Numerical Simulation Methods in Geophysics, Exercise 13: 1D/2D EM

1. MGPY+MGIN

thomas.guenther@geophysik.tu-freiberg.de

TU Bergakademie Freiberg
TUBAF
Die Ressourcenuniversität.
Seit 1765.

1D/2D (FD) EM modelling

The induction equation for perpendicular (E or H) fields

$$-\boldsymbol{\nabla}^2 u + \imath\omega\mu\sigma u = f$$

is discretized by stiffness matrix $\mathbf{A}$ and mass matrix $\mathbf{M}$

$$(\mathbf{A} + \imath\omega\mathbf{M})\mathbf{u} = \mathbf{f}$$

See Lecture in Theory EM

TM polarization

$$\nabla \times \sigma^{-1} \nabla \times \mathbf{H} + \imath\omega\mu\mathbf{H} = \nabla \times \sigma^{-1} \mathbf{j}_s$$

Transverse magnetic (TM) mode

Assume the source field is oscillating perpendicular to the modelling plane, i.e.

$\mathbf{H} = [H_x, 0, 0]^T e^{\imath\omega t}$.

Then the PDE holds for the scalar H_x (now only H)

$$-\nabla \cdot \sigma^{-1} \nabla H_x(y, z) + \imath\omega\mu H_x(y, z) = 0$$

TE polarization

$$\boldsymbol{\nabla} \times \mu^{-1} \boldsymbol{\nabla} \times \mathbf{E} + \imath\omega\sigma\mathbf{E} = \boldsymbol{\nabla} \times \mathbf{j}_s$$

Transverse electric (TE) mode

Assume the source field is oscillating perpendicular to the modelling plane, i.e.

$\mathbf{E} = [E_x, 0, 0]^T e^{\imath\omega t}$.

Then the PDE holds for the scalar E_x (now only E)

$$-\boldsymbol{\nabla} \cdot \mu^{-1} \boldsymbol{\nabla} E_x(y, z) + \imath\omega\sigma E_x(y, z) = 0$$

The analytical solution

$$H = H_0 e^{-\imath k z} \quad \text{with} \quad k^2 = -\imath \omega \mu \sigma$$

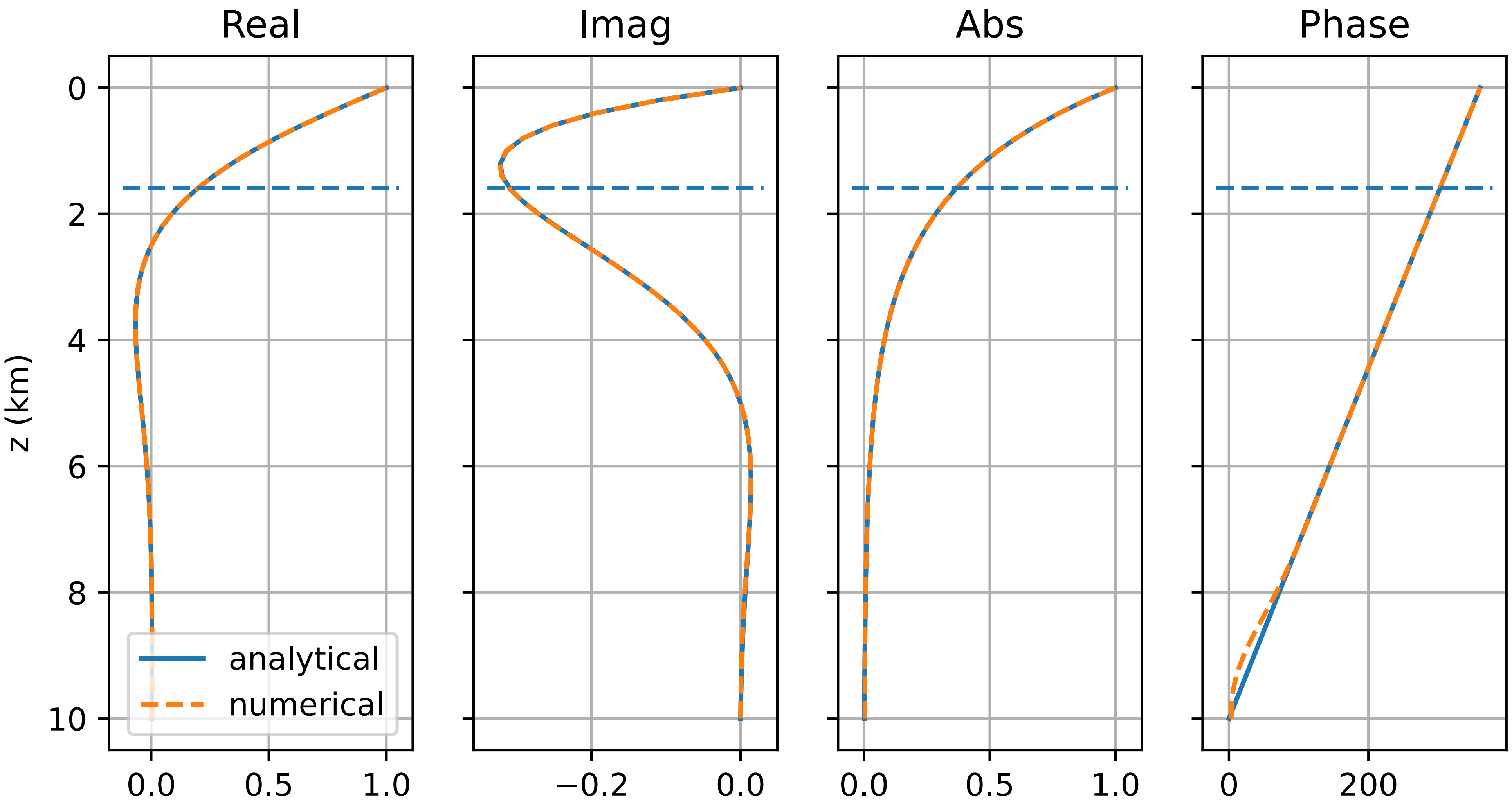
```
import numpy as np
import matplotlib.pyplot as plt
from poisson1d import stiffnessMatrix1DFE, massMatrix1DFE
T = 0.1  # 1, 10
w = 2 * np.pi / T
sigma0 = 1/100  # 100 Ohmm, 10, 1000
mu = np.pi * 4e-7
d = np.sqrt(2/(mu*w*sigma0))
print(d)
z = np.arange(0, 100001, 100)

def Hanalytical(z, sigma, w):
    k = np.sqrt(-1j*w*mu*sigma0)

#def plotResults(z, u)
fig, ax = plt.subplots(ncols=4, sharey=True)
```

```
1591.5494309189535
```

