Para $\rho = 0$ and $\mathbf{J} = 0$ las leyes de Faraday y Ampère quedan:

$$-\frac{1}{\mu_0\mu}\oint_C \mathbf{E}(\mathbf{r},t)\cdot d\mathbf{s} = \frac{\partial}{\partial t}\int_A \mathbf{H}(\mathbf{r},t)\cdot \mathbf{n}da$$

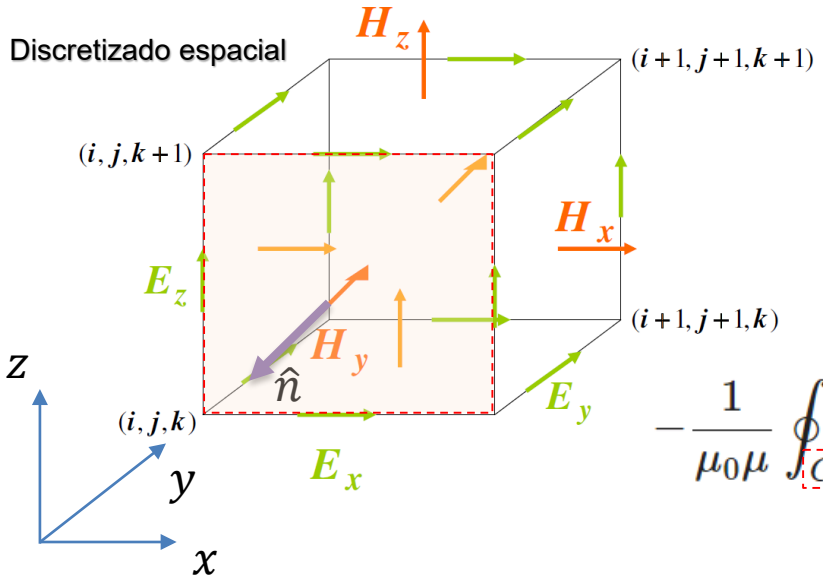
$$\frac{1}{\varepsilon_0\varepsilon}\oint_C \mathbf{H}(\mathbf{r},t)\cdot d\mathbf{s} = \frac{\partial}{\partial t}\int_A \mathbf{E}(\mathbf{r},t)\cdot \mathbf{n}da$$

donde se han utilizado las relaciones constitutivas en un medio i.h.l

$$\mathbf{D}(\mathbf{r},t) = \varepsilon_0\varepsilon\mathbf{E}(\mathbf{r},t), \quad \mathbf{B}(\mathbf{r},t) = \mu_0\mu\mathbf{H}(\mathbf{r},t)$$

Versión discreta de las leyes de Faraday y Ampére: *el método Finite Difference Time Domain* (FDTD)[1]

Discretizado espacial



Ley de Faraday

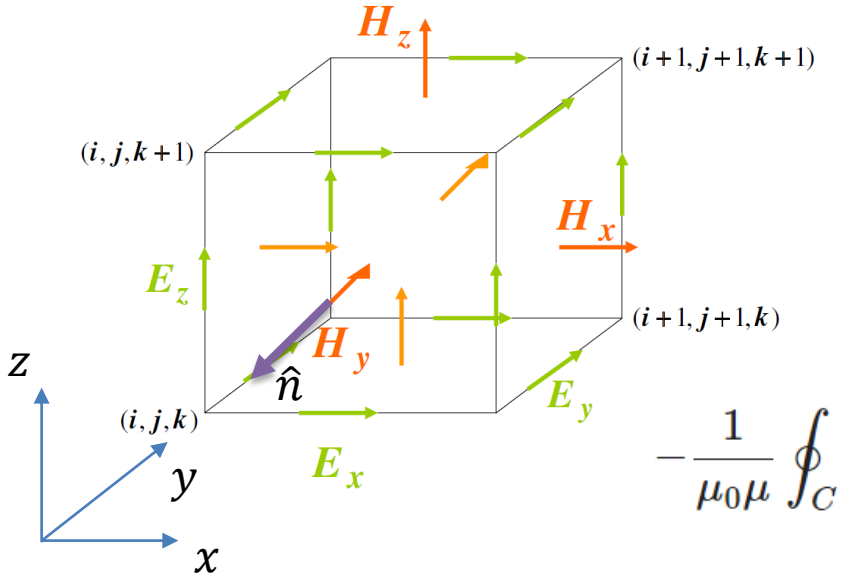
$$\frac{\partial}{\partial t}\int_A \mathbf{H}(\mathbf{r},t)\cdot \mathbf{n}da \approx -\Delta x\,\Delta z\frac{\partial}{\partial t}\left[H_y|_{i+\frac{1}{2},j,k+\frac{1}{2}}\right]$$

