KEMP User Guide & Tutorials

Measure transmittance (T), reflectance (R), and absorbance (A)

Contributors: Min Gon Lee and SeokJae Yoo
Last Update: 09/25/2015

Measure T, R, A

Contents

1) Basic setting
2) Space setting
3) Structure setting
4) Wave setting
5) FDTD setting
6) Data collection
7) Execution
8) Calcuation of T, R, A

Measure T, R, A

1) Basic Setting

```
import numpy as np
import h5py as h5
from KEMP import Basic_FDTD, Dielectric, to_epr, to_SI, to_NU
c = 299792458.
nm = 1e-9
```

- 확장자명 ".py"로 된 python 파일을 만든다
  - Windows: 메모장 혹은 python 파일을 편집할 수 있는 프로그램 사용
  - Linux: vi editor 이용하여 편집
- Python 파일은 위 코드로 시작된다
  - Numpy: 수학 계산
  - h5: 데이터 추출
  - KEMP: KEMP에서 투과, 반사, 흡수 (T, R, A) 측정에 필요한 module을 로드
  - 자주 사용하는 상수나 단위를 정의 (광속 c, 길이 nm)

Measure T, R, A

2) Space Setting

```
dx, dy, dz = [1*nm, 1*nm, 1*nm]
nx, ny, nz = [30, 30, 2000]
x = np.ones(nx, dtype=np.float64)*dx
y = np.ones(ny, dtype=np.float64)*dy
z = np.ones(nz, dtype=np.float64)*dz
space_grid = (x, y ,z)
lx, ly, lz = dx*nx, dy*ny, dz*nz
fdtd = Basic_FDTD('3D', space_grid, dtype=np.complex64, engine='intel_cpu')
pml_apply = {'x':'' , 'y':''  , 'z':'+-'  }
pbc_apply = {'x':True, 'y':True, 'z':False}
fdtd.apply_PML(pml_apply)
fdtd.apply_PBC(pbc_apply)
```

- dx, dy, dz: 한 칸의 크기
- nx, ny, dz: 총 시뮬레이션 공간의 칸 개수
  - Ex) nx=100이라면 x는 0~100까지 100+1 칸이 들어간다

Measure T, R, A

2) Space Setting

```
dx, dy, dz = [1*nm, 1*nm, 1*nm]
nx, ny, nz = [30, 30, 2000]
x = np.ones(nx, dtype=np.float64)*dx
y = np.ones(ny, dtype=np.float64)*dy
z = np.ones(nz, dtype=np.float64)*dz
space_grid = (x, y ,z)
lx, ly, lz = dx*nx, dy*ny, dz*nz
fdtd = Basic_FDTD('3D', space_grid, dtype=np.complex64, engine='intel_cpu')
pml_apply = {'x':'' , 'y':''  , 'z':'+-'  }
pbc_apply = {'x':True, 'y':True, 'z':False}
fdtd.apply_PML(pml_apply)
fdtd.apply_PBC(pbc_apply)
```

- pml_apply: PML 설정
  - 파동의 진행방향인 z축에 PML 설정 ('z':'+-')
- pbc_apply: PBC 설정
  - 나머지 x, y축에 PBC 설정 (True는 PBC 설정, False는 PBC 미지정)

