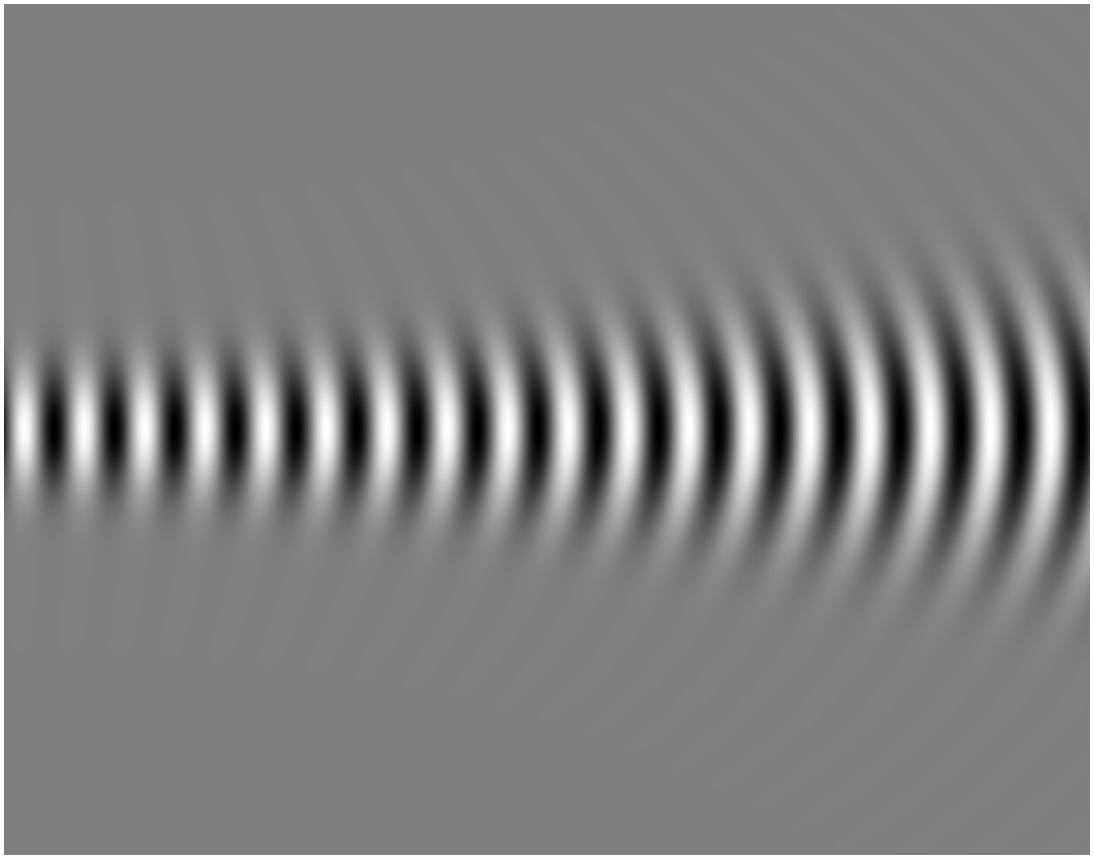
Physics 325 Final Project

Henry Schnieders
University of Wisconsin–Madison

May 8, 2025

Abstract

This project is about solving the 1D paraxial wave equation using various finite difference methods. method. The solution obtained by numerically integrating the Rayleigh-Sommerfeld integral under the appropriate assumptions is also presented. All solutions are programmed using Python.



Origin of the paraxial wave equation

Consider an empty region of free space. Maxwell's Equations reduce to

$$\nabla\cdot\mathbf{E} = 0, \qquad \nabla\times\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}, \tag{1}$$

$$\nabla\cdot\mathbf{B} = 0, \qquad \nabla\times\mathbf{B} = \frac{1}{c^2}\,\frac{\partial\mathbf{E}}{\partial t}. \tag{2}$$

Wave equation for the electric field.

$$\nabla\times\big(\nabla\times\mathbf{E}\big) = -\frac{\partial}{\partial t}\big(\nabla\times\mathbf{B}\big), \tag{3}$$

$$\nabla\big(\nabla\cdot\mathbf{E}\big) - \nabla^2\mathbf{E} = -\frac{1}{c^2}\,\frac{\partial^2\mathbf{E}}{\partial t^2}, \tag{4}$$

$$\nabla^2\mathbf{E} - \frac{1}{c^2}\,\frac{\partial^2\mathbf{E}}{\partial t^2} = 0. \tag{5}$$

