rcfdtdpy Documentation

Jack Roth

Jul 09, 2019

Contents:

RC-FDTD simulations are a staple of electromagnetic field simulations, and can be found in many fields and applications. This package provides a framework for performing RC-FDTD simulations aimed at investigating a particular problem: the simulation of materials that have rapidly evolving electric susceptibilities.

The scope of this problem is such that a few assumptions have been made that simplify the implementation of the simulation as well as the computational complexity of the simulation. These are as follows

- All materials are linear dielectrics such that $P(z,\omega)=\epsilon_0\chi E(z,\omega)$.
- The electric and magnetic fields are plane waves propagating along spatial coordinate z.
- Materials are uniform along spatial coordinates x and y.
- The electric and magnetic fields are zero for all time prior to the start of the simulation ($E(z,t)=0$ for all $t<0$).
- The electric field $E(z,t)$ is approximately constant over all time intervals of duration Δt.
- The magnetization of all materials is zero ($\vec{M}=\vec{0}$).

Refer to the RC-FDTD Simulations page in order to learn more about how RC-FDTD simulations work and how simulation parameters might be tweaked to produce more accurate results. For a description of various simulations run with this package refer to `this report`.

CHAPTER 1

Getting Started

rcfdtdpy can be installed into an Anaconda 3.6 environment via

```
conda install -c jr137 rcfdtdpy
```

or using pip via

```
pip install rcfdtdpy
```

From here it is easy to start one's first simulation. Perhaps we want to simulate a terahertz spectroscopy conductivity measurement of a Drude metal using the NumericMaterial class. We must first import the required libraries and define our simulation parameters.

```
# Imports
from rcfdtdpy import Simulation, Current, NumericMaterial
import numpy as np
from scipy.fftpack import fft, fftfreq
from scipy.optimize import curve_fit
from matplotlib import pyplot as plt

# Speed of light
c0 = 3e8  # m/s
# Spatial step size
di = 0.03e-6  # 0.03 um
# Temporal step size
dn = di / c0  # (0.03 um) / (3e8 m/s) = 0.1 fs
# Permittivity of free space
epsilon0 = 8.854187e-12
# Permeability of free space
mu0 = np.divide(1, np.multiply(epsilon0, np.square(c0)))
# Define simulation bounds
i0 = -1e-6  # -1 um
i1 = 1e-6  # 1 um
n0 = -0.5e-12  # -0.5 ps
n1 = 2.5e-12  # 2.5 ps
```

(continues on next page)

(continued from previous page)

```
# Calculate simulation dimensions
ilen, nlen = Simulation.calc_dims(i0, i1, di, n0, n1, dn)
# Calculate arrays that provide the spatial and temporal value of each cell
z, t = Simulation.calc_arrays(i0, i1, di, n0, n1, dn)
```

We next need to define the current present in our simulation. We do this by defining the location of the current pulse and the time at which the center of the current pulse occurs and then determining the spatial and temporal indices at which these space and time values correspond to.

```
# Define current pulse location and time
thz_loc = -0.5e-6  # -0.5 um
thz_time = 0  # 0 fs
# Find the corresponding location and time indices
thz_loc_ind = np.argmin(np.abs(np.subtract(z, thz_loc)))
thz_time_ind = np.argmin(np.abs(np.subtract(t, thz_time)))
```

We will define the terahertz pulse time profile as the second derivative of a Gaussian with FWHM of 90fs. We define this pulse as follows

```
thzshape = np.append(np.diff(np.diff(np.exp(-(((t - thz_time)/90e-15) ** 2)))), [0,␣
↪0])
```

We can now create our `Current` object

```
thzpulse = Current(thz_loc_ind, 0, ilen, nlen, thzshape)
```

Note that the length of the `thzshape` Numpy array is the same as the length of the simulation in time `nlen`. For this reason the starting spatial index of the `Current` object must be set to `0`. However the length of the `thzshape` Numpy array can be less then the length of the simulation in time. If the user is worried about the `Current` object taking up too much space in memory they might choose to define `thzshape` over a small number of indices and simply define its starting index in space and time.

We are now prepared to define our material. Like with the `Current` object we begin by defining the location and spatial and temporal extent of our material. We specify that our material starts at location 0 nm and is 50 nm thick.

```
# Set material length
material_length = 0.050e-6  # 50 nm
# Set locations
material_start = 0
material_end = material_start + material_length
# Find the corresponding location and time indices
material_ind_start = np.argmin(np.abs(np.subtract(z, material_start)))
material_ind_end = np.argmin(np.abs(np.subtract(z, material_end)))
# Determine matrix length in indices
material_ind_len = material_ind_end - material_ind_start
```

The electric susceptibility of a Drude metal in time is given by

$$\chi(t) = A\left(1 - \exp\left[-2\gamma t\right]\right)$$

If you have no idea where this definition of susceptibility comes from, read up on the Lorentz oscillator. It is very easy to implement this material into our simulation.

```
# Define constants
a = np.complex64(1e16)
gamma = np.complex64(1e12 * 2 * np.pi)
```

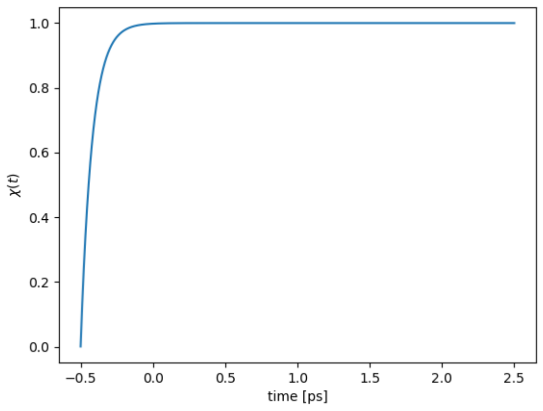
(continues on next page)

(continued from previous page)

```
# Define electric susceptibility in time
def chi(t):
    return a*(1-np.exp(-2*gamma*t))

# Define the high frequency permittivity in time (simply a constant)
def inf_perm(t):
    return 1

# Create our material!
drude = NumericMaterial(di, dn, ilen, nlen, material_ind_start, material_ind_end, chi,
↪ inf_perm)

# Export the susceptibility of the material
drude_chi = drude.export_chi()
```

Now that χ^m has been calculated for each simulation time step, we can check that our Drude material has the expected form of electric susceptibility in time. We plot the electric susceptibility versus time

```
plt.plot(t*1e12, drude_chi)
plt.xlabel('time [ps]')
plt.ylabel('$\chi(t)$')
plt.show()
```



The analytic and simulated values of $\chi(t)$ are in agreement. We must now specify what field values our simulation will record.