Measure T, R, A

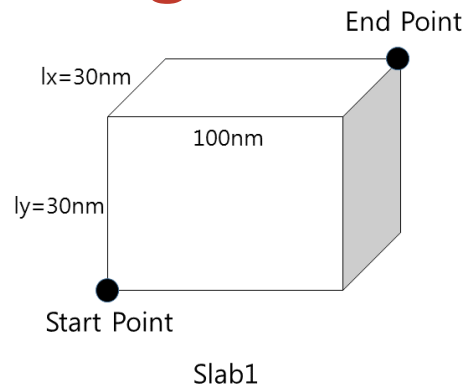
2) Space Setting

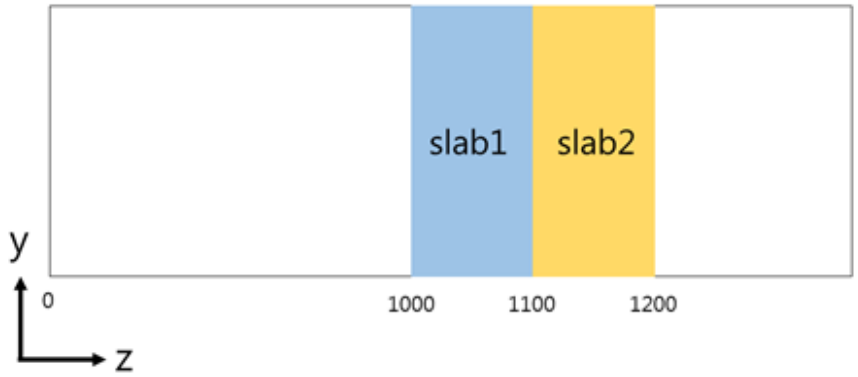
```
dx, dy, dz = [1*nm, 1*nm, 1*nm]
nx, ny, nz = [30, 30, 2000]
x = np.ones(nx, dtype=np.float64)*dx
y = np.ones(ny, dtype=np.float64)*dy
z = np.ones(nz, dtype=np.float64)*dz
space_grid = (x, y ,z)
lx, ly, lz = dx*nx, dy*ny, dz*nz
fdtd = Basic_FDTD('3D', space_grid, dtype=np.complex64, engine='intel_cpu')
pml_apply = {'x':'' , 'y':''  , 'z':'+-'  }
pbc_apply = {'x':True, 'y':True, 'z':False}
fdtd.apply_PML(pml_apply)
fdtd.apply_PBC(pbc_apply)
```

- Tip
  - x, y 축으로 PBC가 들어가므로, 빠른 계산을 위해 nx, ny 값을 낮게 잡음.
  - PML은 한 방향으로 10칸씩 들어가므로, 위 코드에서는 z축으로 0~9칸, 1991~2000칸에 PML이 설정된다.

Measure T, R, A

3) Structure Setting



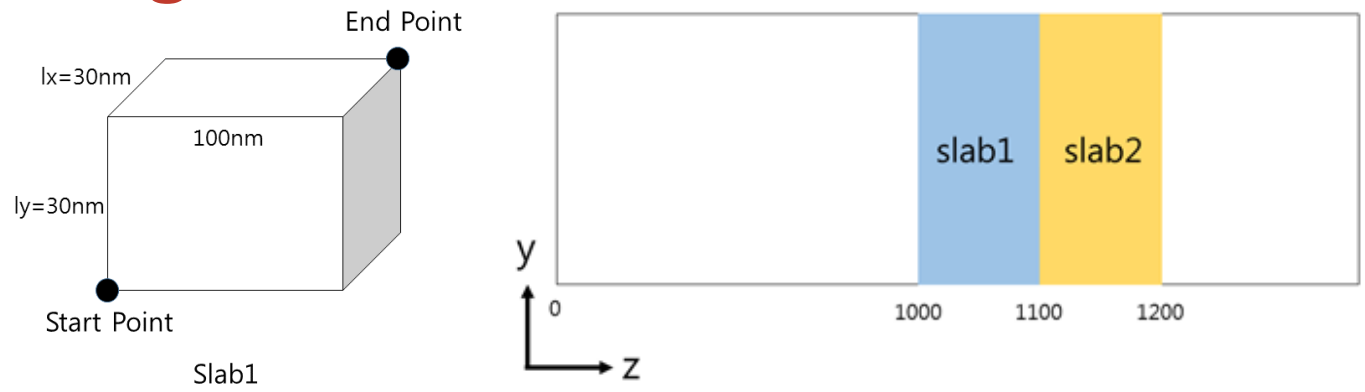



```
diel1 = Dielectric(to_epr(fdtd, n=2.))
diel2 = Dielectric(to_epr(fdtd, n=3.))
slab1 = structures.Box(diel1, ((0.*nm,0.*nm,1000.*nm-dz/2),(lx,ly,1100.*nm-dz/2)))
slab2 = structures.Box(diel2, ((0.*nm,0.*nm,1100.*nm-dz/2),(lx,ly,1200.*nm-dz/2)))
fdtd.set_structures([slab1,slab2])
```

- 이번에 사용할 두 slab은 두께가 100nm로 동일하며 각각 n=2, n=3이다.
- Slab을 만들기 위해 KEMP의 structure 구현 기능 중 가장 알맞은 것은 Box이다.

