities to practice with direct and iterative solvers to illustrate their pros and cons.

4.5.3 Dispersion, accuracy, stability, and grid-induced anisotropy

Derivation of the Crank-Nicolson dispersion relation for two transverse dimensions is a simple generalization of the one-dimensional procedure outlined previously. Instead for a one-dimensional plane wave one calculates the numerical propagation constant of a wave in two dimensions. Such a plane-wave can be parameterized by two transverse wavenumbers, k_x and k_y as independent variables:

$$\psi = e^{ik_x x} e^{ik_y y} e^{K_z z} = e^{i(k_x x + k_y y + K_z z)} \ , \qquad (4.39)$$

with the third spatial wavenumber $K_z = K_z(k_x, k_y)$ understood as a function that has to be determined so that the above is a solution to the discrete-grid evolution equation realized by the two-dimensional Crank-Nicolson beam propagation method. It is thanks to the translation symmetry that we know that a plane-wave solution must exist in this form, i.e. as a product of three exponentials, each dependent of one spatial dimension. The function, when restricted for x and y to a discrete grid, becomes an eigenfunction of the two-dimensional discrete Laplacian operator Δ, if and only if K_z has the "right value" for given k_x and k_y.

With E^n_{ps} standing for the electric field amplitude obtained at the integration step n, for transverse grid points s and p, we take an ansatz for the plane-wave

$$E^n_{ps} \equiv E(n\Delta z, p\Delta x, s\Delta y) = A e^{iK_z z_n} e^{ik_x x_p} e^{ik_y y_s} \equiv A e^{iK_z z_n} \psi_{ps}(k_x, k_y) \ , \qquad (4.40)$$

and insert it into the discrete propagation scheme. The only non-trivial part is the result of the action of the discrete Laplacian acting and this plane wave. To evaluate it, one has to use double indices to denote a specific location on the computational grid. Calculation of the corresponding eigenvalue is straightforward, yielding

$$\begin{aligned}\sum_{ps} \Delta_{mn,ps}\psi_{ps} &= \psi_{p-1,s} - 2\psi_{ps} + \psi_{p+1,s} + \psi_{p,s-1} - 2\psi_{ps} + \psi_{p,s+1} = \\ &+ (e^{ik_x x_{m-1}} - 2e^{ik_x x_m} + e^{ik_x x_{m+1}})e^{ik_y y_n} + (e^{ik_y y_{n-1}} - 2e^{ik_y y_n} + e^{ik_y y_{n+1}})e^{ik_x x_m} \\ &= [2(\cos k_x\Delta x - 1) + 2(\cos k_y \Delta y - 1)]\psi_{mn} \equiv \lambda(k_x, k_y)\psi_{mn}(k_x, k_y). \qquad (4.41)\end{aligned}$$

As one should expect, eigenvalue $\lambda(k_x, k_y)$ is a sum of eigenvalues corresponding to two orthogonal directions in the grid. The propagation constant K_z corresponding to $\lambda(k_x, k_y)$ is obtained after using the above in the Crank-Nicolson update formula (4.29),

$$(1 - i\delta\Delta)\exp\left[iK_z(k_x, k_y)(\delta z + z_n)\right]\psi(k_x, k_y) = (1 + i\delta\Delta)\exp\left[iK_z(k_x, k_y)z_n\right]\psi(k_x, k_y) \ ,$$

where one can replace each Δ by $\lambda(k_x, k_y)$ because it acts on its own eigen-function:

$$(1 - i\delta\lambda(k_x, k_y))\exp[iK_z(k_x, k_y)(\delta z + z_n)]\psi(k_x, k_y) = (1 + i\delta\lambda(k_x, k_y))\exp[iK_z(k_x, k_y)z_n]\psi(k_x, k_y) \, .$$

From here, one obtains the numerical propagation constant in the form

$$K_z(k_x,k_y) = \frac{-i}{\Delta z} \ln \left[\frac{1 + \frac{i\Delta z}{2k_0 \Delta x^2}(\cos k_x \Delta x - 1) + \frac{i\Delta z}{2k_0 \Delta y^2}(\cos k_y \Delta y - 1)}{1 - \frac{i\Delta z}{2k_0 \Delta x^2}(\cos k_y \Delta x - 1) - \frac{i\Delta z}{2k_0 \Delta y^2}(\cos k_y \Delta y - 1)} \right] \ , \tag{4.42}$$