We would like to view our simulation evolving in time, meaning that we must store field values at each step in time. Lets say we would like to view the first third of the simulation.

```
nstore = np.arange(0, int(nlen/3), 100)
```

We choose to record the field values every 100 simulation steps for the first third of the simulation. We also would like to be able to calculate the transmission of our material in time. Therefore we wish to record the field value at every time step at the opposite side of the material from the current pulse. Since the material is 50 nm in length and starts at location 0 nm, recording the field value near the end of the simulation space will provide us with the transmitted field.

We specify that the simulation will have absorbing boundaries. The `Simulation` object is initialized and the simulation is run.

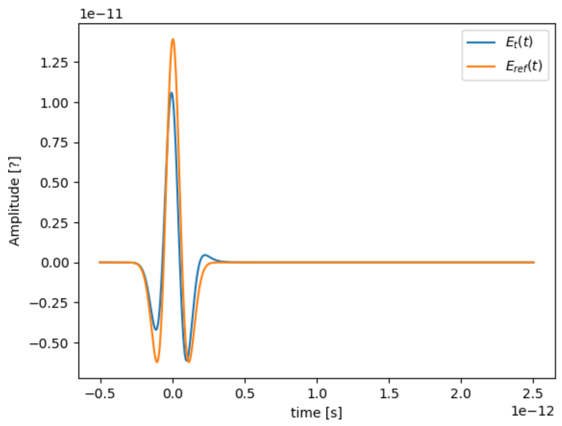
```
s = Simulation(i0, i1, di, n0, n1, dn, epsilon0, mu0, 'absorbing', thzpulse, drude,␣
↪nstore=nstore, istore=[ilen-6])
# Run simulation
s.simulate()
```

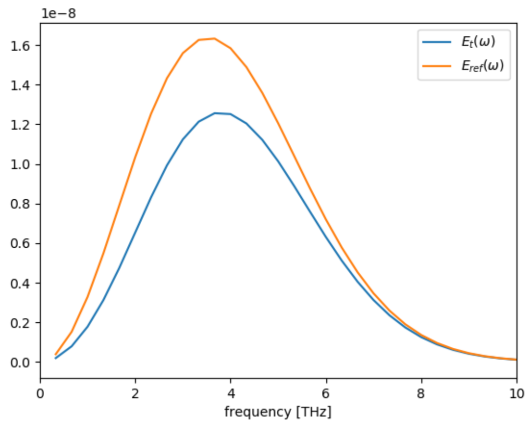
Now that the simulation has been run we export the fields stored by the `Simulation` object. The `Simulation` object simulates two sets of electric and magnetic fields: a field that interacts with materials and one that does not. This provides every simulation with a reference set of field values. We export the stored field values as well as the electric electric susceptibility.

```
# Export field values
hfield, efield, hfield_ref, efield_ref = s.export_ifields()
```

We proceed to produce plots of the transmitted and reference fields in time and frequency.

```
# Plot in time
plt.plot(t, np.real(efield), label='$E_{t}(t)$')
plt.plot(t, np.real(efield_ref), label='$E_{ref}(t)$')
plt.ylabel('Amplitude [?]')
plt.xlabel('time [s]')
plt.legend()
plt.show()

# Calculate time difference
dt = np.diff(t)[0] # Calculate time step difference in fs

# Calculate Fourier transforms
freq = fftfreq(nlen, dt) # in Hz
trans = fft(np.real(efield[:,0]))
ref = fft(np.real(efield_ref[:,0]))

# Remove unwanted frequencies
freq = freq[1:int(nlen/2)]
trans = trans[1:int(nlen/2)]
ref = ref[1:int(nlen/2)]

# Plot transformed fields
plt.plot(freq * 1e-12, np.abs(trans), label='$E_{t}(\omega)$')
plt.plot(freq * 1e-12, np.abs(ref), label='$E_{ref}(\omega)$')
plt.xlabel(r'frequency [THz]')
plt.xlim(0, 10)
plt.legend()
plt.show()
```





In the thin sample limit the conductivity of a material can be calculated via

$$\sigma(\omega) = \frac{2}{Z_0 d}\left(\frac{1}{t(\omega)} - 1\right)$$

where Z_0 is the impedance of free space and $t(\omega) = \frac{E_t(\omega)}{E_{ref}(\omega)}$. We next extract the conductivity of our simulated material and compare it to the analytical form of the conductivity of a Drude metal

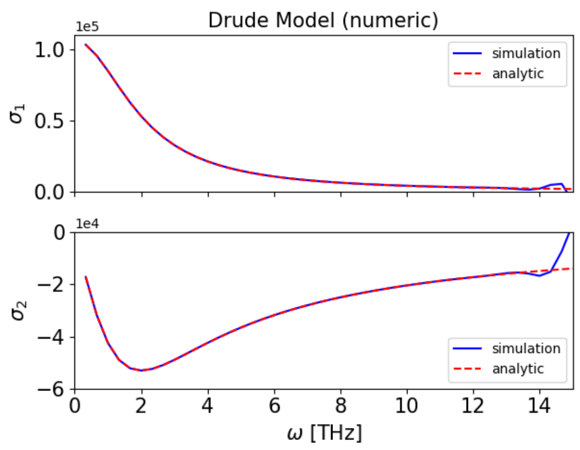
$$\sigma(\omega) = \frac{\sigma_0}{1 + i\omega\tau}$$

```
# Remove zero indicies from all arrays
nonzero_ind = np.nonzero(ref)
freq = freq[nonzero_ind]
ref = ref[nonzero_ind]
trans = trans[nonzero_ind]

# Calculate t
spec = np.divide(trans, ref)

# Set constants
Z0 = np.multiply(mu0, c0) # Ohms (impedance of free space)

# Calculate the angular frequency
ang_freq = 2 * np.pi * freq # THz * 2pi
```