Measure T, R, A

3) Structure Setting

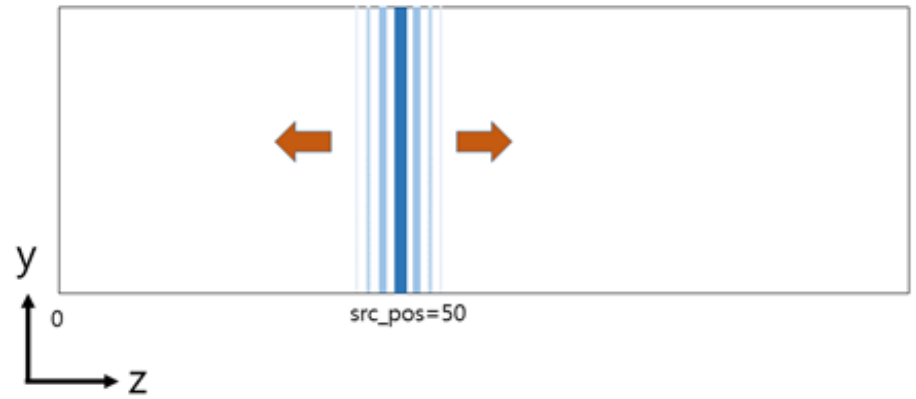



```
diel1 = Dielectric(to_epr(fdtd, n=2.))
diel2 = Dielectric(to_epr(fdtd, n=3.))
slab1 = structures.Box(diel1, ((0.*nm,0.*nm,1000.*nm-dz/2),(lx,ly,1100.*nm-dz/2)))
slab2 = structures.Box(diel2, ((0.*nm,0.*nm,1100.*nm-dz/2),(lx,ly,1200.*nm-dz/2)))
fdtd.set_structures([slab1,slab2])
```

- structures.Box(물질, (시작점),(끝점))을 적어놓으면, KEMP는 위 그림과 같은 육면체를 만든다. 여기서 시작점과 끝점의 위치는 meter 단위로 적어놓아야 하며, 이 단위계에서 다시 grid로 계산시에 오차를 줄이기 위해 한 칸이 차지하는 길이의 절반을 빼준다. fdtd.set_structures를 통해 위에서 만든 slab을 적용한다.

Measure T, R, A

4) Wave Setting

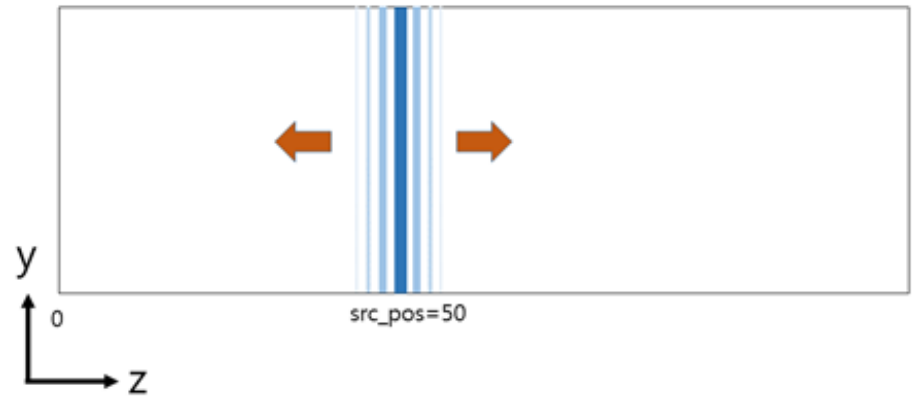



```
wavelength = np.arange(400, 801, 10.)*nm
src_pos=50
trs_pos=-50
inc    = fdtd.apply_direct_source('ey', ((0,0,src_pos),(-1,-1,src_pos)))
```

- 먼저 전자기파 spectrum 영역대를 설정한다. 여기서는 400nm부터 800nm까지 10nm 간격으로 설정하였다. 이것으로 400nm, 410nm … 800nm에서의 데이터를 얻어낼 수 있다.

Measure T, R, A

4) Wave Setting




```
wavelength = np.arange(400, 801, 10.)*nm
src_pos=50
trs_pos=-50
inc    = fdtd.apply_direct_source('ey', ((0,0,src_pos),(-1,-1,src_pos)))
```

- Plane Wave를 만들기 위해, Wave source의 z축 위치(src_pos)를 정해주고, Poynting vector를 측정하려 하는 z축 위치(trs_pos)를 정한다. 마지막으로 fdtd.apply_direct_source('polarlized electric field', ((0,0,src_pos),(-1,-1,src_pos))) 를 적어주면, z축 방향으로 진행하는 xy-plane wave가 만들어진다.