$$-\frac{1}{\mu_0\mu}\oint_C \mathbf{E}(\mathbf{r},t)\cdot d\mathbf{s} \approx \frac{1}{\mu_0\mu}\left\{\Delta x\left[E_x|_{i+\frac{1}{2},j,k+1} - E_x|_{i+\frac{1}{2},j,k}\right] + \Delta z\left[E_z|_{i,j,k+\frac{1}{2}} - E_z|_{i+1,j,k+\frac{1}{2}}\right]\right\}$$

[1] *Kane Yee. "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media". IEEE Transactions on Antennas and propagation.* ***14*** *(3): 302–307 (1966)*

Versión discreta de las leyes de Faraday y Ampére:

el método Finite Difference Time Domain (FDTD) (K. Yee (1966))



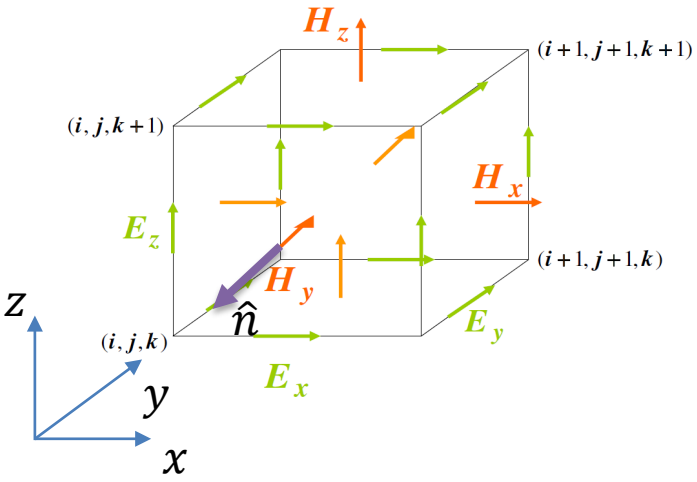
Faraday

$$\frac{\partial}{\partial t}\int_A \mathbf{H}(\mathbf{r},t)\cdot\mathbf{n}da \approx \Delta x\,\Delta z\frac{\partial}{\partial t}\left[H_y|_{i+\frac{1}{2},j,k+\frac{1}{2}}\right]$$

$$-\frac{1}{\mu_0\mu}\oint_C \mathbf{E}(\mathbf{r},t)\cdot d\mathbf{s} \approx \frac{1}{\mu_0\mu}\left\{\Delta x\left[E_x|_{i+\frac{1}{2},j,k+1} - E_x|_{i+\frac{1}{2},j,k}\right] + \Delta z\left[E_z|_{i,j,k+\frac{1}{2}} - E_z|_{i+1,j,k+\frac{1}{2}}\right]\right\}$$

Discretizado espacial

$$\frac{\partial}{\partial t}\left[H_y|_{i+\frac{1}{2},j,k+\frac{1}{2}}\right] = -\frac{1}{\mu_0\mu}\left\{\frac{E_x|_{i+\frac{1}{2},j,k+1} - E_x|_{i+\frac{1}{2},j,k}}{\Delta z} + \frac{E_z|_{i,j,k+\frac{1}{2}} - E_z|_{i+1,j,k+\frac{1}{2}}}{\Delta x}\right\}$$