The analytical solution



The problem in 1D

```
import numpy as np
import matplotlib.pyplot as plt
from poisson1d import stiffnessMatrix1DFE, massMatrix1DFE
T = 0.1  # 0.1
w = 2 * np.pi / T
sigma0 = 1/100  # 1/100
mu = np.pi * 4e-7
d = np.sqrt(2/(mu*w*sigma0))
z = np.arange(-10000, 0.1, 200)

A, b = stiffnessMatrix1DFE(x=z, uL=1)
M = massMatrix1DFE(x=z, a=mu*sigma0)
AM = A + M * 1j * w
u = np.linalg.solve(AM, b)
# plotResult(u)
```

Complex or real-valued?

The complex-valued system

$$(\mathbf{A} + \imath\omega\mathbf{M})\mathbf{u} = (\mathbf{A} + \imath\omega\mathbf{M})(\mathbf{u}_r + \imath\mathbf{u}_i) = \mathbf{b}_r + \imath\mathbf{b}_i$$

can be transferred into a doubled real-valued system

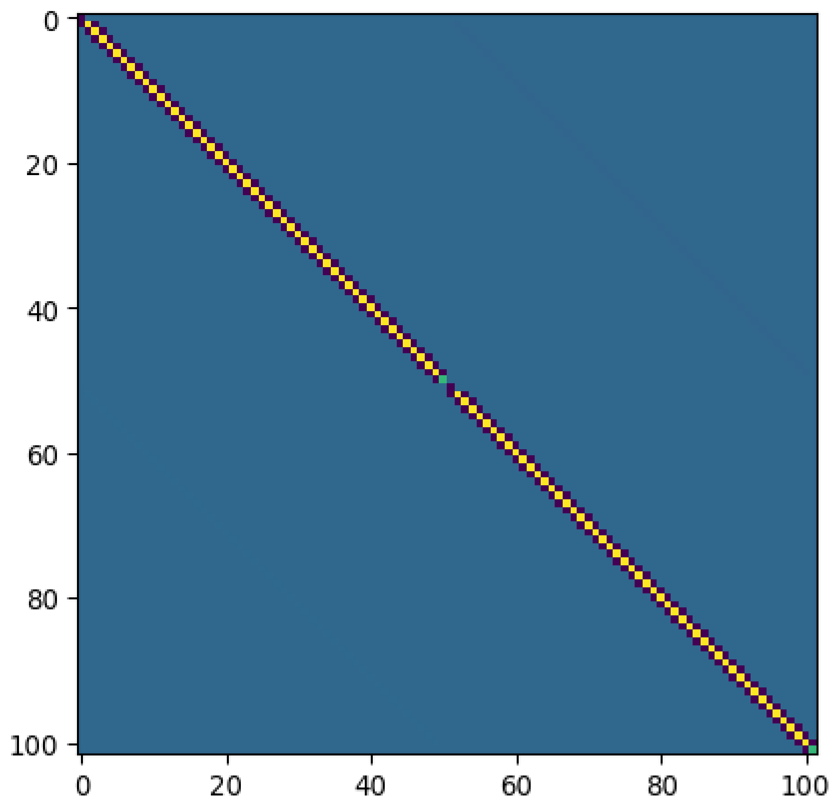
$$\mathbf{A}\mathbf{u}_r + \imath\mathbf{A}u_i + \imath\omega\mathbf{M}\mathbf{u}_r - \omega\mathbf{M}\mathbf{u}_i = \mathbf{b}_r + \imath\mathbf{b}_i$$

$$\begin{pmatrix} A & -\omega M \\ \omega M & A \end{pmatrix} \begin{pmatrix} u_r \\ u_i \end{pmatrix} = \begin{pmatrix} b_r \\ b_i \end{pmatrix}$$