Monochromatic assumption. Assume harmonic time dependence

$$\mathbf{E}(\mathbf{r},t) = \mathrm{Re}\big\{\tilde{\mathbf{E}}(\mathbf{r})e^{i\omega t}\big\},$$

so that

$$-\frac{1}{c^2}\,\frac{\partial^2\mathbf{E}}{\partial t^2} = k^2\tilde{\mathbf{E}}(\mathbf{r}), \qquad k = \frac{\omega}{c} = \frac{2\pi}{\lambda}.$$

Scalar Helmholtz equation. For a uniformly–polarised, source-free beam (for $z > 0$) may treat the field as a scalar, $E(\mathbf{r})$, obeying

$$\nabla^2 E + k^2 E = 0.$$

Factor out the carrier. Assume the field propagates mainly along the z-axis and write

$$E(\mathbf{r}) = U(x,y,z)\,e^{ikz}.$$

Then

$$\nabla^2 E = \big[\partial_{xx}U + \partial_{yy}U + \partial_{zz}U + 2ik\partial_z U - k^2 U\big]e^{ikz}$$

so the Helmholtz equation becomes

$$\partial_{xx}U + \partial_{yy}U + \partial_{zz}U + 2ik\partial_z U = 0.$$

Slowly-varying-envelope approximation (SVEA). If the envelope varies slowly along z, $\partial_{zz}U \approx 0$, giving the **paraxial wave equation**

$$\boxed{\partial_{xx}U + \partial_{yy}U + 2ik\partial_z U = 0}.$$

1-D transverse case. With only one transverse coordinate x,

$$\boxed{\partial_{xx}U + 2ik\partial_z U = 0}.$$

Integral Form

The general form of the Rayleigh–Sommerfeld integral for a 1D propagating wave is

$$E(\mathbf{r}) = \int_S \tilde{E}(\mathbf{r}') \, \frac{\exp\big(ik|\mathbf{r}-\mathbf{r}'|\big)}{\sqrt{|\mathbf{r}-\mathbf{r}'|}} \left(\frac{\mathbf{r}-\mathbf{r}'}{|\mathbf{r}-\mathbf{r}'|} \cdot \hat{\mathbf{n}} \right) \mathrm{d}S.$$

Here, the obliquity factor is $cos(\theta) = \frac{z}{r}$, so the expression becomes

$$E(\mathbf{r}) = \int_S \tilde{E}(\mathbf{r}') \; z \frac{\exp\big(ik\,|\mathbf{r}-\mathbf{r}'|\big)}{|\mathbf{r}-\mathbf{r}'|^{\frac{3}{2}}} \, \mathrm{d}S.$$

For a 1D Gaussian beam of waist w_0 centered at the origin, the field at the observation point (z, x) becomes

$$E(z,x) = \int_{-w/2}^{w/2} z \exp\left[-\left(\frac{w'}{w_0} \right)^2 \right] \exp\left[ik\sqrt{z^2 + \big(x-w'\big)^2} \right] \frac{\mathrm{d}w'}{|z^2 + \big(x-w'\big)^2|^{\frac{3}{2}}}.$$