Discretizado espacial

$$\frac{\partial}{\partial t}\left[H_y|_{i+\frac{1}{2},j,k+\frac{1}{2}}\right] = -\frac{1}{\mu_0\mu}\left\{\frac{E_x|_{i+\frac{1}{2},j,k+1} - E_x|_{i+\frac{1}{2},j,k}}{\Delta z} + \frac{E_z|_{i,j,k+\frac{1}{2}} - E_z|_{i+1,j,k+\frac{1}{2}}}{\Delta x}\right\}$$

Discretizado temporal *leapfrog* (1)

$$\frac{\partial \vec{E}}{\partial t} \approx \frac{\vec{E}^{n+1} - \vec{E}^{n}}{\Delta t}$$

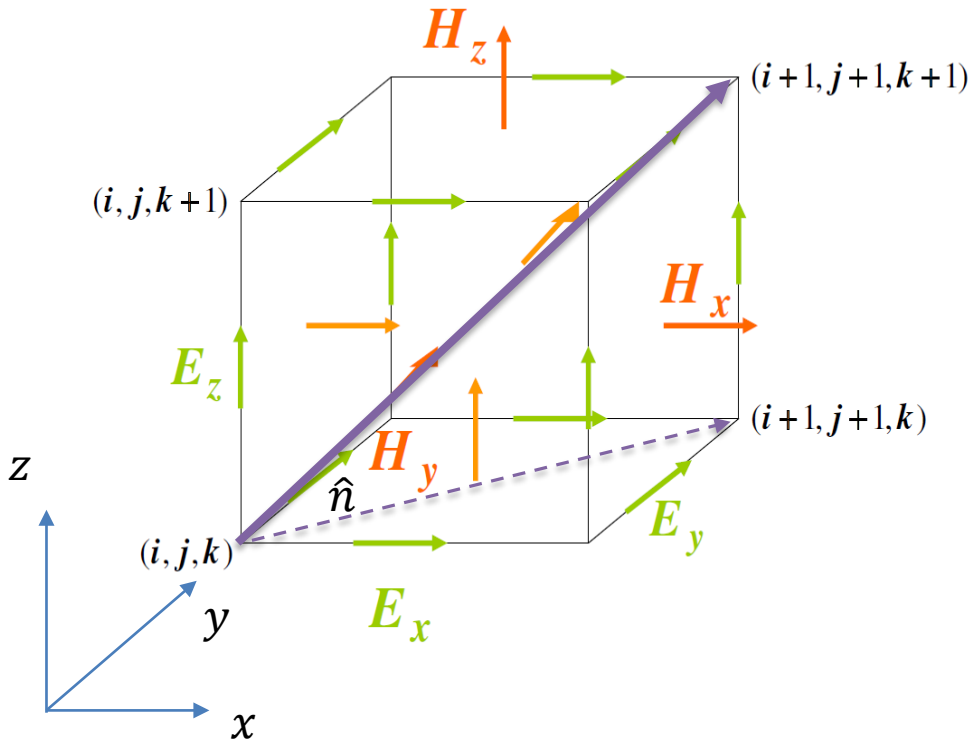
$$\frac{\partial \vec{H}}{\partial t} \approx \frac{\vec{H}^{n+1/2} - \vec{H}^{n-1/2}}{\Delta t}$$

$$H_y|^{n+\frac{1}{2}}_{i+\frac{1}{2},j,k+\frac{1}{2}} = H_y|^{n-\frac{1}{2}}_{i+\frac{1}{2},j,k+\frac{1}{2}} - \frac{\Delta t}{\mu_0\mu}\left\{\frac{E_x|^{n}_{i+\frac{1}{2},j,k+1} - E_x|^{n}_{i+\frac{1}{2},j,k}}{\Delta z} + \frac{E_z|^{n}_{i,j,k+\frac{1}{2}} - E_z|^{n}_{i+1,j,k+\frac{1}{2}}}{\Delta x}\right\}$$

