[1]:
```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, sin
```

[2]:
```
from matplotlib import animation
from IPython.display import HTML
```

Course: Computational Electrodynamics and Applications

Tutor: Theodoros Samaras

Hand-in date: June 5, 2020

Student: Konstantinos Foutzopoulos < kfoutzop@auth.gr >

### 0.1 Total-field/Scattered-field implementation in 1D FDTD

- The code that implemented the required technique is given below.

[3]:
```
def source_fun(n):
    return sin(wdt*n)

def fdtd(n):
    hy[:] = hy[:] + iimpz*(ez[1:] - ez[:-1])
    hy[bs] -= iimpz*source_fun(n)  #Hy correction @ tf/sf boundary
    ez[0] = ez[1]                  #Mur's 1st-order ABC @ x=0
    ez[-1] = ez[-2]                #Mur's 1st-order ABC @ x=L
    ez[1:-1] = ca*ez[1:-1] + cb*(hy[1:] - hy[:-1])
    ez[be] += source_fun(n+1)      #Hz correction @ tf/sf boundary
    e0z[:] = np.maximum(ez, e0z)
```

For this program Courant number is hardcoded to unity. Thus

$$S = cdt/dx = 1 \Rightarrow dx = cdt$$

which means that the field moves one spatial step for every time step.

The first-order Mur's ABC is derived

$$E_0^{n+1} = E_1^n + \frac{S-1}{S+1}(E_1^{n+1} - E_0^n) \overset{S=1}{\Rightarrow} E_0^{n+1} = E_1^n$$

and

$$E_N^{n+1} = E_{N-1}^n + \frac{S-1}{S+1}(E_{N-1}^{n+1} - E_N^n) \overset{S=1}{\Rightarrow} E_N^{n+1} = E_{N-1}^n$$

[4]:
```
#set constants
cz = 2.99792458e8
```

```
muz = 4*pi*1e-7
```

[5]:
```
#derived constants
epsz = 1/(cz*cz*muz)
iimpz = cz*epsz
```

[6]:
```
#options
Nx = 400            #no of spatial nodes
Nt = 400            #no of time steps
bs = 29             #tf/sf boundary, start
be = 30             #tf/sf boundary, end
```

[7]:
```
freq = 1e9          #source frequency
```

[8]:
```
#derived variables
wl = cz/freq        #wavelength
dx = wl/20          #spatial step
dt = dx/cz          #time step
w = 2*pi*freq       #angular freq
wdt = w*dt
print("dx = {}".format(dx))
print("dt = {}".format(dt))
```

```
dx = 0.0149896229
dt = 4.9999999999999995e-11
```

[9]:
```
def set_zero():     #initialize ez, hy, e0z
    return np.zeros(Nx), \
           np.zeros(Nx - 1), \
           np.zeros(Nx)

eps = np.ones(Nx)  #relative electric permittivity
sig = np.zeros(Nx) #electric conductivity
x = np.linspace(0, Nx*dx, Nx)
```

[10]:
```
eps[200:] = 9
sig[200:] = 0.02   #conductor
```

[11]:
```
ca = (1.0 - (dt*sig)/(2.0*epsz*eps))/(1.0 + (dt*sig)/(2.0*epsz*eps))
ca = ca[1:-1]
cb = (dt/epsz/eps/dx)/(1.0 + (dt*sig)/(2.0*epsz*eps))
cb = cb[1:-1]
```

We first check that our code is working as expected.

[12]:
```
%%capture
ez, hy, e0z = set_zero()
```

```
fig, ax = plt.subplots()
line1, = ax.plot([], [], lw=2)
line2, = ax.plot([], [], lw=2)

ax.set_xlim(0, Nx*dx)
ax.set_ylim(-2, 2)
ax.set_xlabel('x')
ax.set_ylabel('E(x,t)')

def init():
    line1.set_data([], [])
    line2.set_data([], [])
    return (line1, line2)

def animate(n):
    fdtd(n)
    line1.set_data(x[:], ez[:])
    line2.set_data(x[:], e0z[:])
    return (line1, line2)

anim = animation.FuncAnimation(fig, animate, init_func=init, frames=Nt)
```

[13]:
```
#HTML(anim.to_jshtml())
```

We then use the code to show that the skin depth of a conductor is inversely proportional to the wave's frequency and the conductor's conductivity.

[14]:
```
ie = 1/np.exp(1)
```

(i)