Measure T, R, A

5) FDTD Setting

```
print 'Setting Complete and READY to RUN'

tc = 500
ts = 50.
tmax = 20000

ex_src = np.zeros(tmax, dtype=np.complex64)
ey_src = np.zeros(tmax, dtype=np.complex64)
hx_src = np.zeros(tmax, dtype=np.complex64)
hy_src = np.zeros(tmax, dtype=np.complex64)
ex_trs = np.zeros(tmax, dtype=np.complex64)
ey_trs = np.zeros(tmax, dtype=np.complex64)
hx_trs = np.zeros(tmax, dtype=np.complex64)
hy_trs = np.zeros(tmax, dtype=np.complex64)

for tstep in xrange(tmax):
    src = np.exp(-(tstep-tc)**2/(ts**2))

    ex_src[tstep] = fdtd.ex[:,:,src_pos].mean()
    ey_src[tstep] = fdtd.ey[:,:,src_pos].mean()
    hx_src[tstep] = fdtd.hx[:,:,src_pos].mean()
    hy_src[tstep] = fdtd.hy[:,:,src_pos].mean()
    ex_trs[tstep] = fdtd.ex[:,:,trs_pos].mean()
    ey_trs[tstep] = fdtd.ey[:,:,trs_pos].mean()
    hx_trs[tstep] = fdtd.hx[:,:,trs_pos].mean()
    hy_trs[tstep] = fdtd.hy[:,:,trs_pos].mean()

    inc.set_source(src)
    fdtd.updateE()
    fdtd.updateH()

print('Simulation Complete')
```

- 입사할 wave는 Gaussian Pulse로 넣고, Gaussian function에서 사용할 constant를 먼저 설정하고, FDTD step(wave가 진행되는 시간)을 설정한다. 사용할 wave가 xy-plane wave이며 y-polarized electric field를 source로 쓰기 때문에, Poynting vector의 z성분만 나오게 된다. 따라서 각 step마다 저장할 x, y 성분의 field를 2개(src_pos과 trs_pos)씩 만들어준다. Step을 진행할 for문을 생성하고 wave에 사용할 gaussian function을 넣어준다. 매 step마다 저장할 field는 xy 평면에서 뽑아낸 값들의 평균값-혹은 pbc를 적용한 것을 생각해 특정 x값과 y값에 해당되는 field 값-을 사용한다. 그 다음 Wave를 넣어주고 FDTD 계산을 한다.

Measure T, R, A

6) Data Collection

```
dt_SI = to_SI(fdtd, 'time', fdtd.dt)
tsteps = np.arange(tmax, dtype=fdtd.dtype)
t = tsteps*dt_SI
freqs = c/wavelength
src = np.exp(-(tsteps-tc)**2/(ts**2))

src_ft = ((dt_SI/(2.*np.pi))*src[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
ex_src_ft = ((dt_SI/(2.*np.pi))*ex_src[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
ey_src_ft = ((dt_SI/(2.*np.pi))*ey_src[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
hx_src_ft = ((dt_SI/(2.*np.pi))*hx_src[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
hy_src_ft = ((dt_SI/(2.*np.pi))*hy_src[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
ex_trs_ft = ((dt_SI/(2.*np.pi))*ex_trs[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
ey_trs_ft = ((dt_SI/(2.*np.pi))*ey_trs[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
hx_trs_ft = ((dt_SI/(2.*np.pi))*hx_trs[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)
hy_trs_ft = ((dt_SI/(2.*np.pi))*hy_trs[np.newaxis,:]*np.exp(-2.j*np.pi*freqs[:,np.newaxis]*t[np.newaxis,:])).sum(1)

sz_ref_ft = np.zeros(len(wavelength), dtype=np.complex64)
sz_trs_ft = np.zeros(len(wavelength), dtype=np.complex64)
sz_ref_ft = ex_src_ft*np.conjugate(hy_src_ft)-(ey_src_ft-src_ft)*np.conjugate(hx_src_ft+src_ft)
sz_trs_ft = ex_trs_ft*np.conjugate(hy_trs_ft)-ey_trs_ft*np.conjugate(hx_trs_ft)

f = h5.File('./save/%s.h5' %'slab','w')
f.create_dataset('Sz_ref', data=sz_ref_ft)
f.create_dataset('Sz_trs', data=sz_trs_ft)
f.close()
```