(continues on next page)

(continued from previous page)

```
# Calculate conductivity
conductivity = np.multiply(np.divide(2, Z0*material_length), np.subtract(np.divide(1,␣
↪spec), 1))

# Only fit to frequencies below 14THz, as the terahertz pulse has approximately zero␣
↪amplitude above 14THz
freq_max = np.argmin(np.abs(np.subtract(14e12, freq)))

# Define fit functions
def cond_real(omega, sigma0, tau):
    return sigma0/(1+(tau*omega)**2)

def cond_imag(omega, sigma0, tau):
    return (-omega*tau*sigma0)/(1+(tau*omega)**2)

# Take real and imaginary parts
cfreq = freq[:freq_max]
creal = np.real(conductivity)[:freq_max]
cimag = np.imag(conductivity)[:freq_max]

# Run curve fit
popt_real, pcov_real = curve_fit(cond_real, cfreq, creal, p0=[1e5, 0.4e-12])
popt_imag, pcov_imag = curve_fit(cond_imag, cfreq, cimag, p0=[1e5, 0.2e-12])

fit_real = cond_real(freq, *popt_real)
fit_imag = cond_imag(freq, *popt_imag)

# Setup plot
fig, (ax0, ax1) = plt.subplots(2, 1, sharex=True, dpi=100)
ax0.set_ylabel(r'$\sigma_1$', fontsize=15)
ax1.set_ylabel(r'$\sigma_2$', fontsize=15)
ax1.set_xlabel(r'$\omega$ [THz]', fontsize=15)
ax0.set_title(r'Drude Model (numeric)', fontsize=15)
ax1.set_xlim(0, 15)
ax0.ticklabel_format(style='sci', scilimits=(0,0), axis='y')
ax0.tick_params(labelsize=15)
ax0.set_ylim(0, 1.1e5)
ax1.ticklabel_format(style='sci', scilimits=(0,0), axis='y')
ax1.tick_params(labelsize=15)
ax1.set_ylim(-6e4, 0)

# Plot simulated conductivity
ax0.plot(freq*1e-12, np.real(conductivity), 'b-', label='simulation')
ax1.plot(freq*1e-12, np.imag(conductivity), 'b-', label='simulation')

# Plot analytic conductivity
ax0.plot(freq*1e-12, fit_real, 'r--', label='analytic')
ax1.plot(freq*1e-12, fit_imag, 'r--', label='analytic')

ax0.legend()
ax1.legend()

plt.tight_layout()

plt.show()
```



That's it! We have successfully simulated a Drude metal and examined how simulations are run with rcfdtdpy! You can download the complete `start.py` file.

CHAPTER 2

RC-FDTD Simulations

Recursive convolution finite difference time domain (RC-FDTD) simulations have long been used to numerically solve Maxwell's equations. This simulation technique discretizes the time domain and evolves the electric and magnetic fields in time using a set of update equations. Within the simulation, space is discretized into intervals of length Δz and time into intervals of length Δt. A specific point in time and space is accessed via $z = i\Delta z$ and $t = n\Delta t$. The simulation relies on a number of assumptions:

- All materials are linear dielectrics such that $P(z,\omega) = \epsilon_0 \chi E(z,\omega)$.
- The electric and magnetic fields are plane waves propagating along spatial coordinate z.
- Materials are uniform along spatial coordinates x and y.
- The electric and magnetic fields are zero for all time prior to the start of the simulation ($E(z,t) = 0$ for all $t < 0$).
- The electric field $E(z,t)$ is approximately constant over all time intervals of duration Δt.
- The magnetization of all materials is zero ($\vec{M} = \vec{0}$).

These assumptions allow the derivation of the discretized displacement field $D^{i,n}$. The displacement field $\vec{D}(\vec{r},\omega)$, with the requirement that simulated materials are linear dielectrics such that $P(z,\omega) = \epsilon_0 \chi(z,\omega) E(z,\omega)$ and the requirement that the field varies over only the spatial coordinate z we find that $D(z,\omega)$ is

$$D(z,\omega) = \epsilon_0 \left[1 + \chi(z,\omega)\right] E(z,\omega)$$

The displacement field $D(z,\omega)$ can be transformed to the time domain via

$$\begin{aligned} D(z,t) =& \mathcal{F}^{-1}\left\{D(z,\omega)\right\} \\ =& \mathcal{F}^{-1}\left\{\epsilon_0 \left[1+\chi(\omega)\right] E(z,\omega)\right\} \\ =& \mathcal{F}^{-1}\left\{\epsilon_0 \mathcal{F}\left\{1+\chi(t)\right\} \mathcal{F}\left\{E(z,t)\right\}\right\} \end{aligned}$$

where $\mathcal{F}\left\{a(t)\right\}$ and $\mathcal{F}^{-1}\left\{a(\nu)\right\}$ to denote Fourier and inverse Fourier transforms. Thus via the convolution theorem

$$\begin{aligned} D(z,t) &= \mathcal{F}^{-1}\left\{\epsilon_0 \mathcal{F}\left\{1+\chi(t)\right\} \mathcal{F}\left\{E(z,t)\right\}\right\} \\ &= \epsilon_0 \left[1+\chi(t)\right] * \left[E(z,t)\right] \\ &= \epsilon_0 \left[\epsilon_\infty E(z,t) + \int_0^t \chi(\tau) E(z,t-\tau) d\tau\right] \end{aligned}$$

where $*$ denotes a convolution. It is assumed that $E(z,t)=0$ for all $t<0$. We discretize this result by replacing the z and t coordinates via $z=i\Delta z$ and $t=n\Delta t$ where $i,n\in\mathbb{R}$, yielding

$$\begin{aligned}D(i\Delta z,n\Delta t)=&\epsilon_0\epsilon_\infty E(i\Delta z,n\Delta t)\\&+\epsilon_0\int_0^{n\Delta t}\chi(\tau)E(i\Delta z,n\Delta-\tau)d\tau\end{aligned}$$