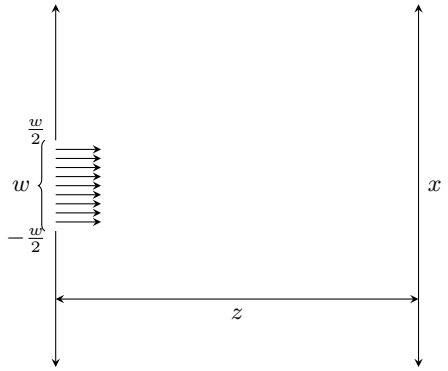
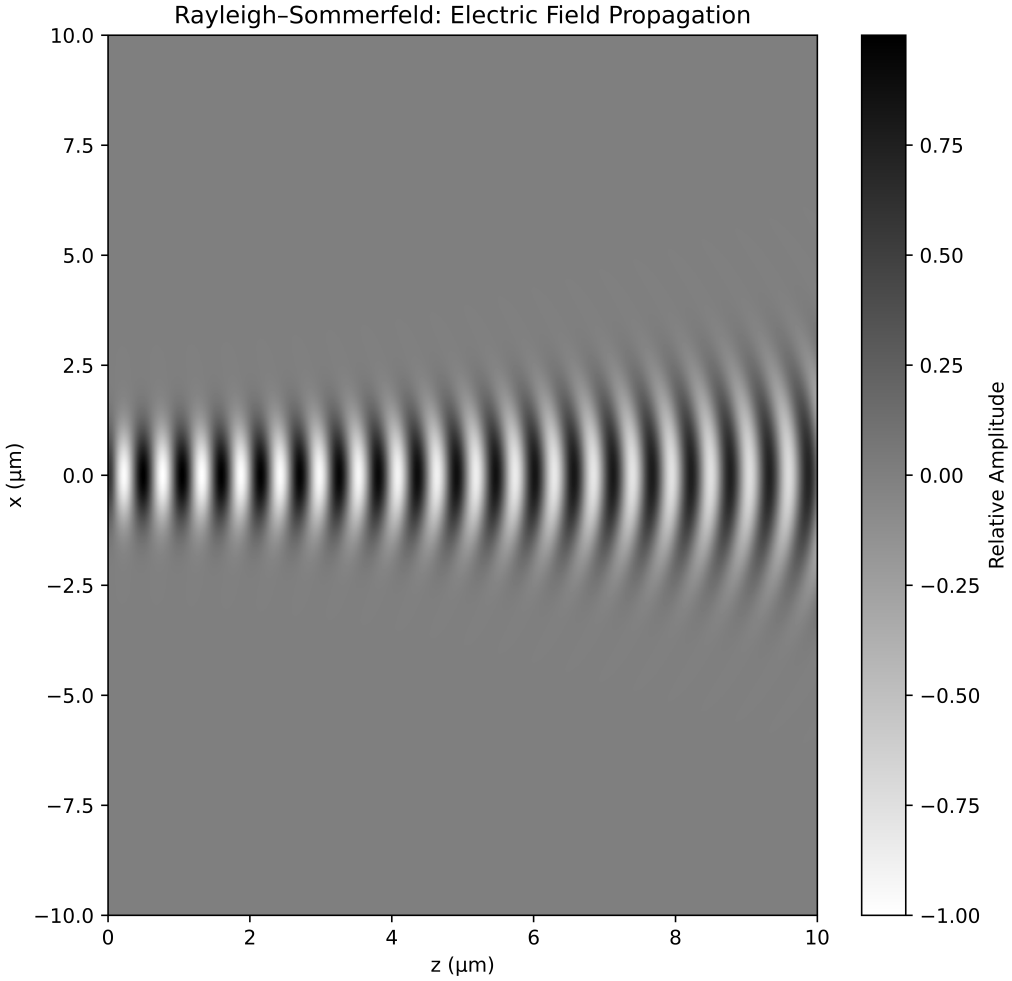


Figure 1: Diagrammatic representation of the solved configuration by the Rayleigh-Sommerfeld Integral and all numerical methods to follow.


Rayleigh–Sommerfeld: Electric Field Propagation
x (μm)
z (μm)
Relative Amplitude
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
2
4
6
8
10
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Numerical solution 1: Leapfrog method

$$\frac{\partial^2 u}{\partial x^2} + 2\,\mathrm{i}k\,\frac{\partial u}{\partial z} = 0, \tag{6}$$

$$\Longrightarrow \frac{\partial^2 u}{\partial x^2} = \frac{2k}{\mathrm{i}}\,\frac{\partial u}{\partial z}. \tag{7}$$

Now discretize both sides with steps $\Delta x, \Delta z$. For the left-hand side use a centered difference in x and for the right-hand side the average of forward/backward differences in z:

$$\frac{1}{\Delta x}\Big[\frac{u(x+\Delta x,z)-u(x,z)}{\Delta x} - \frac{u(x,z)-u(x-\Delta x,z)}{\Delta x}\Big] = \frac{2k}{\mathrm{i}}\,\frac{1}{2}\Big[\frac{u(x,z+\Delta z)-u(x,z)}{\Delta z} + \frac{u(x,z)-u(x,z-\Delta z)}{\Delta z}\Big]. \tag{8}$$

Noting that the left-hand side is

$$\frac{1}{(\Delta x)^2}\Big[u(x+\Delta x,z)+u(x-\Delta x,z)-2u(x,z)\Big]$$

and the right-hand side is

$$\frac{2k}{\mathrm{i}}\,\frac{1}{2\Delta z}\Big[u(x,z+\Delta z)-u(x,z-\Delta z)\Big],$$

we obtain the marching formula

$$u(x,z+\Delta z) = u(x,z-\Delta z) + \frac{\Delta z\,\mathrm{i}}{k\,(\Delta x)^2}\Big[u(x+\Delta x,z)+u(x-\Delta x,z)-2u(x,z)\Big]. \tag{9}$$

Boundary handling: At $z = 0$, $z - \Delta z$ lies outside the domain, so we replace the centered z–derivative by a one-sided "three-point" formula,

$$\left.\frac{\partial u}{\partial z}\right|_{z=0} \approx \frac{1}{3}\big(2\,u(x,0)+u(x,\Delta z)\big). \tag{10}$$

