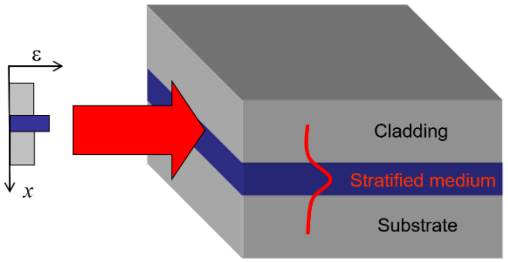
Computational Photonics

Seminar 05, 16.05.2024

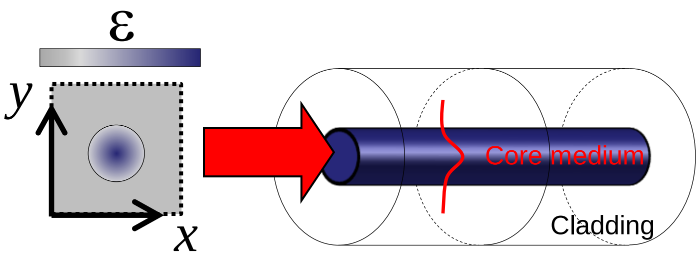
Implementation of a Finite-Difference Mode Solver

(2nd order finite difference schemes in matrix notation)

- Calculation of the guided modes in a **slab waveguide** system (1+1=2D)



- Calculation of the guided modes in a **strip waveguide** system (2+1=3D)



Abbe School of Photonics | JENA
Friedrich-Schiller-Universität

Guided modes in 1+1D systems (matrix form)

Analytic equation of continuous variable x

$$\left(\frac{1}{k_0^2}\frac{\partial^2}{\partial_x^2} + \varepsilon(x,\omega)\right)E_0(x) = \varepsilon_{eff}E_0(x)$$

matrix diagonal: $-2/h^2 + k_0^2\varepsilon$
adjacent diagonals: $1/h^2$

Algebraic equation of discrete variable x_i

$$\frac{1}{k_0^2}\begin{pmatrix} -\frac{2}{h^2}+k_0^2\varepsilon_1 & 1/h^2 & 0 & 0 & 0 & 0 & \cdots \\ 1/h^2 & -\frac{2}{h^2}+k_0^2\varepsilon_2 & 1/h^2 & 0 & 0 & 0 & \cdots \\ 0 & 1/h^2 & -\frac{2}{h^2}+k_0^2\varepsilon_3 & 1/h^2 & 0 & 0 & \cdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & \cdots \\ & & & & & & \end{pmatrix}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix} = \varepsilon_{\mathrm{eff}}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix}$$

`numpy.diag` and `numpy.linalg.eig` might be useful

Task I: Assignment description

- Use the input variables: $\varepsilon(x,\omega), k_0, h$
- Assume a Gaussian waveguide profile: $\varepsilon(x,\omega) = \varepsilon_{\text{Substrate}} + \Delta\varepsilon e^{-(x/W)^2}$
- Select guided modes according to their eigenvalue: $\varepsilon_{\text{Substrate}} < \varepsilon_{\text{eff}} < \varepsilon(x)_{\max}$
- Use the given parameters (see separate files) for testing

<u>Subtasks:</u>

1. Calculate the **effective permittivity** and **field distribution** of the guided eigenmodes in TE-polarization and for PEC boundary conditions!
2. Show and discuss the dependence of your numerical solution on the discretization size *h*! (Hint: Consider the trade-off between convergence and computation time and motivate a reasonable cut-off value. `time.time` and `numpy.logspace` might be useful.)

Task I: Script Mode solver

- **Step 1:** 1D mode solver script

Task I: Script Mode solver

```
def guided_modes_1DTE(prm, k0, h):
```

...

```
    # set up operator matrix
    main = -2.0 * np.ones(len(prm)) / (h**2 * k0**2) + prm
    sec = np.ones(len(prm) - 1) / (h**2 * k0**2)
    L = np.diag(sec, -1) + np.diag(main, 0) + np.diag(sec, 1)

    # solve eigenvalue problem
    eff_eps, guided = np.linalg.eig(L)

    # pick only guided modes
    idx = (eff_eps < np.max(prm)) & (eff_eps > np.min(prm))
    eff_eps = eff_eps[idx]
    guided = guided[:, idx]

    # sort modes from highest to lowest effective permittivity
    # (the fundamental mode has the highest effective permittivity)
    idx = np.argsort(-eff_eps)
    eff_eps = eff_eps[idx]
    guided = guided[:, idx]
    return eff_eps, guided
```