As always, the continuum-limit sanity check is in order. For a small spatial grid spacing, and small integration step, ($\Delta z \to 0$, $\Delta x \to 0$), it is straightforward to obtain the first non-vanishing term in the Taylor expansion of $K_z(k_x, k_y)$

$$K_z(k_x,k_y) \approx -\frac{1}{2k_0}\left(k_x^2 + k_y^2\right) \ , \tag{4.43}$$

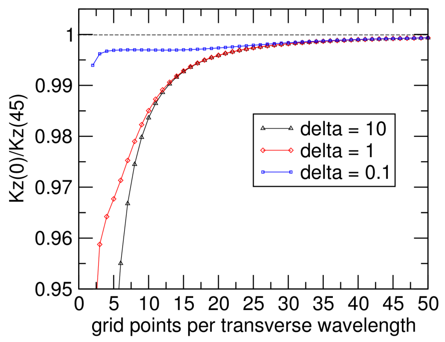
which is the correct propagation constant for a plane wave in the paraxial approximation. It is left to the reader to evaluate the higher-order corrections, and show that the deviation from the continuum limit scale as Δz^2 and/or Δx^2, thus corroborating that the method is second order in both dimensions. This confirms that similar to the situation in one transverse dimension, the 2D method retains the second-order accuracy.

As for the stability, the argument is exactly the same as in one dimension. No matter what values of k_x, k_y one may choose, the eigenvalue $\lambda(k_x, k_y)$ is real, which in turn implies that $\exp(iK_z\Delta z)$ is a ratio of two complex-conjugate expressions and thus have a unit modulus. That is why the norm is preserved for any plane-wave as we step from z_{n-1} to z_n. It is easy to show that on a computational domain with periodic boundary conditions all plane waves are mutually orthogonal, and from there it follows that the propagation scheme is norm-preserving, or unitary, for an arbitrary initial beam. Details of this argument are left to the reader as a useful exercise.

In summary, one finds that the accuracy and stability of the 2D method are completely analogous to those of the one-dimensional method. Now comes the departure brought by the additional dimension, and it is the numerical anisotropy. At this point the reader should recall the discussion of the analogous issue in the context of the Maxwell numerical solver in Section 1.

The formula (4.42) is clearly anisotropic, because an isotropic expression would be some function of $|k| = (k_x^2 + k_y^2)^{1/2}$, which it is not. The grid-induced anisotropy results in different propagation properties, specifically the phase and group velocity, for different relative values of k_x and k_y (i.e. different direction in the transverse plane). Having showed that the continuum limit is correctly captured, we know that in principle the anisotropy artifacts can be controlled by improving the grid resolution and shortening the integration step. But recalling the experience with the Maxwell solver, we also know that the anisotropy shows up more pronounced for solutions that exhibit more variation across the transverse cross-section of the beam. So the question stands what is the number of grid points needed per transverse wavelength in order to reduce the anisotropy to an acceptable level?

The following figure shows the ratio between the propagation constants of waves propagating with transverse wavevectors along coordinate axis and along the grid diagonal, respectively:



Numerical anisotropy caused by the discrete grid in the two-dimensional Crank-Nicolson method using a three-point discretization scheme in each spatial direction. Plots correspond to several values of $\delta = \Delta z/(4k_0\Delta x)$, and the "transverse wavelength" should be understood in general as the length scale over which the transverse beam amplitude changes significantly. Here, for the plane wave solution it is $1/(2\pi k_x)$. The dotted line indicates the ideal result only achieved in the continuum limit.

As immediately evident from the plots that quickly deviate form unity, the anisotropy become visible even for waveforms that are sampled by a relatively large number of grid points. In other words, small Δx **and** small Δz must be used to suppress anisotropy. If one wants to achieve less than one percent anisotropy, a few tens of grid points are needed per a "characteristic transverse length" over which the beam profile changes significantly. This may imply large computational effort. So the message here is that for practical purposes **certain degree of anisotropy must be accepted**. To develop a sense of what it takes to control these effects in simulations, the reader is invited to conduct a guided numerical experiment in the Practice track section that follows...