Boundary conditions

For best realistic accuracy, we impose Robin conditions at $x = \pm L$. To approximate the behavior of an outgoing wave, $\alpha = -ik,\ \beta = 1$. In the usual second-difference approximation scheme

$$D_{xx} \approx \frac{1}{(\Delta x)^2}\begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & \ddots \\ & & \ddots & \ddots \end{bmatrix},$$

the top-left entry "-2" must be replaced by a modified value "$*$":

$$D_{xx} \;\longrightarrow\; \frac{1}{(\Delta x)^2}\begin{bmatrix} * & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & \ddots \\ & & \ddots & \ddots \end{bmatrix}.$$

Here $*$ is found from the Robin condition, which is used to approximate $u(-L-\Delta x), u(L+\Delta x)$ that fall outside of the domain.

$$\alpha \frac{\partial u}{\partial x} + \beta\, u = 0, \tag{11}$$

Using $u(L+\Delta x)$ as an example,

$$\alpha\left[\frac{u\big(L+\Delta x, z\big) - u\big(L, z\big)}{\Delta x}\right] + \beta\, u\big(L+\Delta x, z\big) = 0,$$

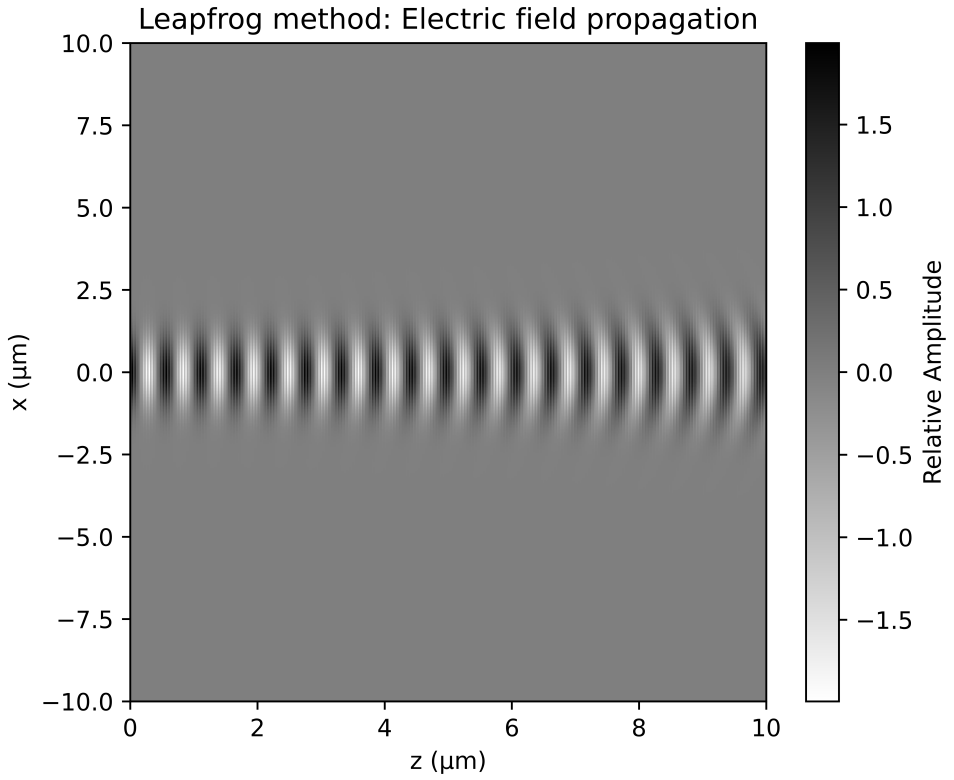
$$\left(\beta + \frac{\alpha}{\Delta x}\right) u\big(L+\Delta x, z\big) = \frac{\alpha}{\Delta x}\, u\big(L, z\big).$$

Thus

$$u(L+\Delta x, z) = \frac{\frac{\alpha}{\Delta x}}{\frac{\alpha}{\Delta x} + \beta}\, u(L).$$

so that the new entry is

$$* \;=\; \frac{\frac{\alpha}{\Delta x}}{\frac{\alpha}{\Delta x} + \beta} \;-\; 2. \tag{12}$$



This method is numerically unstable for the paraxial wave equation. For smaller step sizes, the solution diverges.