Task I: Script Mode solver

```
# set up operator matrix
main = -2.0 * np.ones(len(prm)) / (h**2 * k0**2) + prm
sec = np.ones(len(prm) - 1) / (h**2 * k0**2)
L = np.diag(sec, -1) + np.diag(main, 0) + np.diag(sec, 1)

# solve eigenvalue problem
eff_eps, guided = np.linalg.eig(L)

# pick only guided modes
```

$$\frac{1}{k_0^2}\begin{pmatrix} -\frac{2}{h^2}+k_0^2\varepsilon_1 & 1/h^2 & 0 & 0 & 0 & 0 & \cdots \\ 1/h^2 & -\frac{2}{h^2}+k_0^2\varepsilon_2 & 1/h^2 & 0 & 0 & 0 & \cdots \\ 0 & 1/h^2 & -\frac{2}{h^2}+k_0^2\varepsilon_3 & 1/h^2 & 0 & 0 & \cdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix} = \varepsilon_{\text{eff}}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix}$$

Task I: Script Mode solver

```
# set up operator matrix
main = -2.0 * np.ones(len(prm)) / (h**2 * k0**2) + prm
sec = np.ones(len
L = np.diag(sec,                                        , 1)

# solve eigenvalue problem
eff_eps, guided = np.linalg.eig(L)
```

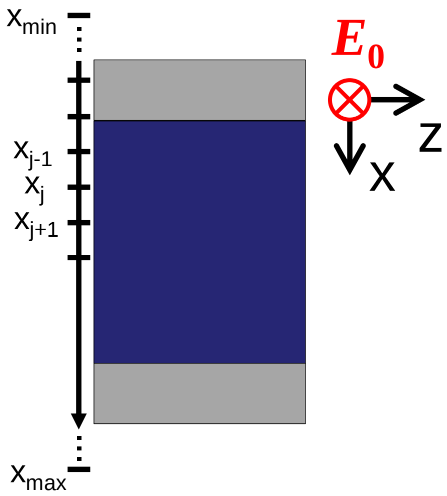
$$\varepsilon_{\text{Substrate}} < \varepsilon_{\text{eff}} < \varepsilon(x)_{\text{max}}$$

```
# pick only guided modes
idx = (eff_eps < np.max(prm)) & (eff_eps > np.min(prm))
eff_eps = eff_eps[idx]
guided = guided[:, idx]

# sort modes from highest to lowest effective permittivity
# (the fundamental mode has the highest effective permittivity)
idx = np.argsort(-eff_eps)
eff_eps = eff_eps[idx]
guided = guided[:, idx]
return eff_eps, guided
```

Task I: Test Mode solver

- **Step 1:** 1D mode solver script
- **Step 2:** Check convergence for
  - Step size ***h***
  - Grid size ***N***



Task I: Convergence Test – e.g. Grid Size

New script to iterate over simulation parameters

```
''' Tests the convergence when varying the grid size.
'''
import time
import numpy as np
from matplotlib import pyplot as plt
from Homework_1_solution import guided_modes_1DTE
```

...

```
save_figures = False
```

Set up physical parameters

```
# %%
h              = 0.2
lam            = 0.78
k0             = 2 * np.pi / lam
e_substrate    = 1.5**2
delta_e        = 1.5e-2
num_modes      = 15
w              = 15.0
```

Task I: Convergence Test – e.g. Grid Size

Logarithmically spaced grid sizes

```
#grid_sizes = np.logspace(np.log10(2 * w), np.log10(100 * w), 50)
grid_sizes = np.logspace(np.log10(2 * w), np.log10(30 * w), 20)
eff_eps = np.nan * np.zeros((grid_sizes.size, num_modes))
for i, grid_size in enumerate(grid_sizes):
    Nx = np.ceil(int(0.5 * grid_size / h))
    x = np.arange(-Nx, Nx + 1) * h
    prm = e_substrate + delta_e * np.exp(-(x/w)**2)
    start = time.time()
    mode_eps, guided = guided_modes_1DTE(prm, k0, h)
    N = min(num_modes, len(mode_eps))
    eff_eps[i, :N] = mode_eps[:N]
    stop = time.time()
    print("grid size = %8.3fµm, time = %gs" % (grid_size, stop - start))
```