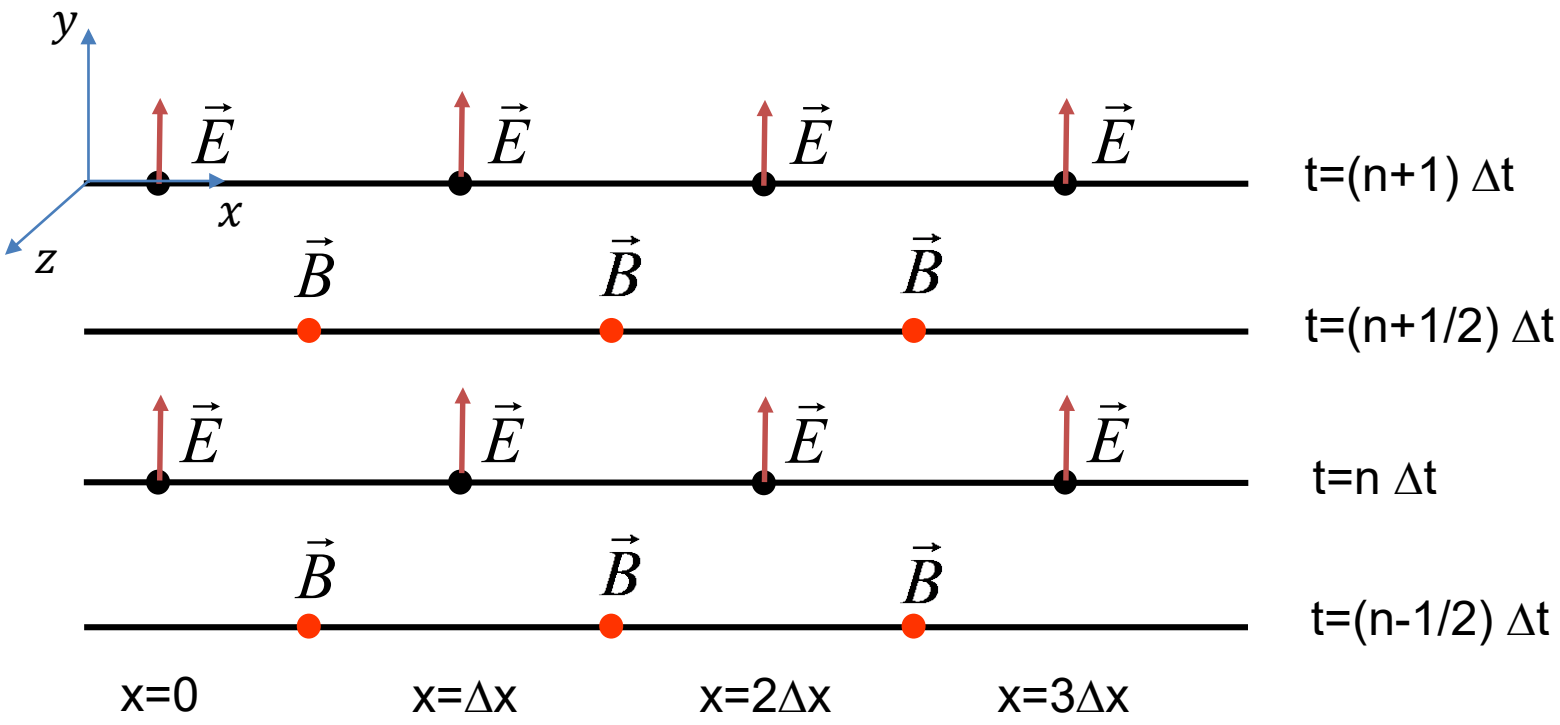
Misma receta con resto de componentes + ley de Ampère

(1) *Kane Yee. IEEE Transactions on Antennas and propagation.* ***14*** *(3): 302–307 (1966)*