4.6 Practice track: Finite-difference methods in two dimensions

Summary:

- Computational complexity increases significantly in the two-dimensions: Here we are to appreciate this difficulty with the finite-difference BPM using Crank-Nicolson method.
- Practice construction of sparse matrices to represent discretized differential operators
- Appreciate the advantages and disadvantages of direct linear-system solvers.

4.6.1 Construction of sparse-matrices for differential operators

Task 1: Implementation
Starting from the one-dimensional implementation of Crank-Nicolson-BPM (see Practice-Track package WP08), modify the program for two transverse dimensions. An important aspect of this exercise is to re-use as much as possible from the previous implementation, and thus appreciate the common parts of the algorithm.

Solution:

To reduce the code writing to a minimum, it is probably best to utilize the very first implementation of the Crank-Nicolson based beam propagation as a point of departure for this exercise. In Practice-Track package 08 we defined the operator pair L^+, L^- in terms of sparse matrices, and then the main simulator loop only required a few lines of code. The most significant addition to the code is filling in the sparse matrices representing the L^+ and L^- operators, while the main loop should not require any changes.

The matrix representation of these operators must be based on sparse matrices, because the size of the matrix is significantly larger in two dimension, and a great majority of the matrix elements are zeros. We will look in detail on the construction of a sparse-matrix operator — it is a very common and important task in many contexts.

To keep things simple, Matlab and its sparse-matrix capabilities are used in what follows. In this case it is practical to view the sparse matrix structure as a black box; details of how it is implemented are unimportant. Sparse matrices can be parameterized in several different ways. The simplest one is to provide three lists of numbers specifying the rows, columns, and values of all non-zero elements. The construction of these arrays is the first step.

To simplify the code, and thus reduce the room for errors, one can create first a sparse matrix Δ that represents the discrete Laplacian operator on the given computational grid. If the sparse-matrix library supports matrix algebra, $L^{\pm}$ can be simply derived from Δ.

The computational grid representing the transverse cross-section of the beam is now two-dimensional; indices i and j will be used to specify a grid location along the x and y directions, respectively. It will be assumed that the number N of grid points in both directions is the same. What we need next is a mapping the location in the grid onto an index specifying a column (or a row) in the sparse matrix. In Matlab, the following function realizes such a mapping:

$$\mathrm{M} = @(i,j)\ j * N + i - N$$

It should not be difficult to see that it assigns to (i,j) an index $M(i,j)$ that represents the position in the given grid as counted in the typewriter order. Alternatively, one can recall that there are no "real" matrices in the computer memory. Rather, every array is stored as a linear vector. The above is nothing but a position in such a vector. Note that the definition of this function will be slightly different in different languages. In Matlab, where one starts indexing with one, the first position $(1,1)$ maps to index 1 as it should. In C, the mapping would be $j * N + i$, with $(0,0)$ mapped to 0.

Having specified the correspondence between the sparse matrix index and the computational grid location, we are ready to identify the non-zero matrix elements of the discrete Laplacian operator. Recall that the diagonal elements of this operator (on an isotropic grid) are simply the number of nearest neighbors but with a negative sign. For the non-diagonal elements, we have one $+1$ for each pair of nearest neighbors. In other words, we need

$$\Delta_{ij,ij} \rightarrow \Delta_{M(i,j),M(i,j)} = -4 \quad i,j = 1,\dots,N$$

for the diagonal part of the operator, and

$$\Delta_{ij,ij\pm 1} \rightarrow \Delta_{M(i,j),M(i,j\pm 1)} = +1 \quad i,j = 1,\dots,N$$

for the non-diagonal part originating in partial derivatives w.r.t. y, and finally

$$\Delta_{ij,i\pm 1j} \rightarrow \Delta_{M(i,j),M(i\pm 1,j)} = +1 \quad i,j = 1,\dots,N$$

for the non-diagonal part representing partial derivatives w.r.t. x. From here, it is also easy to see that the size of the matrix Δ is $N^2 \times N^2$, and the number of non-zero elements in it is not greater that $5 \times N^2$. We therefore need three arrays of this length that will hold the column, row, and the value of each non-vanishing element of Δ.