Assuming that $E(i\Delta z,n\Delta-\tau)$ is constant over all time intervals of duration Δt the integral is replaced with a sum

$$D^{i,n}=\epsilon_0\epsilon_\infty E^{i,n}+\epsilon_0\sum_{m=0}^{n-1}E^{i,n-m}\chi^m$$

where

$$\chi^m=\int_{m\Delta t}^{(m+1)\Delta t}\chi(\tau)d\tau$$

It is *not* assumed $\chi(t)$ is constant over any time interval. This result is consistent with the result derived in Luebbers et al. and Beard et al..

$$D^{i,n}=\epsilon_0\epsilon_\infty E^{i,n}+\epsilon_0\sum_{m=0}^{n-1}E^{i,n-m}\chi^m$$

where

$$\chi^m=\int_{m\Delta t}^{(m+1)\Delta t}\chi(\tau)d\tau$$

This result is significant for the RC-FDTD simulation framework implmented here as it means a material can be simulated as long as one can define $chi(t)$ for that material.

We proceed by deriving the update equations for the electric and magnetic fields. With the requirement that $\vec{M}=\vec{0}$ and the requirement that the electric and magnetic fields are uniform in spatial coordinates x and y, Faraday's law of induction and Ampere's law with Maxwell's addition reduce to

$$\frac{\partial E}{\partial z}=-\mu_0\frac{\partial H}{\partial t}\qquad-\frac{\partial H}{\partial z}=I_f+\frac{\partial D}{\partial t}$$

where I_f is along $\hat{z}$. Noting the definition of a derivative we find

$$\begin{aligned}\lim_{\Delta z\to0}\frac{E(z+\Delta z,t)-E(z,t)}{\Delta z}&=-\mu_0\lim_{\Delta t\to0}\frac{H(z,t+\Delta t)-H(z,t)}{\Delta t}\\-\lim_{\Delta z\to0}\frac{H(z+\Delta z,t)-H(z,t)}{\Delta z}&=I_f+\lim_{\Delta t\to0}\frac{D(z,t+\Delta t)-D(z,t)}{\Delta t}\end{aligned}$$

From here the discretization process is simple. We simply remove each limit from the equations, define an appropriate value of Δz and Δt, and replace the fields with their discretized forms.

$$\begin{aligned}\frac{E^{i+1,n}-E^{i,n}}{\Delta z}&=-\mu_0\frac{H^{i,n+1}-H^{i,n}}{\Delta t}\\-\frac{H^{i+1,n}-H^{i,n}}{\Delta z}&=I_f+\frac{D^{i,n+1}-D^{i,n}}{\Delta t}\end{aligned}$$

If Δz and Δt aren't small enough such that the derivative is accurate then the RC-FDTD simulation will break down.

We solve Eq.(ref{eq:faraday}) for $H^{i,n+1}$, finding the following update equation

$$H^{i,n+1}=H^{i,n}-\frac{1}{\mu_0}\frac{\Delta t}{\Delta z}\left[E^{i+1,n}-E^{i,n}\right]$$

In order to solve Eq.(ref{eq:ampere}) we use the result of Eq.(ref{eq:disp}) to determine $D^{i,n+1} - D^{i,n}$ in terms of $E^{i+1,n}$ and $E^{i,n}$

$$\begin{aligned} D^{i,n+1} - D^{i,n} &= \epsilon_0\epsilon_\infty E^{i,n+1} + \epsilon_0 \sum_{m=0}^{n} E^{i,n+1-m}\chi^m - \epsilon_0\epsilon_\infty E^{i,n} - \epsilon_0 \sum_{m=0}^{n-1} E^{i,n-m}\chi^m \\ &= \epsilon_0\epsilon_\infty \left[E^{i,n+1} - E^{i,n}\right] + \epsilon_0 \left[\sum_{m=0}^{n} E^{i,n+1-m}\chi^m - \sum_{m=0}^{n-1} E^{i,n-m}\chi^m\right] \end{aligned}$$

Noting that

$$\begin{aligned} \sum_{m=0}^{n} E^{i,n+1-m}\chi^m - \sum_{m=0}^{n-1} E^{i,n-m}\chi^m &= E^{i,n+1}\chi^0 + \sum_{m=1}^{n} E^{i,n+1-m}\chi^m - \sum_{m=0}^{n-1} E^{i,n-m}\chi^m \\ &= E^{i,n+1}\chi^0 + \sum_{m=0}^{n-1} E^{i,n+1-(m+1)}\chi^{m+1} - \sum_{m=0}^{n-1} E^{i,n-m}\chi^m \\ &= E^{i,n+1}\chi^0 + \sum_{m=0}^{n-1} E^{i,n-m}\left[\chi^{m+1} - \chi^m\right] \end{aligned}$$

and letting

$$\begin{aligned} \Delta\chi^m &= \chi^m - \chi^{m+1} \\ \psi^n &= \sum_{m=0}^{n-1} E^{i,n-m}\Delta\chi^m \end{aligned}$$

we find

$$\begin{aligned} D^{i,n+1} - D^{i,n} &= \epsilon_0\epsilon_\infty \left[E^{i,n+1} - E^{i,n}\right] + \epsilon_0 \left[E^{i,n+1}\chi^0 - \psi^n\right] \\ &= \epsilon_0 \left[\epsilon_\infty + \chi^0\right] E^{i,n+1} - \epsilon_0\epsilon_\infty E^{i,n} - \epsilon_0\psi^n \end{aligned}$$

Substituting this result into Eq.(ref{eq:ampere}) and solving for $E^{i,n+1}$ we find

$$\begin{aligned} E^{i,n+1} =& \frac{\epsilon_\infty}{\epsilon_\infty + \chi^0} E^{i,n} + \frac{1}{\epsilon_\infty + \chi^0}\psi^n - \frac{\Delta t I_f}{\epsilon_0\left[\epsilon_\infty + \chi^0\right]} \\ & - \frac{1}{\epsilon_0\left[\epsilon_\infty + \chi^0\right]}\frac{\Delta t}{\Delta z}\left[H^{i+1,n} - H^{i,n}\right] \end{aligned}$$