Complex-to-real conversion (numpy)

$$\mathbf{B} = \begin{pmatrix} \mathbf{A} & -\omega\mathbf{M} \\ \omega\mathbf{M} & \mathbf{A} \end{pmatrix}$$

```
D = np.vstack([np.hstack([A, -M*w]),
               np.hstack([M*w, A])])
plt.imshow(D)
d = np.hstack([b, b*0])
uri = np.linalg.solve(D, d)
u = uri[:len(z)] + uri[len(z):] * 1j
```



Complex-to-real conversion (pyGIMLi)

$$\mathbf{B} = \begin{pmatrix} \mathbf{A} & -\omega\mathbf{M} \\ \omega\mathbf{M} & \mathbf{A} \end{pmatrix}$$

Secondary field approach

Consider the field to consist of a primary (background) and an secondary (anomalous) field $F = F_0 + F_a$

solution for F_0 known, e.g. analytically or 1D (semi-analytically)

$\Rightarrow$ form equations for F_a, because

- F_a is weaker or smoother (e.g. $F_0 \propto 1/$ at sources)
- boundary conditions easier to set (e.g. homogeneous Dirichlet)

Secondary field Helmholtz equation

The equation $-\boldsymbol{\nabla}^2 F - k^2 F = 0$ is solved by the primary field for k_0:

$-\boldsymbol{\nabla}^2 F_0 - k_0^2 F_0 = 0$ and the total field for $k_0 + \delta k$:

$$-\boldsymbol{\nabla}^2 (F_0 + F_a) - (k_0^2 + \delta k^2)(F_0 + F_a) = 0$$

$$-\boldsymbol{\nabla}^2 F_a - k^2 F_a = \delta k^2 F_0$$

Note

Source terms only arise at anomalous terms, weighted by the primary field.

Secondary field for E field

$$-\boldsymbol{\nabla}^2\mathbf{E}_0 + \imath\omega\mu\sigma_0\mathbf{E}_0 = 0$$

leads to

$$-\boldsymbol{\nabla}^2\mathbf{E}_a + \imath\omega\mu\sigma\mathbf{E}_a = -\imath\omega\mu\delta\sigma\mathbf{E}_0$$

Note

Source terms only arise at anomalous conductivities and increase with primary field

Secondary field in matrix form

$$-\boldsymbol{\nabla}^2\mathbf{E}_a + \imath\omega\mu\sigma\mathbf{E}_a = -\imath\omega\mu\delta\sigma\mathbf{E}_0$$

leads to the discretized form ($\mathbf{A}$-stiffness, $\mathbf{M}$-mass)

$$\mathbf{A}\mathbf{u}_a + \imath\omega\mathbf{M}_\sigma\mathbf{u}_a = (\mathbf{A} + \imath\omega\mathbf{M}_\sigma)\mathbf{u}_a = -\imath\omega\mathbf{M}_{\delta\sigma}\mathbf{u}_0$$

```
A = stiffnessMatrix1DFE(x=z)
M = massMatrix1DFE(x=z, a=w*mu*sigma)
dM = massMatrix1DFE(x=z, a=w*mu*(sigma-sigma0))
u = uAna + solve(A+M*w*1j, dM@uAna * w*1j)
```

Secondary field for H field

$$-\boldsymbol{\nabla} \cdot \rho_0 \boldsymbol{\nabla} \mathbf{H}_0 + \imath\omega\mu \mathbf{H}_0 = 0$$

leads to

$$-\boldsymbol{\nabla} \cdot \rho \boldsymbol{\nabla} \mathbf{H}_a + \imath\omega\mu \mathbf{H}_a = \boldsymbol{\nabla} \cdot \delta\rho_0 \boldsymbol{\nabla} \mathbf{H}_0$$

$$\mathbf{A}_\rho \mathbf{u}_a + \imath\omega \mathbf{M} \mathbf{u}_a = (\mathbf{A}_\rho + \imath\omega \mathbf{M}) \mathbf{u}_a = -\mathbf{A}_{\delta\rho} \mathbf{u}_0$$

Task

1. Create a 1D discretization vector (once with air and once without)
2. Use your own functions to generate stiffness and mass matrix
3. Implement Dirichlet BC on the upper and Neumann BC on the lower boundary
4. Choose a period and solve the equation for homogeneous conductivity
5. Compare with the analytical solution
6. Solve for both E and H, play with conductivity and period
7. Add inhomogeneity and repeat computations