Criterio de estabilidad: $(\Delta t)^2 < \left(\frac{c^2}{\Delta x^2} + \frac{c^2}{\Delta y^2} + \frac{c^2}{\Delta z^2} \right)^{-1}$



Onda plana linealmente polarizada propagándose en vacío en 1D (eje x)



$$E_y^{n+1} = E_y^n - \frac{\Delta t}{\mu_0 \varepsilon_0} \frac{\Delta B_z^{n+1/2}}{\Delta x}$$

$$B_z^{n+1/2} = B_z^{n-1/2} - \Delta t \frac{\Delta E_y^n}{\Delta x}$$

Onda plana linealmente polarizada propagándose en vacío en 1D (eje x)

Versión continua

Versión discreta

Ley de Ampère-Maxwell:

$$\frac{\partial E_y}{\partial t} = -\frac{1}{\mu_0 \varepsilon_0}\frac{\partial B_z}{\partial x} \longrightarrow E_y^{n+1} = E_y^n - \frac{\Delta t}{\mu_0 \varepsilon_0}\frac{\Delta B_z^{n+1/2}}{\Delta x}$$

Ley de Faraday :

$$\frac{\partial B_z}{\partial t} = -\frac{\partial E_y}{\partial x} \longrightarrow B_z^{n+1/2} = B_z^{n-1/2} - \Delta t \frac{\Delta E_y^n}{\Delta x}$$

$$E_y^{n+1}(x) = E_y^n(x) - \frac{\Delta t}{\mu_0 \varepsilon_0}\frac{B_z^{n+1/2}\left(x+\frac{\Delta x}{2}\right) - B_z^{n+1/2}\left(x-\frac{\Delta x}{2}\right)}{\Delta x}$$

$$B_z^{n+1/2}\left(x+\frac{\Delta x}{2}\right) = B_z^{n-1/2}\left(x+\frac{\Delta x}{2}\right) - \Delta t \frac{E_y^n(x+\Delta x) - E_y^n(x)}{\Delta x}$$

Onda plana linealmente polarizada propagándose en vacío en 1D (eje x)

Versión continua

$$\frac{\partial E_y}{\partial t} = -\frac{1}{\mu_0 \varepsilon_0}\frac{\Delta B_z}{\Delta x}$$

$$\frac{\partial B_z}{\partial t} = -\frac{\Delta E_y}{\Delta x}$$

Versión discreta

$$E_y^{n+1} = E_y^n - \frac{\Delta t}{\mu_0 \varepsilon_0}\frac{\Delta B_z^{n+1/2}}{\Delta x}$$

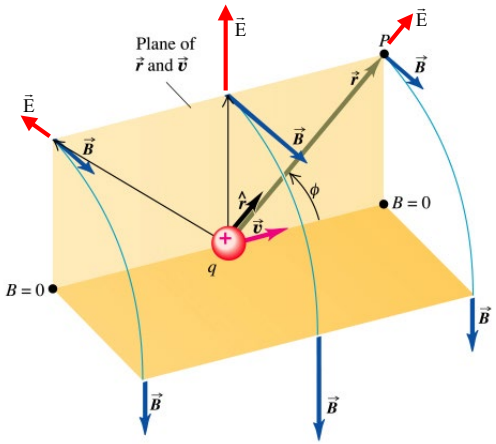
$$B_z^{n+1/2} = B_z^{n-1/2} - \Delta t \frac{\Delta E_y^n}{\Delta x}$$

Versión código Python

```
ey[k] = ey[k] - (dt/(dx*mu0*eps0))*(bz[k] - bz[k-1])
```

```
bz[k] = bz[k] - (dt/dx)*(ey[k+1] - ey[k])
```

¿Fuente de campo EM?



Una partícula cargada crea campo eléctrico, si se está moviendo crea además campo magnético y si se está acelerando **ondas EM**.