Task I: Convergence Test – e.g. Grid Size

For each grid size, initialize physical domain…

```
#grid_sizes = np.logspace(np.log10(2 * w), np.log10(100 * w), 50)
grid_sizes = np.logspace(np.log10(2 * w), np.log10(30 * w), 20)
eff_eps = np.nan * np.zeros((grid_sizes.size, num_modes))
for i, grid_size in enumerate(grid_sizes):
    Nx = np.ceil(int(0.5 * grid_size / h))
    x = np.arange(-Nx, Nx + 1) * h
    prm = e_substrate + delta_e * np.exp(-(x/w)**2)
    start = time.time()
    mode_eps, guided = guided_modes_1DTE(prm, k0, h)
    N = min(num_modes, len(mode_eps))
    eff_eps[i, :N] = mode_eps[:N]
    stop = time.time()
    print("grid size = %8.3fµm, time = %gs" % (grid_size, stop - start))
```

Task I: Convergence Test – e.g. Grid Size

… calculate modes and run time …

```
#grid_sizes = np.logspace(np.log10(2 * w), np.log10(100 * w), 50)
grid_sizes = np.logspace(np.log10(2 * w), np.log10(30 * w), 20)
eff_eps = np.nan * np.zeros((grid_sizes.size, num_modes))
for i, grid_size in enumerate(grid_sizes):
    Nx = np.ceil(int(0.5 * grid_size / h))
    x = np.arange(-Nx, Nx + 1) * h
    prm = e_substrate + delta_e * np.exp(-(x/w)**2)
    start = time.time()
    mode_eps, guided = guided_modes_1DTE(prm, k0, h)
    N = min(num_modes, len(mode_eps))
    eff_eps[i, :N] = mode_eps[:N]
    stop = time.time()
    print("grid size = %8.3fµm, time = %gs" % (grid_size, stop - start))
```

Task I: Convergence Test – e.g. Grid Size

… and calculate error for modes that are stable with grid size

```
# remove modes that do not exist for any grid size
# = remove mode number if eff_eps is NaN for all grid sizes
eff_eps = eff_eps[:, ~np.all(np.isnan(eff_eps), axis=0)]

# calculate relative error to the value obtained at highest resolution
# this should approximate the true error
rel_error = np.abs(eff_eps[:-1, :] / eff_eps[-1, :] - 1.0)
```

See:

testscript1d_convergence_grid_size.py

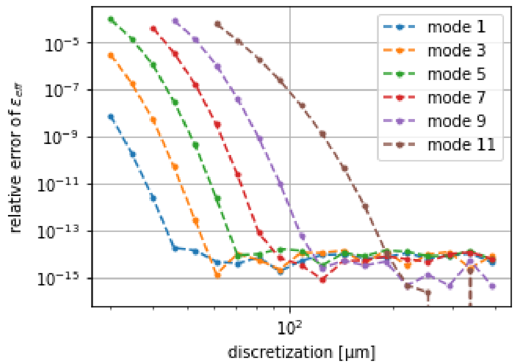
&

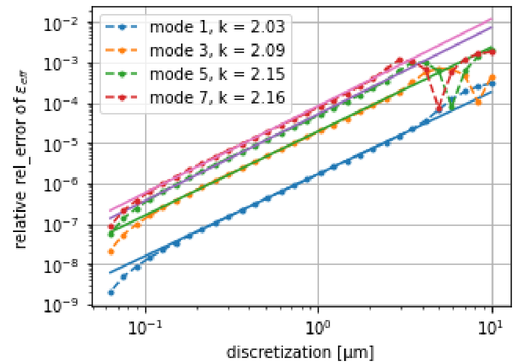
testscript1d_convergence_discretization.py