We then implement the Yee cell in the simulation by offsetting the electric and magnetic field cells by half a spatial and temporal incrementcite{beard}, producing

$$\begin{aligned} H^{i+1/2,n+1/2} =& H^{i+1/2,n-1/2} - \frac{1}{\mu_0}\frac{\Delta t}{\Delta z}\left[E^{i+1,n} - E^{i,n}\right] \\ E^{i,n+1} =& \frac{\epsilon_\infty}{\epsilon_\infty + \chi^0} E^{i,n} + \frac{1}{\epsilon_\infty + \chi^0}\psi^n - \frac{\Delta t I_f}{\epsilon_0\left[\epsilon_\infty + \chi^0\right]} \\ & - \frac{1}{\epsilon_0\left[\epsilon_\infty + \chi^0\right]}\frac{\Delta t}{\Delta z}\left[H^{i+1/2,n+1/2} - H^{i-1/2,n+1/2}\right] \end{aligned}$$

where

$$\begin{aligned} \Delta\chi^m &= \chi^m - \chi^{m+1} \\ \psi^n &= \sum_{m=0}^{n-1} E^{i,n-m}\Delta\chi^m \end{aligned}$$

The accuracy of the derivative approximation inherent to these update equations relies on choosing some Δz and Δt small enough such that the electric and magnetic fields are approximately linear over spatial intervals Δz and time intervals Δt. If this condition is not met then the accuracy of the derivative approximation breaks down. The update equations derived here are significant as they reveal that any linear dielectric can be accurately simulated via the RC-FDTD method as long as the electric susceptibility of the material $\chi(t)$ is well defined. We turn our attention to modeling the electric susceptibility of materials in section ref{sec:susceptibility}.

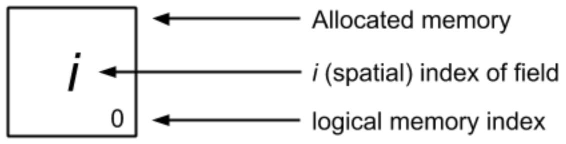
rcfdtdpy provides a framework in which the user need only provide the electric susceptibility $\chi(t)$ to run a simulation.

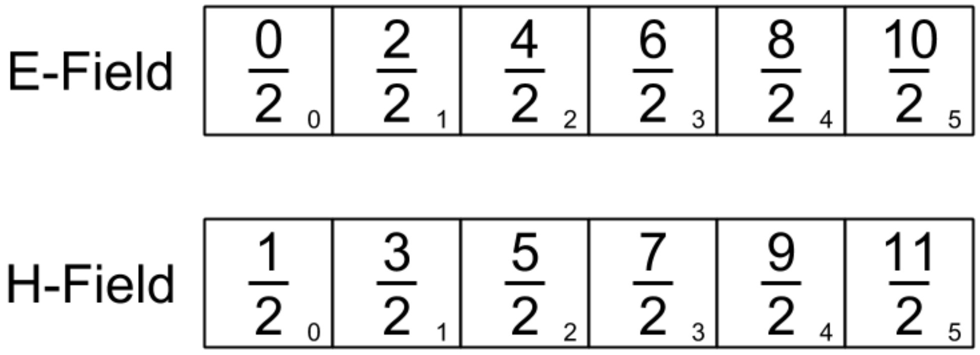
CHAPTER 3

Reference

3.1 Simulation

The `Simulation` class treats field indices as follows





class `rcfdtdpy.sim.`**`Simulation`**(*i0*, *i1*, *di*, *n0*, *n1*, *dn*, *epsilon0*, *mu0*, *boundary*, *currents=[]*, *materials=[]*, *nstore=[]*, *istore=[]*)

Represents a single simulation. Field is initialized to all zeros.

Parameters

- **`i0`** – The spatial value at which the field starts
- **`i1`** – The spatial value at which the field ends
- **`di`** – The spatial step size

- **`n0`** – The temporal value at which the field starts
- **`n1`** – The temporal value at which the field ends
- **`dn`** – The temporal step size
- **`epsilon0`** – ϵ_0, the vacuum permittivity
- **`mu0`** – μ_0, the vacuum permeability
- **`boundary`** – The boundary type of the field, either 'zero', for fields bounded by zeros or 'absorbing' for absorbing boundary conditions
- **`currents`** – A Current object or a list of Current objects that represent the currents present in the simulation, defaults to none
- **`materials`** – A Material object or a list of Material objects that represent the materials present in the simulation, defaults to none
- **`nstore`** – A list of time indices to save field values at in all points in space
- **`istore`** – A list of spatial indices to save field values at in all points in time

`static calc_arrays`(*i0*, *i1*, *di*, *n0*, *n1*, *dn*)

Calculates spatial and time arrays of the same dimensions of the simulation. Array values are populated by their the spatial and temporal values at their respective simulation spatial and temporal indices.

Parameters

- **`i0`** – The spatial value at which the field starts
- **`i1`** – The spatial value at which the field ends
- **`di`** – The spatial step size
- **`n0`** – The temporal value at which the field starts
- **`n1`** – The temporal value at which the field ends
- **`dn`** – The temporal step size

Returns A tuple *(z, t)* of the spatial and temporal arrays

`static calc_dims`(*i0*, *i1*, *di*, *n0*, *n1*, *dn*)

Calculates the dimensions of the simulation in cells.

Parameters

- **`i0`** – The spatial value at which the field starts
- **`i1`** – The spatial value at which the field ends
- **`di`** – The spatial step size
- **`n0`** – The temporal value at which the field starts
- **`n1`** – The temporal value at which the field ends
- **`dn`** – The temporal step size

Returns A tuple *(ilen, nlen)* of the spatial and temporal dimensions

`export_ifields`()

Exports the field values at spatial indices specified by istore at the Simulation object's initialization, where an index along axis=1 corresponds to the corresponding spatial index in istore. Values along axis=0 correspond to the temporal index.

Returns *(hfield, efield, hfieldr, efieldr)* where the suffix *r* corresponds to a reference field. If istore is unspecified returns None

`export_nfields``()`
Exports the field values at temporal indices specified by nstore at the Simulation object's initialization, where an index along axis=0 corresponds to the corresponding temporal index in nstore. Values along axis=1 correspond to the spatial index.