Numerical solution 2: Forward Euler

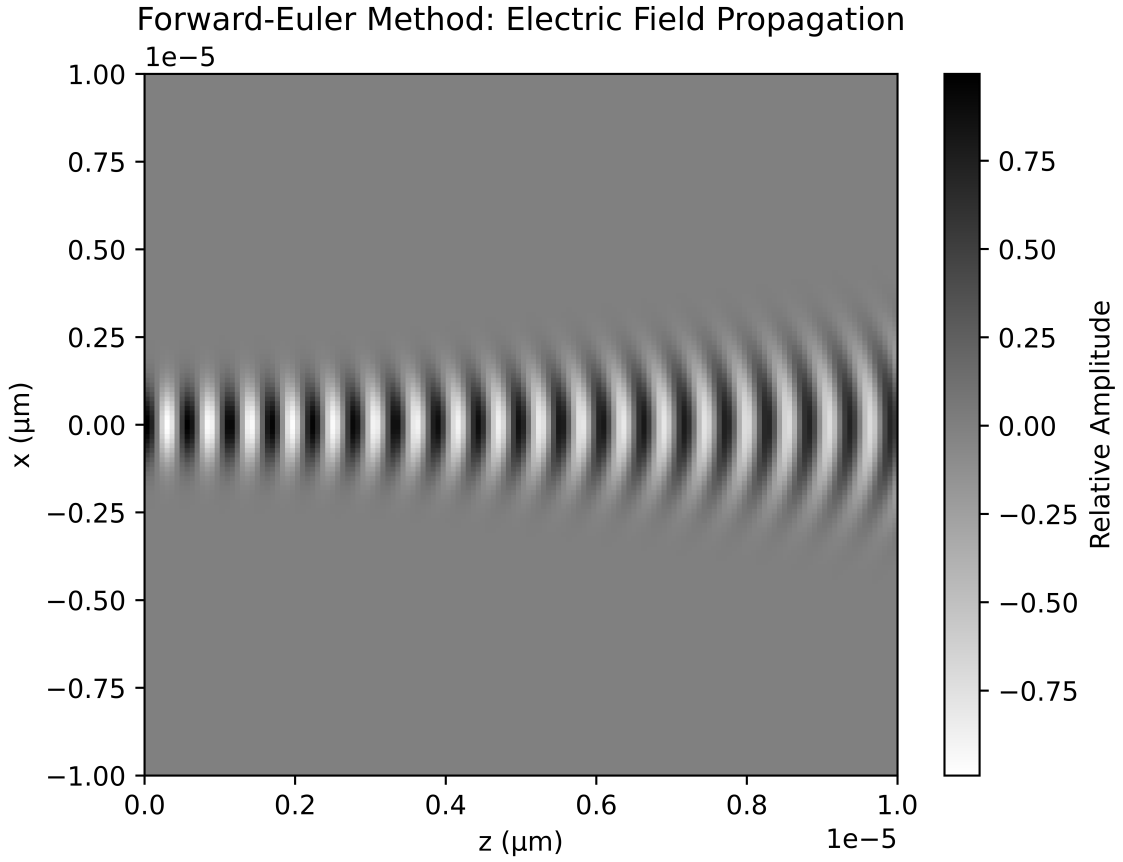
This is similar to the leapfrog method, except that the z–derivative is approximated by a forward difference:

$$\frac{\partial u}{\partial z} \approx \frac{u(x, z+\Delta z) - u(x,z)}{\Delta z}.$$

Consequently, the marching formula becomes

$$u(x, z+\Delta z) = u(x,z) + \frac{\Delta z\, \mathrm{i}}{2\,k\,(\Delta x)^2}\Big[u(x+\Delta x, z) + u(x-\Delta x, z) - 2\,u(x,z)\Big]. \tag{13}$$

The same boundary conditions (Robin at $x = \pm L$) are imposed as before.



The figure granularity is due to the numerical instability of this method when applied to the paraxial wave equation. For higher resolution, the solution begins to diverge.

Numerical solution 3: Crank-Nicolson

$$u_i^m = u(i\Delta x,\ m\Delta z), \quad 0 \le i \le N_x,\ 0 \le m \le N_z,$$

$$\frac{\partial u_i^m}{\partial z} = \frac{i}{2k}\,\frac{u_{i+1}^m + u_{i-1}^m - 2\,u_i^m}{\Delta x^2} = \frac{i}{2k}\,D_x\,u_i^m,$$

$$u_i^{m+1} - u_i^m = \int_{z^m}^{z^{m+1}} \frac{i}{2k}\,D_x\,u_i^m\,\mathrm{d}z,$$