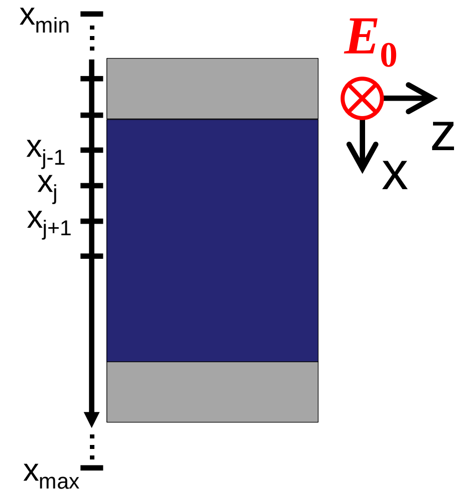
Task I: Test Mode solver

- **Step 1:** 1D mode solver script

- **Step 2:** Check convergence for
  - Step size ***h***
  - Grid size ***N***





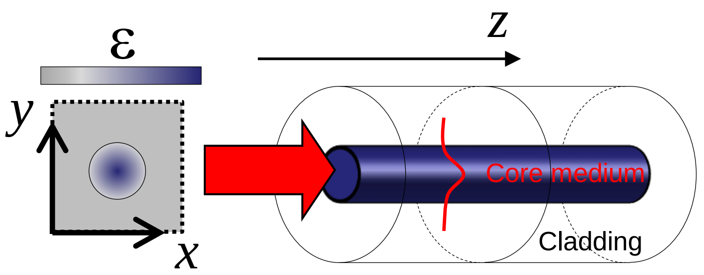


- **Step 3:** Calculate mode indices and mode profiles

See: **`Homework_1_testscript1d.py`**

Guided modes in 2+1 (=3D) systems (strip waveguide)

in scalar approximation



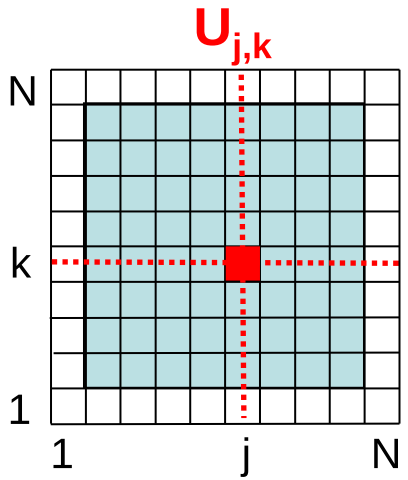
Assumptions:

- no z-dependence
- weak guiding: $\varepsilon_0 \varepsilon(\omega) \operatorname{div} \mathbf{E}(\mathbf{r},\omega) \approx 0$

Ansatz:

- scalar field and stationary states (modes)

$$v(\mathrm{r}) = u(x,y)\exp\left(\mathbf{i}\beta z\right)$$



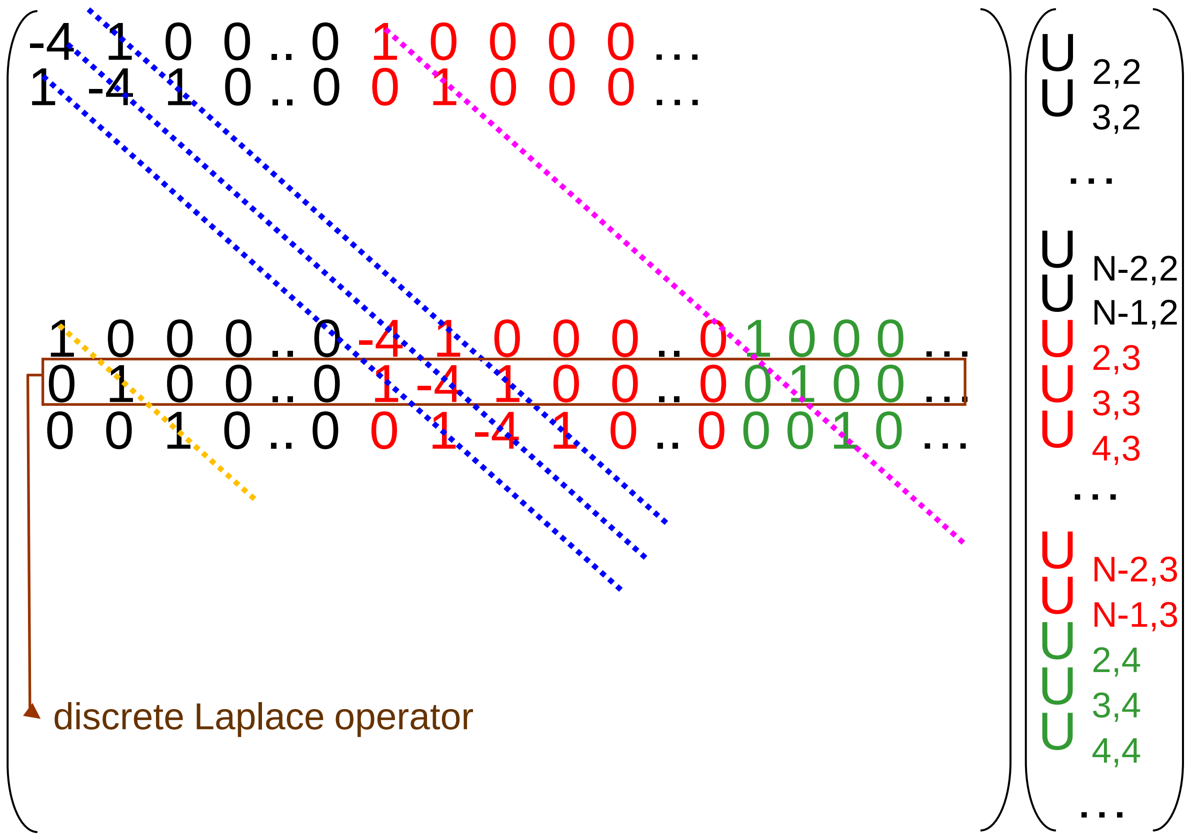
- leads to scalar Helmholtz equation (eigenvalue problem for scalar field "intensity")