Returns *(hfield, efield, hfieldr, efieldr)* where the suffix *r* corresponds to a reference field. If nstore is unspecified returns None

`get_dims``()`
Returns the dimensions of the simulation.

Returns A tuple `(ilen, nlen)` containing the spatial and temporal dimensions in cells

`get_materials``()`
Returns the list of Material objects present in the simulation.

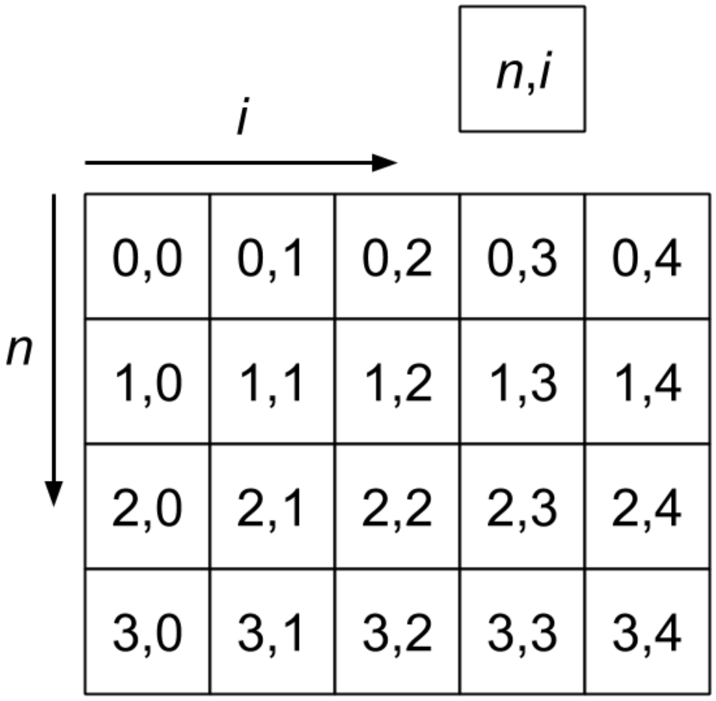
Returns A list of Material objects

`simulate``(`*tqdmarg={}*`)`
Executes the simulation.

Parameters **`tqdmarg`** – The arguments to pass the tdqm iterator (lookup arguments on the tqdm documentation)

3.2 Current

The `Current` class is used to represent currents in the simulation. The `current` argument passed to the `Current` object at initialization is represented using the following two dimensional array



where n represents the time index of the current and i represents the spatial index of the current.

class `rcfdtdpy.sim.`**`Current`**(*i0*, *n0*, *ilen*, *nlen*, *current*)

The Current class is used to represent a current in the simulation.

Parameters

- **`i0`** – The starting spatial index of the current
- **`n0`** – The starting temporal index of the current
- **`ilen`** – The number of spatial indices in the simulation
- **`nlen`** – The number of temporal indices in the simulation
- **`current`** – A matrix representing the current, where axis=0 represents locations in time n and axis=1 represents locations in space i

`get_current`(*n*)

Returns the current at time index n as an array the length of the simulation

3.3 Materials

There are a number of material types that have been implemented for use with the RC-FDTD simulation.

3.3.1 Material

The `Material` class is an abstract class containing the properties and functions that any material implementation should have. Any subclass of material should call to super at initialization and implement each function defined here.

class `rcfdtdpy.sim.`**`Material`**(*di*, *dn*, *ilen*, *nlen*, *material_i0*, *material_i1*, *material_n0*, *material_n1*)

The Material class is an abstract class that defines the minimum requirements for a Material object to have in the simulation. All Materials in the simulation must inherit Material.

Parameters

- **`di`** – The spatial time step of the simulation
- **`dn`** – The temporal step size of the simulation
- **`ilen`** – The number of spatial indices in the simulation
- **`nlen`** – The number of temporal indices in the simulation
- **`material_i0`** – The starting spatial index of the material
- **`material_i1`** – The ending spatial index of the material
- **`material_n0`** – The starting temporal index of the material
- **`material_n1`** – The ending temporal index of the material

`get_chi0`()

Returns the value of χ_0 at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of χ_0 at each spatial location in the simulation

`get_epsiloninf`()

Returns the value of ϵ_∞ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ϵ_∞ at each spatial location in the simulation

`get_psi`()

Returns the value of ψ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ψ at each spatial location in the simulation

`reset_material`()

This function is called before each simulation. It should reset any material values that are calculated during the simulation to their initial values.

`update_material`(*n*, *efield*)

This function is called at the start of each simulation time step. It should update the χ and ψ values of the material to their values at n.

Parameters

- **`n`** – The current temporal index of the simulation.
- **`efield`** – The previous electric field of the simulation.

3.3.2 EmptyMaterial

The `EmptyMaterial` class represents vacuum

class `rcfdtdpy.sim.`**`EmptyMaterial`**(*di*, *dn*, *ilen*, *nlen*)

Represents an empty Material, i.e. vacuum

get_chi0()
Returns the value of χ_0 at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of χ_0 at each spatial location in the simulation

get_epsiloninf()
Returns the value of ϵ_∞ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ϵ_∞ at each spatial location in the simulation

get_psi()
Returns the value of ψ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ψ at each spatial location in the simulation

reset_material()
This function is called before each simulation. It should reset any material values that are calculated during the simulation to their initial values.