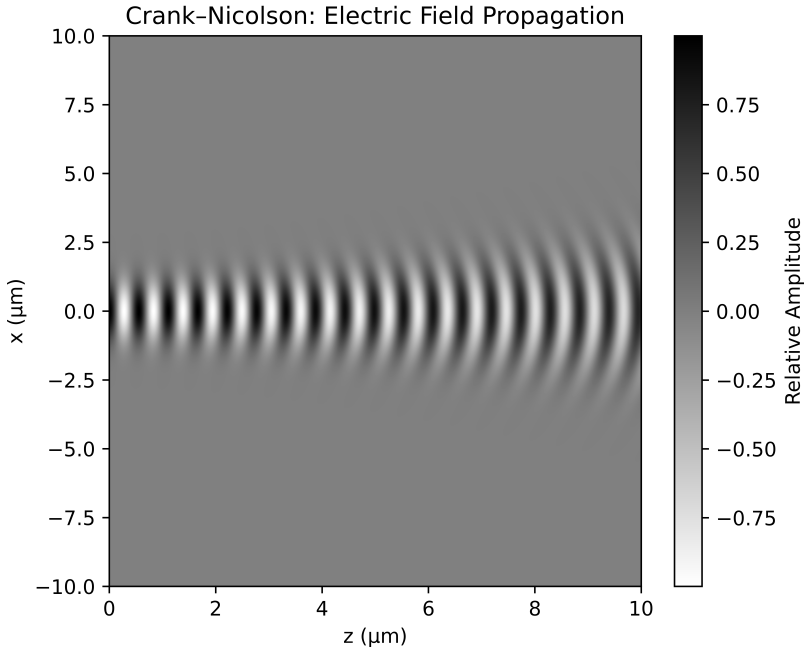
(trapezoidal rule) $\quad u_i^{m+1} - u_i^m \approx \dfrac{\Delta z}{2}\,\dfrac{i}{2k}\Big[D_x\,u_i^{m+1} + D_x\,u_i^m\Big],$

$$\implies \quad u_i^{m+1} = u_i^m + \frac{i\,\Delta z}{4k}\,\big[D_x\,u_i^{m+1} + D_x\,u_i^m\big],$$

or in matrix form: $\quad \big[I - \frac{i\,\Delta z}{4k}D_x\big]\,\mathbf{u}^{m+1} = \big[I + \frac{i\,\Delta z}{4k}D_x\big]\,\mathbf{u}^m,$

where $I \in \mathbb{R}^{N_x \times N_x}$ is the identity and $D_x \in \mathbb{R}^{N_x \times N_x}$ the second-difference matrix.

The same boundary conditions (Robin at $x = \pm L$) are imposed as before. This method uses the implicit midpoint rule (trapezoidal approximation) for the second derivative. It is unconditionally stable for the paraxial wave equation, hence the high resolution solution compared to the prior two methods.



Code: Rayleigh-Sommerfeld integral

```
import numpy as np
import matplotlib.pyplot as plt

def rayleigh_sommerfeld(E0, x, z, k):
    """
    1-D slice of the RS integral for a field E0(x') defined at z=0.
    Returns E(x,z) for every x in `x`.
    """
    Xp, X  = np.meshgrid(x, x)           # Xp: source, X: observation
    R      = np.sqrt((X - Xp)**2 + z**2)
    kernel =  z*np.exp(1j*k*R) / R**(1.5)

    # Use np.trapezoid for fast integration
    return np.trapezoid(kernel * E0[None, :], x, axis=1)

# ---------- parameters ----------
w0   = 1e-6             # beam waist (m)
λ    = 550e-9           # wavelength (m)
k    = 2*np.pi/λ
Nx   = 1000
Nz   = 1000
dist = 10
xmax = dist*w0
zmax = dist*w0

x = np.linspace(-xmax, xmax, Nx)
z = np.linspace(0,   zmax, Nz)
X, Z = np.meshgrid(x, z)

E0 = np.exp(-(x/w0)**2)             # Gaussian aperture
E  = np.vstack([rayleigh_sommerfeld(E0, x, zj, k) for zj in z])

z_nonzero = z.copy(); z_nonzero[0] = z[1]

E[:,0] = np.exp(-X[:,0]**2 / w0**2)  # initial condition
E = np.nan_to_num(E, nan=0.0)        # initial condition


# ---------- plot ----------
plt.figure(figsize=(6,5))

plt.imshow(
    (np.real(E).T)/np.max(np.abs(np.real(E))),  #normalization
    extent=[0, zmax*1e6, -xmax*1e6, xmax*1e6],  #display in units of
        microns
    origin='lower',
    aspect='auto',
    cmap='binary',
```

```
)

plt.xlabel('z (µm)')
plt.ylabel('x (µm)')
plt.title('Rayleigh-Sommerfeld: Electric Field Propagation')
plt.colorbar(label='Relative Amplitude')

plt.savefig('/Users/henryschnieders/desktop/rayleigh_sommerfeld.png', dpi
    =1200)
plt.show()
```