The following listing illustrates the relevant code that constructs the sparse matrix operator for the discrete Laplacian an a square grid:

Listing 4.4. Filling in a sparse matrix

```
% maximal number of non-zero matrix entries
nzmax = N*N*5;

% these arrays will encode sparse matrix of DELTA:
rows = zeros(nzmax,1);  %rows
cols = zeros(nzmax,1);  %columns
valp = zeros(nzmax,1);  %values

% mapping grid location to matrix index
M = @(i,j) j*N + i - N;

% enumerate all non-zero matrix elements
count = 0;   % count how many non-zeros enumerated so far

% loop over each location in the grid
for i=1:N
for j=1:N

% diagonal elements
loc = M(i,j);  % central stencil point location

count = count + 1;  % update count
rows(count) = loc;  % loc is the row index
```

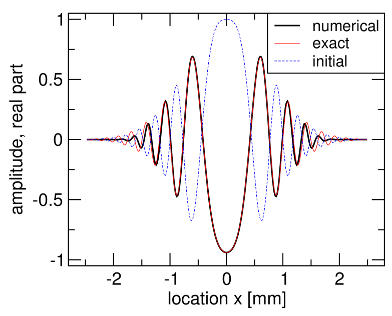
```
cols(count) = loc;  % loc is the column index
valp(count) = -4;   % diagonal element value

% nondiagonal elemens for grid neighbors on the ``right''
if(i<N)  %right neigbor must exist!
  loc = M(i,j);     % central stencil point location
  nnl = M(i+1,j);  % and its nearest neighbor location

  count = count + 1;
  rows(count) = loc;
  cols(count) = nnl;
  valp(count) = 1;  %non-diagonal value
end

% similar code for grid neighbors on ``left''
...
% similar code for grid neighbors ``up''
...
% similar code for grid neighbors ``down''
...
end  % j loop
end  % i loop

% discrete Laplacian operator: insert elements into sparse matrix
DELTA = sparse(rows(1:count),cols(1:count),valp(1:count),N*N,N*N,count);

% auxiliary Kronecked delta (identity) matrix
KD = sparse(1:N*N,1:N*N,1);

% construct Lplus, Lminus
idelta    = 1i*dz/(4*k0*dx^2);

LP =  KD + idelta*DELTA;
LM =  KD - idelta*DELTA;
```

Needless to say, the above could be programmed more efficiently, but this codelet is meant to illustrate the general idea of how sparse-matrix operators are constructed in numerical solutions of partial differential equations. Instructor‘s version for this program is included in XXX.

4.6.2 Issues in 2D: computational complexity increase

Task 2: Testing
Test the implementation of the 2D beam simulator with the analytic Gaussian beam solution. Start with a modest grid size and a not too-tightly focused initial condition in order to make the simulation easier. Make sure to numerically propagate for at least one Rayleigh range so that the beam profile exhibits significant change and thus serves as a basis for a non-trivial test. Reasonable agreement between numerical and analytic solution must be achieved before proceeding to Task 3. While this exercise should be relatively straightforward and is left to the reader, the following figure should give a sense of what kind of agreement between the simulated

and analytic solution one can expect for a "modest" simulation with a relatively coarse grid resolution and a long integration step.



Testing 2D Crank-Nicolson BPM algorithm. Only real part of the complex beam amplitude is shown for a simulation starting from an initial beam (dashed blue) focused to a distance of 0.5 m. Simulated (black) and analytic (red) solutions are shown at a distance of $2f$. The computational grid was 300×300 points, representing an area of 5×5 mm. The integration step was chosen to be 1 cm for this beam with wavelength of 800 nm. Note how the near-perfect agreement in the center deteriorates further from the beam axis.