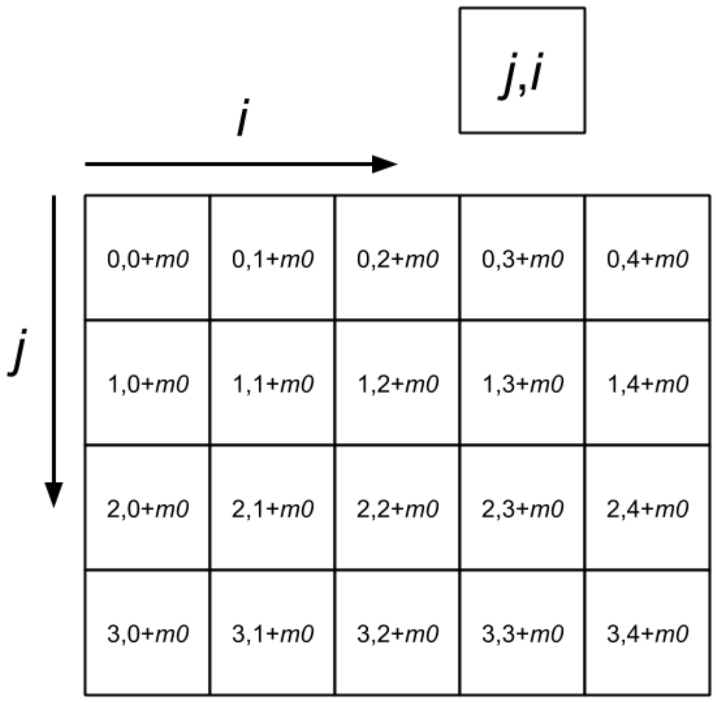
update_material(*n*, *efield*)
This function is called at the start of each simulation time step. It should update the χ and ψ values of the material to their values at n.

Parameters

- **n** – The current temporal index of the simulation.
- **efield** – The previous electric field of the simulation.

3.3.3 StaticMaterial

The `StaticMaterial` class is an implementation of the `Material` class. The variables A_1, A_2, γ, and β are represented using the following two dimensional array



where m_0 is the starting index of the material. Increments along the vertical axis represent increments in the oscillator index j, and increments along the horizontal axis represent increments in space.

The `StaticMaterial` class uses the update equations for ψ^n derived in Beard et al..

class `rcfdtdpy.sim.`**`StaticMaterial`**(*di*, *dn*, *ilen*, *nlen*, *material_i0*, *epsiloninf*, *a1*, *a2*, *g*, *b*, *opacity=None*, *istore=[]*)

The StaticMaterial class allows for the simulation of a static material, that is a material that has a constant definition of electric susceptibility in time. The electric susceptibility is modeled using a harmonic oscillator. This material is more computationally efficient than NumericMaterial, and uses the update equations specified in Beard et al..

Parameters

- **`di`** – The spatial time step of the simulation
- **`dn`** – The temporal step size of the simulation
- **`ilen`** – The number of spatial indices in the simulation
- **`nlen`** – The number of temporal indices in the simulation
- **`material_i0`** – The starting spatial index of the material
- **`epsiloninf`** – The ϵ_∞ of the material, which is constant over space and time.
- **`a1`** – A matrix representing A_1 where axis=0 represents the j th oscillator and axis=1 represents the i th spatial index

- **a2** – A matrix representing A_2 where axis=0 represents the j th oscillator and axis=1 represents the i th spatial index
- **g** – A matrix representing γ where axis=0 represents the j th oscillator and axis=1 represents the i th spatial index
- **b** – A matrix representing eta where axis=0 represents the j th oscillator and axis=1 represents the i th spatial index
- **opacity** – A vector representing the opacity of the material in time. Each index corresponds to the n th time index of the material where *1* corresponds to the material being opaque and *0* corresponds to the material being transparent. Values can be real numbers. Defaults to an opaque material for all time.
- **istore** – A list of spatial indices to save χ values at in all points in time

export_ifields()

Exports the χ values at spatial indices specified by istore at the Material object's initialization, where an index along axis=1 corresponds to the corresponding spatial index in istore. Values along axis=0 correspond to the temporal index.

Returns *chi* or None if istore is unspecified.

get_chi0()

Returns the value of χ^0 at all spatial indices resulting from the material.

Returns χ^0 at all values of the simulation resulting from the material.

get_epsiloninf()

Returns the value of ϵ_∞ at all spatial indices resulting from the material.

Returns ϵ_∞ at all values of the simulation resulting from the material.

get_psi()

Calculates psi at all points in the simulation using the current value of psi_1 and psi_2. Scaled by the *opacity* array passed in at initialization.

Returns ψ^n at all values of the simulation resulting from the material where n is given by the number of times *update_material* has been called by the *Material*'s corresponding *Simulation* object, which is once per simulation step.

reset_material()

This function is called before each simulation. It should reset any material values that are calculated during the simulation to their initial values.

update_material(*n*, *efield*)

Updates the values of ψ and χ Saves the values of chi requested via the *istore* parameter.

Parameters

- **n** – The iteration index n
- **efield** – The efield to use in update calculations

3.3.4 NumericMaterial

The `NumericMaterial` class is an implementation of the `Material` class designed to simulate a material with any arbitrary definition of electric susceptibility $\chi(t)$. The value of χ^m at each simulation step is calculated at the initialization of the `NumericMaterial`.

class rcfdtdpy.sim.**NumericMaterial**(*di*, *dn*, *ilen*, *nlen*, *material_i0*, *material_i1*, *chi_func*, *epsiloninf_func*, *tqdmarg={}*)

The NumericMaterial class represents a material that has a non-constant definition of electric susceptibility in time. Currently this Material can only represent materials that have a constant definition of electric susceptibility in space.

Parameters

- **di** – The spatial time step of the simulation
- **dn** – The temporal step size of the simulation
- **ilen** – The number of spatial indices in the simulation
- **nlen** – The number of temporal indices in the simulation
- **material_i0** – The starting spatial index of the material
- **material_i1** – The ending spatial index of the material
- **chi_func** – A function representing the electric susceptibility χ as a function of time. The function should accept a single argument n corresponding to the time index.
- **epsiloninf** – A function representing the ϵ_∞ of the material as a function of time. The function should accept a single argument n corresponding to the time index.
- **tqdmarg** – The arguments to pass the tdqm iterator (lookup arguments on the tqdm documentation).

export_chi()

Exports the electric susceptibility χ at each time index.

Returns The electric susceptibility χ

get_chi0()

Returns the value of χ_0 at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of χ_0 at each spatial location in the simulation

get_epsiloninf()

Returns the value of ϵ_∞ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ϵ_∞ at each spatial location in the simulation

get_psi()

Returns the value of ψ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ψ at each spatial location in the simulation

reset_material()

This function is called before each simulation. It should reset any material values that are calculated during the simulation to their initial values.

update_material(*n*, *efield*)

This function is called at the start of each simulation time step. It should update the χ and ψ values of the material to their values at n.