Code: Leapfrog method

```python
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import diags, lil_matrix, csc_matrix

def build_Dx_robin(Nx, dx, alpha, beta):
    """
    Build the 1D second-derivative matrix D_x with a Robin BC at both ends:
        alpha * u +/- beta * u_x = 0.
        star = (alpha/dx)/(alpha/dx + beta) - 2.
    """
    # 1) compute the modified diagonal entry at x=±L
    star = (alpha/dx) / (alpha/dx + beta) - 2

    # 2) create the three central diagonals [1, -2, 1]
    main_diag = -2 * np.ones(Nx, dtype=complex)
    off_diag  =  1 * np.ones(Nx-1, dtype=complex)

    # 3) assemble into a sparse LIL matrix
    Dx = diags(
        diagonals=[off_diag, main_diag, off_diag],
        offsets=[-1, 0, 1],
        shape=(Nx, Nx),
        format='lil'
    )

    # 4) overwrite the two boundary rows with the "star" on the diagonal
    Dx[0,   0] = star
    Dx[-1, -1] = star
    # (the off-diagonal +1 entries at [0,1] and [-1,-2] stay as they are)

    # 5) convert to CSC for fast solves / mat-vecs
    return Dx


x_num = 450
z_num = 450
wavelength = 550e-9
w0 = 1e-6  # beam waist
dist = 10
k = 2*np.pi/wavelength
dx = 2*dist*w0 / x_num
dz = dist*w0 / z_num
x_vals = np.linspace(-dist*w0, dist*w0, x_num)
z_vals = np.linspace(0, dist*w0, z_num)


alpha = -1j * 2*np.pi/wavelength   # e.g. Sommerfeld: α = -i k
```

```
beta  = +1.0                    #          β = 1
Dx    = build_Dx_robin(x_num, dx, alpha, beta)


soln_map = np.zeros((x_num, z_num), dtype=complex)

#center the initial profile in the middle of the image
initial_data = np.exp(-(x_vals)**2/w0**2)

soln_map[:, 0] = initial_data[:]

soln_map[:, 1] = 2*soln_map[:, 0] + (3*dz*1j / (k*dx**2)) * (Dx @ soln_map
    [:, 0]) #first step in z


for zmarch in range(2,z_num): #iterate from x_1 to x_N
    soln_map[:, zmarch] = soln_map[:, zmarch-2] + ((dz*1j) / (2*k*dx**2)) *
         (Dx @ soln_map[:, zmarch-1]) #march in z

for zs in range(z_num):
    for xs in range(x_num):
        soln_map[xs, zs] = np.real(soln_map[xs, zs]*np.exp(1j*k*z_vals[zs])
            ) #apply the phase factor to the solution

X, Z = np.meshgrid(x_vals, z_vals)

plt.figure(figsize=(6,5))
plt.imshow(np.real(soln_map), extent=[0, dist*w0*1e6, -dist*w0*1e6, dist*w0
    *1e6], aspect='auto', cmap='binary')
plt.colorbar(label='Relative Amplitude')
plt.title('Leapfrog method: Electric field propagation')
plt.ylabel('x (μm)')
plt.xlabel('z (μm)')

plt.savefig('/Users/henryschnieders/desktop/leapfrog_method.png', dpi=1200,
     bbox_inches='tight')
plt.show()
```

Code: Forward Euler method

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import diags

def build_Dx_robin(Nx, dx, alpha, beta):
    """
    Build the 1D second-derivative matrix D_x with a Robin BC at both ends:
        alpha * u +/- beta * u_x = 0.
        star = (alpha/dx)/(alpha/dx + beta) - 2.
    """
    # 1) compute the modified diagonal entry at x=±L
    star = (alpha/dx) / (alpha/dx + beta) - 2

    # 2) create the three central diagonals [1, -2, 1]
    main_diag = -2 * np.ones(Nx, dtype=complex)
    off_diag  =  1 * np.ones(Nx-1, dtype=complex)

    # 3) assemble into a sparse LIL matrix
    Dx = diags(
        diagonals=[off_diag, main_diag, off_diag],
        offsets=[-1, 0, 1],
        shape=(Nx, Nx),
        format='lil'
    )

    # 4) overwrite the two boundary rows with the "star" on the diagonal
    Dx[0,   0] = star
    Dx[-1, -1] = star
    # (the off-diagonal +1 entries at [0,1] and [-1,-2] stay as they are)

    # 5) convert to CSC for fast solves / mat-vecs
    return Dx


x_num = 150
z_num = 150
wavelength = 550e-9
w0 = 1e-6  # beam waist
dist = 10
k = 2*np.pi/wavelength
dx = 2*dist*w0 / x_num
dz = dist*w0 / z_num
x_vals = np.linspace(-10*w0, 10*w0, x_num)
z_vals = np.linspace(0, 10*w0, z_num)


alpha = -1j * 2*np.pi/wavelength    #  Sommerfeld: α = -i k
```

```
beta  = +1.0                    #          β = 1
Dx    = build_Dx_robin(x_num, dx, alpha, beta)


soln_map = np.zeros((x_num, z_num), dtype=complex)