¿Qué ocurriría sin estas líneas de código?

```
# Una onda plana se excita en x=x_source
pulse =sin(2 * pi * freq_in * dt * time_step)
ey[x_source]=ey[x_source]-pulse
```

El algoritmo FDTD implementado en Python

Innovación docente: cuadernos interactivos *iPython*


```
%matplotlib inline
import numpy as np
from math import pi, sin,cos,sqrt,exp
from matplotlib import pyplot as plt
from matplotlib import animation

# Constantes físicas. Unidades en el Sistema Internacional (SI)
c       = 299792458.0        # m/s
eps0    = 8.854187817e-12    # F·m-1
mu0     = 1.0/(eps0*(c**2))  # N·A-2


# Parámetros iniciales FDTD
nx = 2000                    # Número de puntos en los que se discretiza la dirección de propagación (dirección x)
dx = 0.001                   # Tamaño de la celda unidad en metros
L= nx*dx                     # Longitud del sistema en metros
x_source = 5                 # Una onda plana se excita en la posición/celda x=x_source
nsteps = 10000               # Pasos temporales
dt = dx/(c*sqrt(1.0))        # Paso temporal en segundos
                             # Hay que tener cuidado y definirlo como está, debido a la restricción del algoritmo de Yee
                             # que impone el criterio de estabilidad en el que dt <= dx/(c*sqrt(3)) para sistemas 3D

# Onda sinusoidad propagándose en la dirección x y polarizada linealmente en la dirección y.
ey = np.zeros(nx)  # Se define el vector que contendrá la componente Ey en todos los puntos definidos en el dominio
bz = np.zeros(nx)  # Se define el vector que contendrá la componente Bz en todos los puntos definidos en el dominio

# Frecuencia incidente
freq_in=2.45e9  # frecuencia en Hz
```

```
print("-----------------------------------------------------")
print("Método de Diferencias Finitas en el Dominio del Tiempo")
print("(FDTD, Finite-Difference Time-Domain)")
print("----------------------------------------------------- \n")
print("Tamaño de la simulación de ",L," m")
print("Pasos necesarios para recorrer todo el sistema=",int(L/(c*dt)))

# Variables para definir las condiciones de contorno
boundary_low = [0, 0]
boundary_high = [0, 0]

# Cada show_step veces muestra el tiempo simulado
show_step=500

# Se guarda ey en cada uno de los instantes fijados por show_step
ey_plt = np.zeros((nsteps,nx))
```

```
'''
El algoritmo FDTD
'''
# Loop FDTD principal
print("..............................................................")
print("\n Empieza a correr el algoritmo FDTD...")
for time_step in range(0, nsteps):
    if(time_step % show_step == 0): # Cada show_step veces muestra el tiempo simulado
        print("Método FDTD calculando, paso temporal ",time_step," de ",nsteps)

    # Se calcula la componente Ey del campo EM para las posiciones dadas
    for k in range(1, nx):
        ey[k] = ey[k] - (dt/(dx*mu0*eps0))*(bz[k] - bz[k-1])
        # Una onda plana se excita en x=x_source
        pulse =sin(2 * pi * freq_in * dt * time_step)
        ey[x_source]=ey[x_source]-pulse # Se introduce la fuente(corriente) en el FDTD (campo eléctrico)

        # Condiciones de absorbente en los extremos
        ey[0] = boundary_low.pop(0)
        boundary_low.append(ey[1])
        ey[nx - 1] = boundary_high.pop(0)
        boundary_high.append(ey[nx - 2])

    # Se calcula la componente Bz del campo EM para las posiciones dadas
    for k in range(0,nx - 1):
        bz[k] = bz[k] - (dt/dx)*(ey[k+1] - ey[k])
        bz[x_source]=bz[x_source]- (dt/dx)*pulse  # Se introduce la fuente en el FDTD (campo magnético)

    # Guarda Ey en todos los instantes de tiempo y todos los puntos nx
    for k in range (1,nx):
        ey_plt[time_step,k]=ey[k]

'''
```

```
A continuación el código que se utiliza para pintar gráficos.
La mayor parte de las líneas pueden tomarse como una receta a seguir.
No es necesario entender exactamente qué hacen
'''
# Definición de la figura y los elementos comunes (ejes...)
fig = plt.figure(dpi=60,   figsize=(10,5))
ax = plt.axes(xlim=(0, dx*nx), ylim=(-np.max(ey_plt), np.max(ey_plt)))
ax.set_xlabel("x (en metros)")
ax.set_ylabel("$E_y$ (V/m)")
ax.axhline(y=0, color='k', linestyle='--')
line, = ax.plot([], [], lw=2)
title = ax.text(.3, 1.05, '', transform = ax.transAxes, va='center')
ax.xaxis.set_animated(True)
# Función que inicializa el primer fotograma
def init():
    title.set_text("")
    line.set_data([], [])
    return line,title
# Función para la animación. Cuando se genera cada uno de los fotogramas es llamada.
show_step=50
no_frames = 500
print("..............................................................")
print("\n La película se crea en el mismo directorio en el que está este archivo \n y se llama fdtd_onda_plana.mp4")
print("\n Empieza la animación...")
def animate(i):
    if(i % show_step == 0): # Cada show_step veces muestra el tiempo simulado
        print("Creando animación, paso ",i," de ", no_frames)
    x = np.linspace(0, nx*dx, nx)
    resolution=int(nsteps/no_frames)
    y = ey_plt[resolution*i]
    line.set_data(x, y)
    title.set_text("Paso temporal="+str(resolution*i))
    return line,title
# LLama al "animador", la función que genera la película.
# Con blit=True solo las partes que cambian de fotograma a fotograma cambian.
anim = animation.FuncAnimation(fig, animate, init_func=init,frames=no_frames, interval=200, blit=True)
# Guarda la animación como mp4. Reguiere tener instalado ffmpeg o mencoder
anim.save('fdtd_onda_plana.mp4', fps=10, extra_args=['-vcodec', 'libx264'])
plt.show()
```