Parameters

- **n** – The current temporal index of the simulation.
- **efield** – The previous electric field of the simulation.

3.3.5 TwoStateMaterial

The `TwoStateMaterial` class is possibly unstable. Its use is not recommended.

class `rcfdtdpy.sim.`**`TwoStateMaterial`**(*di*, *dn*, *ilen*, *nlen*, *n_ref_ind*, *material_i0*, *e_a1*, *e_a2*, *e_b*, *e_g*, *g_a1*, *g_a2*, *g_b*, *g_g*, *v_a1*, *v_a2*, *v_b*, *v_g*, *alpha*, *Gamma*, *t_diff*, *tau*, *b*, *epsiloninf*, *tqdmarg_f={}*, *tqdmarg_c={}*)

The TwoStateMaterial class represents a material that has a non-constant definition of electric susceptibility in time. This material is represents the same material proposed in Beard and Schmuttenmaer 2001 (www.doi.org/10.1063/1.1338526). The ground state electric susceptibility oscillator is defined by

$$\chi_{g,j}(t) = e^{-\gamma_{g,j}t}\left[A_{g,1,j}e^{\beta_{g,j}t} + A_{g,2,j}e^{-\beta_{g,j}t}\right]$$

and the excited state electric susceptibility oscillator is defined by

$$\chi_{e,j}(t) = e^{-\gamma_{e,j}t}\left[A_{e,1,j}e^{\beta_{e,j}t} + A_{e,2,j}e^{-\beta_{e,j}t}\right]$$

There is also an visual electric susceptibility oscillator that represents any susceptibility change resulting directly from the incident visual pulse. This is defined by

$$\chi_{\text{vis},j}(t) = e^{-\gamma_{\text{vis},j}t}\left[A_{\text{vis},1,j}e^{\beta_{\text{vis},j}t} + A_{\text{vis},2,j}e^{-\beta_{\text{vis},j}t}\right]$$

The total susceptibility is defined as

$$\chi(t,t'',z) = \sum_j\left[f_e(t,t'',z)\chi_{e,j}(t) + (1-f_e(t,t'',z))\chi_{g,j}(t) + g(t,t'',z)\chi_{\text{vis},j}(t)\right]$$

where $f_e(t,t'',z)$ the the fraction of excited oscillators and $g(t,t'',z)$ is the incident visual pulse. We define $f_e(t,t'',z)$ as a decay as a function of distance into the material times the convolution of the visual pulse and the excited state decay

$$f_e(t,t'',z) = \exp\left(-\alpha z\right)\int_0^t \exp\left[-\left(\frac{t''-t'}{\Gamma}\right)^2\right]\left(\exp\left[-\frac{t'}{\tau}\right]+b\right)\mathrm{d}t'$$

where α is the spatial decay of the visual pulse in the material, Γ is the width of the visual pulse, t'' is the time delay between the visual and simulation pulses, τ is the time decay of an excited oscillator, b is the offset that the excited oscillators have (used if not all excited oscillators ultimately de-excite), and $t' = t - zv_{\text{vis}}$. We define $g(t,t'',z)$ as the visual pulse multiplied by the spatial decay of the pulse in the material

$$g(t,t'',z) = \exp\left(-\alpha z\right)\exp\left[-\left(\frac{t''-t'}{\Gamma}\right)^2\right]$$

The user supplies α, Γ, t'', τ, and b at the simulation initialization to define $f_e(t,t'',z)$ and $g(t,t'',z)$. The user must also supply the constants that define the ground, excited, and visual oscillators. Initializing the material will trigger the computation of the fraction of excited oscillators followed by the computation of the discretized electric susceptibility.

Parameters

- **`di`** – The spatial time step of the simulation
- **`dn`** – The temporal step size of the simulation
- **`ilen`** – The number of spatial indices in the simulation
- **`nlen`** – The number of temporal indices in the simulation
- **`n_ref_ind`** – The time index to use as a reference for the the *t_diff* argument
- **`material_i0`** – The starting spatial index of the material
- **`e_a1`** – The excited state oscillator $A_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.

  - **e_a2** – The excited state oscillator $A_{e,2}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **e_b** – The excited state oscillator $\beta_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **e_g** – The excited state oscillator $\gamma_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **g_a1** – The ground state oscillator $A_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **g_a2** – The ground state oscillator $A_{e,2}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **g_b** – The ground state oscillator $\beta_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **g_g** – The ground state oscillator $\gamma_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **v_a1** – The visual oscillator $A_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **v_a2** – The visual oscillator $A_{e,2}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **v_b** – The visual oscillator $\beta_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **v_g** – The visual oscillator $\gamma_{e,1}$, where axis=0 represents the j th oscillator and axis=1 represents the i th spatial oscillator in the material.
  - **alpha** – The spatial decay constant α in meters
  - **Gamma** – The visual pulse width Γ in seconds
  - **t_diff** – The time difference t'' in seconds between the visual pulse and index *n_ref_ind*
  - **tau** – The oscillator time decay constant τ in seconds
  - **b** – The excited oscillator decay offset_

- **epsiloninf** – The ϵ_∞ of the material, which is constant over space and time.
- **tqdmarg_f** – The arguments to pass the tdqm iterator during the fraction of excited oscillators calculation (lookup arguments on the tqdm documentation).
- **tqdmarg_c** – The arguments to pass the tdqm iterator during the electric susceptibility calculation (lookup arguments on the tqdm documentation).

get_chi0()

Returns the value of χ_0 at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of χ_0 at each spatial location in the simulation

get_epsiloninf()

Returns the value of ϵ_∞ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ϵ_∞ at each spatial location in the simulation

get_psi()

Returns the value of ψ at the current time step in the simulation at each spatial location in the simulation.

Returns The current value of ψ at each spatial location in the simulation

`reset_material`()
This function is called before each simulation. It should reset any material values that are calculated during the simulation to their initial values.

`update_material`(*n*, *efield*)
This function is called at the start of each simulation time step. It should update the χ and ψ values of the material to their values at n.

Parameters

- **`n`** – The current temporal index of the simulation.
- **`efield`** – The previous electric field of the simulation.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index