For further exploration:

i) The above illustrated simulation only requires several seconds to execute. Experiment with the simulation parameters in an attempt to improve the numerical-analytic agreement.

ii) Observe the evolution of the numerical solution and identify the propagation distance when the deviation from the analytic target becomes most obvious. Explain your observation.

iii) This implementation utilized a direct linear-system solver. This means that the matrix of the linear system of equations must be prepared and passed to the library routine that implements the solver. Initially this is a sparse matrix, but as the direct solution proceeds many of the zero elements become non-zero in the transformed matrix. As a result, more memory is needed to store the matrix and this shows up in the memory usage of the program. Readers should observe, during the simulation, how the memory volume in use increases and subsequently fluctuates. This has two unpleasant effects: First, depending on the solver implementation, memory may be repeatedly requested from and released to the system. This is an expensive operation and may be a cause for poor performance when the size of the problem is large. Second, it is in general difficult to know how much memory is sufficient for the given problem because we do not know before hand what will be the filling effect on the sparse matrix representing the linear system. These are just a few reasons why alternatives to direct solvers are also often used, namely iterative solvers which do not require storage for the matrix at all. On the other hand, they also do not guarantee that a solution will be found... We will have an opportunity to experience and compare the advantages and disadvantages of direct and iterative solvers later in this course.

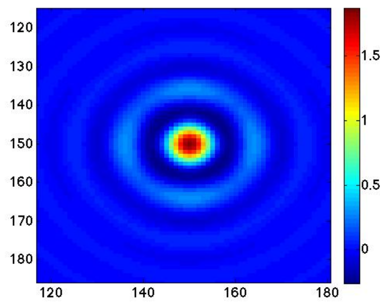
4.6.3 Issues in 2D: grid anisotropy

Task 3: Grid anisotropy manifestation

Set up a simulation with as tight focusing as feasible for a short simulation (experiment with parameters!), and run the evolution until the focal region is reached. Inspect the central lobe of the solution, and observe how the anisotropy of the grid shows up. Try to modify the integration step and/or the grid resolution in order to reduce the observed anisotropy in the beam pattern. During these simulations, note the memory usage and also the CPU utilization if running on a multiprocessor.

Solution:

To observe numerical anisotropy effects in the simulated solution, one needs to "populate" the initial beam profile with waves that have sufficiently high transverse wavenumbers. Such waves show the strongest difference between their propagation constants when the transverse wave-vector points along the grid axis and along the diagonal. On the other hand, the reader should not impose a too tight focusing geometry, because then the simulation needs shorter step and finer transverse resolution otherwise the accuracy suffers. However, it should become evident with little experimentation with the simulation parameters, that anisotropic wave propagation is actually rather easy to see. The following illustration shows data obtained on a grid of 300×300 points, with the computational domain size of 5×5 mm, for a beam with the wavelength 800 nm, and beam waist of 1 mm. The focal length was chosen $f = 0.4$ and the propagation distance was the same.



Center of a simulated beam (real part of the amplitude) close to its focus. Note the variation of the intensity evident in the rings. The pattern has a four-fold symmetry inherited from the grid-symmetry. The origin of this artifact is that the phase velocity of the wave is different when propagating along the grid axis or along the grid diagonal.

This picture demonstrates that numerical anisotropy shows up readily. Readers should try several sets of input parameters and attempt to reduce the anisotropy of the numerical solution — it will become evident very soon that it is not an easy (or numerically inexpensive) task. In fact, this simulation is an example of how a radially symmetric problem *should not* be solved on a rectangular grid.

Take-away lessons:

- two dimensional beam propagation becomes significantly more demanding, computationally, than the problems in one transverse dimension explored in previous sections
- the idea of the Crank-Nicolson method remains the same in two and in higher dimensions, and the programming is not much different or more complicated if a direct linear solver library routine can be used
- even a small problem may require significant memory with the direct linear solver, because the originally very sparse system matrix fills up with many more non-zero elements during the course of the solution
- numerical dispersion and accuracy issues become significantly more severe, because simulations are in general executed with lesser resolutions and longer integration steps than in "smaller," one-dimensional problems.
- numerical grid anisotropy shows up readily in the simulated beam solutions