#center the initial profile in the middle of the image
initial_data = np.exp(-(x_vals)**2/w0**2)

soln_map[:, 0] = initial_data[:]

soln_map[:, 1] = 2*soln_map[:, 0] + (3*dz*1j / (k*dx**2)) * (Dx @ soln_map
    [:, 0]) #first step in z


for zmarch in range(1,z_num): #iterate from x_1 to x_N
    soln_map[:, zmarch] = soln_map[:, zmarch-1] + ((dz*1j) / (2*k*dx**2)) *
        (Dx @ soln_map[:, zmarch-1]) #march in z


for zs in range(z_num):
    for xs in range(x_num):
        soln_map[xs, zs] = np.real(soln_map[xs, zs]*np.exp(1j*k*z_vals[zs])
            ) #apply the phase factor to the solution

X, Z = np.meshgrid(x_vals, z_vals)

plt.figure(figsize=(8, 8))
plt.imshow(np.real(soln_map), extent=[0, dist*w0*1e6, -dist*w0*1e6, dist*w0
    *1e6], aspect='auto', cmap='binary')
plt.ylabel('x (μm)')
plt.xlabel('z (μm)')
plt.title('Forward-Euler Method: Electric Field Propagation')
plt.colorbar(label='Relative Amplitude')

plt.savefig('/Users/henryschnieders/desktop/forward_euler.png', dpi=1200,
    bbox_inches='tight')
plt.show()
```

Code: Crank-Nicolson method

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import diags, eye
from scipy.sparse.linalg import spsolve

def build_Dx_robin(Nx, dx, alpha, beta):
    """
    Build 1D second_derivative D_x with Robin BC:
        alpha * u ± beta * u_x = 0
    """
    # central diagonals
    main = -2*np.ones(Nx, dtype=complex)
    off  =  np.ones(Nx-1, dtype=complex)
    D = diags([off, main, off], [-1,0,1], shape=(Nx,Nx), format='lil')

    # modified diagonal entry at boundaries
    star = (alpha/dx)/(alpha/dx + beta) - 2
    D[0,0]   = star
    D[-1,-1] = star
    return D.tocsc()

# 1) PARAMETERS
wavelength = 550e-9        # m
k          = 2*np.pi/wavelength # wavenumber (m^-1)
w0         = 1e-6          # beam waist (m)
dist       = 10            # range in units of w0

x_num = 2000                # transverse points
z_num = 2000                # propagation steps

# physical grids
x_vals = np.linspace(-dist*w0, dist*w0, x_num)
z_vals = np.linspace(0, dist*w0, z_num)
dx = x_vals[1] - x_vals[0]
dz = z_vals[1] - z_vals[0]

# Robin BC constants (Sommerfeld: α = -i k, β = 1)
alpha = -1j * k
beta  = +1.0

# 2) BUILD SECOND_DIFFERENCE & CN MATRICES
D = build_Dx_robin(x_num, dx, alpha, beta)

# Crank-Nicolson factor:
fac = 1j * dz / (4 * k * dx**2)

```

```
I = eye(x_num, format='csc') # 'Compressed Sparse Column', efficient for large sparse matrices
L = (I - fac * D).tocsc()    # left-hand side matrix
R = (I + fac * D).tocsc()    # right-hand side matrix

# 3) INITIALIZE & MARCH
u = np.exp(-x_vals**2 / w0**2).astype(complex)
soln = np.zeros((x_num, z_num), dtype=complex)
soln[:,0] = u

for m in range(z_num-1):
    rhs    = R.dot(u)
    u_next = spsolve(L, rhs) # "Sparse Solve" is used because this solves the system for sparse matrices much faster
    soln[:, m+1] = u_next
    u = u_next

# 4) PLOT REAL PART

for zs in range(z_num):
    for xs in range(x_num):
        soln[xs, zs] = np.real((soln[xs, zs]*np.exp(1j*k*z_vals[zs]) ))

plt.figure(figsize=(6,5))
plt.imshow(
    np.real(soln),
    extent=[0, dist*w0*1e6, -dist*w0*1e6, dist*w0*1e6],
    origin='lower',
    aspect='auto',
    cmap='binary',
)
plt.colorbar(label='Relative Amplitude')
plt.xlabel('z (μm)')
plt.ylabel('x (μm)')
plt.title('Crank-Nicolson: Electric Field Propagation')
plt.savefig('/Users/henryschnieders/Desktop/Crank_Nicolson.png', dpi=1200)
plt.show()
```