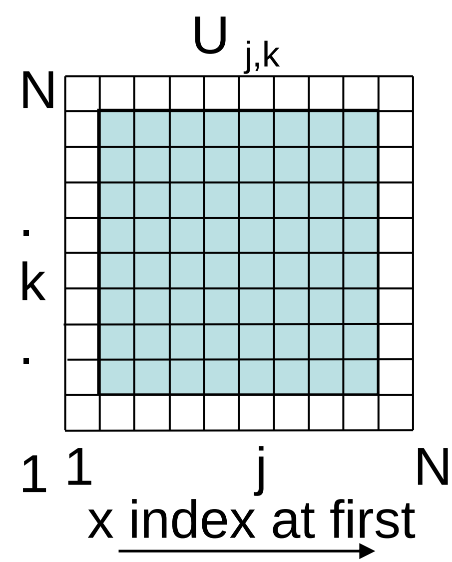
$$\left[\boxed{\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}} + k_0^2 \varepsilon(x,y,\omega)\right] u(x,y) = \beta^2(\omega) u(x,y)$$

Discretization in 2D:

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_{x_j,y_k} + \left.\frac{\partial^2 f}{\partial y^2}\right|_{x_j,y_k} \approx \frac{f\left(x_{j+1},y_k\right) + f(x_{j-1},y_k) + f(x_j,y_{k+1}) + f(x_j,y_{k-1}) - 4f(x_j,y_k)}{h^2}$$

Numerical implementation of the 2D Laplace operator





Note: this only resembles the Laplace operator of the Helmholtz eq.

Differentiation with respect to x (same as 1D)

Differentiation with respect to y

Use the concept of sparse matrices: `scipy.sparse.linalg.eigs` and `scipy.sparse.spdiags` might be useful.

Task II: Assignment description

- Use the input variables: $\varepsilon(x,\omega)$, k_0, h, $numb$
- Assume a Gaussian waveguide profile: $\varepsilon(x,y,\omega) = \varepsilon_{\text{Cladding}} + \Delta\varepsilon \mathrm{e}^{-\frac{x^2+y^2}{W^2}}$
- Select guided modes according to their eigenvalue: $\varepsilon_{\text{Cladding}} < \varepsilon_{\text{eff}} < \max(\varepsilon(x,y))$
- Use the given parameters (see separate files) for testing

Subtasks:

1. Calculate the **effective permittivity** and **field distribution** of the scalar (quasi-TE polarized) guided eigenmodes for PEC boundary conditions!
2. Show and discuss the dependence of the numerical solution on the discretization size *h*! (Hint: Consider the trade-off between convergence and computation time and motivate a reasonable cut-off value. `time.time` and `numpy.logspace` might be useful.)

Task II: Script Mode solver

```python
def guided_modes_2D(prm, k0, h, numb):
```

...

```python
    NX, NY = prm.shape
    N = NX * NY
    prmk = prm * k0**2
    ihx2 = 1 / h**2
    ihy2 = 1 / h**2

# 'F' means to index the elements in column-major,
# with the first index changing fastest, and the last index changing slowest.
    md = -2.0 * (ihx2 + ihy2) * np.ones(N) + prmk.ravel(order='F')
    xd = ihx2 * np.ones(N)
    yd = ihy2 * np.ones(N)
    H = sps.spdiags([yd, xd, md, xd, yd], [-NX, -1, 0, 1, NX], N, N, format='csc')

    # remove the '1' when moving to a new line
    # -> look into script pg. 31, upper and lower blue line in first figure
    for i in range(1, NY):
        n = i * NX
        H[n-1, n] = 0
        H[n, n-1] = 0
```

See: **Homework_1_testscript2d.py**