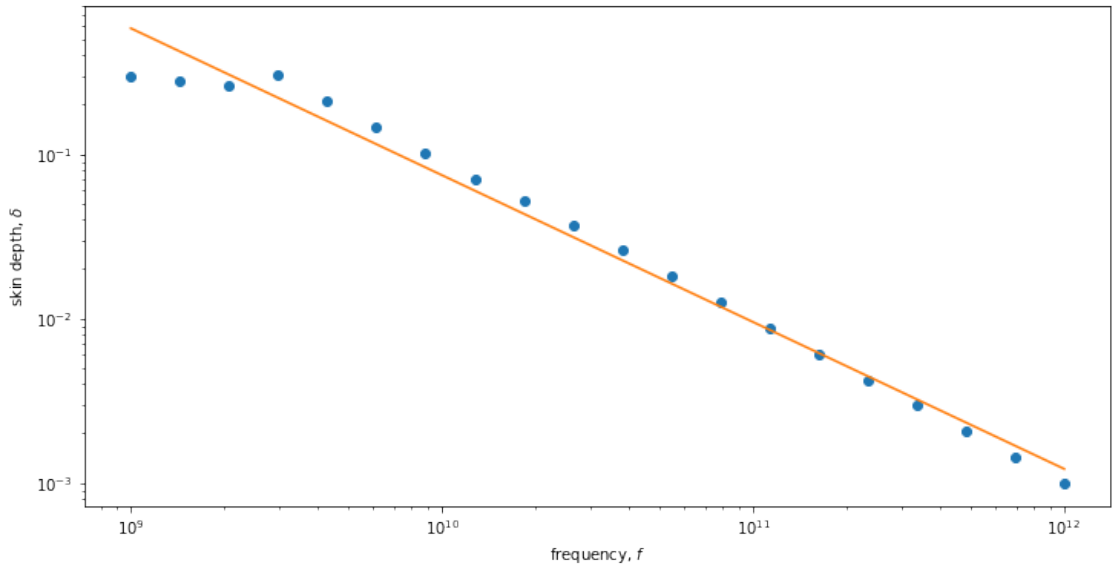
[15]:
```
#conductivity constant
sig[200:] = 0.02
#change frequency
x = np.logspace(np.log10(1e9), np.log10(1e12), num=20)
y = np.zeros(len(x))

for i, freq in enumerate(x):
    dx = (cz/freq)/20.0
    dt = dx/cz
    w = 2*pi*freq
    wdt = w*dt
    ca = (1.0 - (dt*sig)/(2.0*epsz*eps))/(1.0 + (dt*sig)/(2.0*epsz*eps))
    ca = ca[1:-1]
    cb = (dt/epsz/eps/dx)/(1.0 + (dt*sig)/(2.0*epsz*eps))
    cb = cb[1:-1]
```

```
    ez, hy, e0z = set_zero()
    for n in range(Nt):
        fdtd(n)
    y[i] = np.argmax(e0z[200:] < ie)*dx
```

[16]:
```
logx, logy = np.log10([x,y])
fit = np.polyfit(logx, logy, 1)
yfit = 10**fit[1]*x**fit[0]
```

[17]:
```
plt.figure(figsize=(12, 6))
plt.loglog(x, y, 'o')
plt.plot(x, yfit)
plt.xlabel('frequency, $f$')
plt.ylabel('skin depth, $$')
plt.show()
```



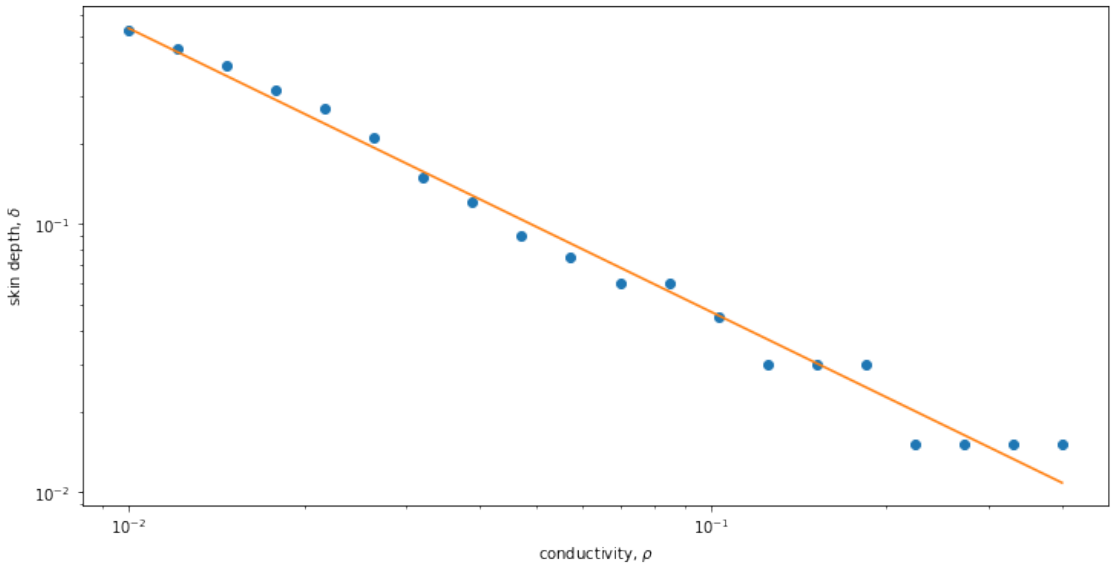

(ii)

[18]:
```
#frequency constant
freq = 1e9
dx = (cz/freq)/20.0
dt = dx/cz
w = 2*pi*freq
wdt = w*dt
#change conductivity
x = np.logspace(np.log10(0.01), np.log10(0.40), num=20)
y = np.zeros(len(x))
```

```
for i, cdc in enumerate(x):
    sig[200:] = cdc
    ca = (1.0 - (dt*sig)/(2.0*epsz*eps))/(1.0 + (dt*sig)/(2.0*epsz*eps))
    ca = ca[1:-1]
    cb = (dt/epsz/eps/dx)/(1.0 + (dt*sig)/(2.0*epsz*eps))
    cb = cb[1:-1]
    ez, hy, e0z = set_zero()
    for n in range(Nt):
        fdtd(n)
    y[i] = np.argmax(e0z[200:] < ie)*dx
```

[19]:
```
logx, logy = np.log10([x,y])
fit = np.polyfit(logx, logy, 1)
yfit = 10**fit[1]*x**fit[0]
```

[20]:
```
plt.figure(figsize=(12, 6))
plt.loglog(x, y, 'o')
plt.plot(x, yfit)
plt.xlabel(r'conductivity, $\rho$')
plt.ylabel(r'skin depth, $\delta$')
plt.show()
```




**Bibliography**


- [Taflove2000] A. Taflove and S.C. Hagness, Computational Electrodynamics: The Finite-Difference Time-Domain Method, 2nd edition. 2000.
- [GedneyEE624] Gedney Stephen, University of Kentucky, EE624 Notes, Fall 2005, "Source Modeling" & "Absorbing Boundary Conditions".