Measure T, R, A

6) Data Collection

- FDTD를 통해 time domain 에서 얻은 데이터를 frequency domain 으로 바꾸기 위해 Fourier transform 수행
- 시간 값을 얻기 위해 FDTD에 설정된 time을 SI unit으로 변환하고, 총 step 수(tmax)를 가진 array 생성. 이를 이용해 시간에 대한 array를 만들어내고, 앞서 만든 파장 영역대의 array를 이용해 frequenc를 나타내는 array도 생성.
- Reflectance를 구하기 위해서는 slab에서 반사되는 값만 필요.
- src_pos에서 나오는 데이터는 반사되는 데이터와 source의 합이므로, source를 따로 빼주어야 한다. 따라서 이를 위한 source 값도 필요하므로, 각 step에 해당되는 source 값을 가진 array(src)를 만들어준다.
- step에 따른 field들(ex_src, ey_src, hx_src, hy_src, ex_trs, ey_trs, hx_trs, hy_trs)과 source(src)를 Fourier transform
- RT를 구하기 위해 필요한 데이터는 wavelength에 따른 poynting vector이므로, 이를 위한 array를 두개(sz_ref_ft, sz_trs_ft) 생성.
- 그 다음 얻어낸 field를 이용해 trs_pos에서의 poynting vector와, 반사되어 나온 poynting vector를 계산.
- hdf5로 poynting vector를 저장한다. 여기서 저장되는 파일이름은 'slab.h5' 이다.

Measure T, R, A

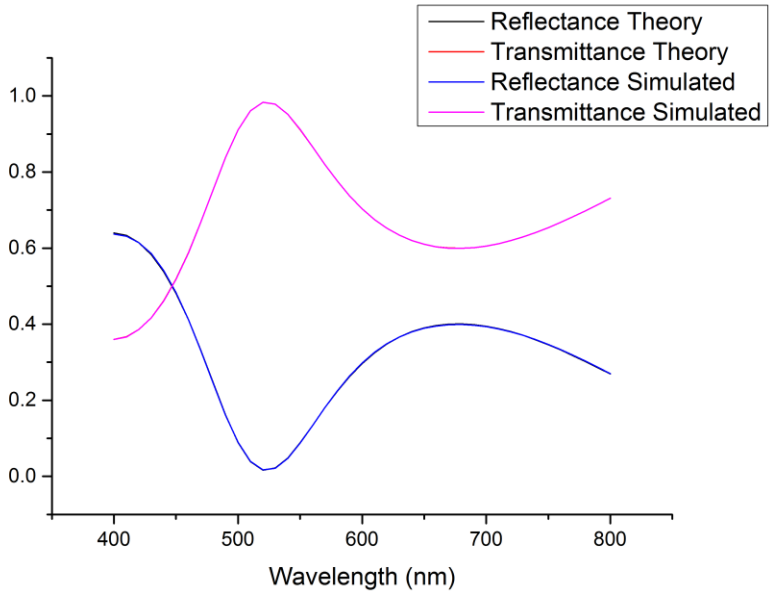
7) Execution

- Window
  - 1.py를 실행한다.

- Linux
  - Python 파일(1.py)이 있는 폴더에서 접속해 명령어 “python 1.py”로 실행한다.

Measure T, R, A

8) Calculation of T, R, A



- RT를 구하기 위해선 순수 Pulse intensity가 필요하므로, slab을 넣지 않은 상태에서 simulation을 한번 더 돌려야 한다. Slab을 없애려면 앞의 코드의 마지막 줄에서 slab1, slab2를 지워버리면 된다.
- Hdf5 저장파일 이름을 다르게 한후(예를 들어 noneslab), 앞의 과정을 반복한다. Slab이 존재할 때 얻은 데이터의 real 값을 Sz_ref1, Sz_trs1 이라 하고 slab이 없을 때 데이터의 real 값을 Sz_ref0, Sz_trs0라고 하자. 그러면 R = Sz_ref1 / Sz_trs0, T = Sz_trs1 / Sz_trs0, A = 1 – R - T 가 된다.