El resultado

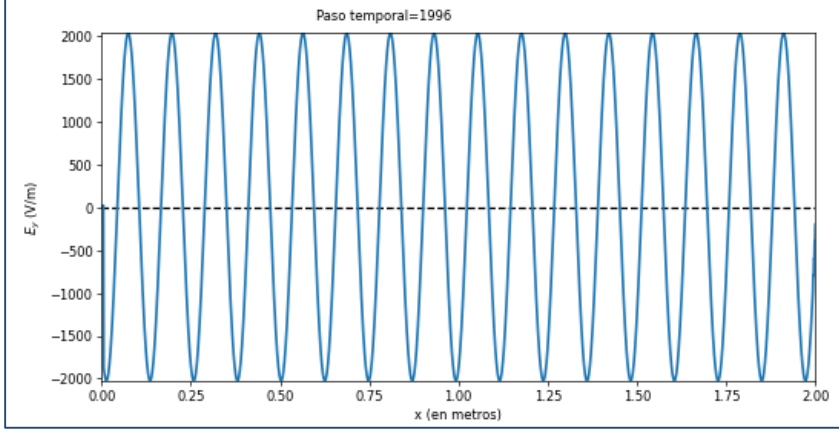
```
-----------------------------------------------------------
Método de Diferencias Finitas en el Dominio del Tiempo
(FDTD, Finite-Difference Time-Domain)
-----------------------------------------------------------

Tamaño de la simulación de  2.0  m
Pasos necesarios para recorrer todo el sistema= 2000
.............................................................

 Empieza a correr el algoritmo FDTD...
Método FDTD calculando, paso temporal  0  de  2000
Método FDTD calculando, paso temporal  500  de  2000
Método FDTD calculando, paso temporal  1000  de  2000
Método FDTD calculando, paso temporal  1500  de  2000
.............................................................

 La película se crea en el mismo directorio en el que está este archivo
 y se llama fdtd_onda_plana.mp4

 Empieza la animación...
Creando animación, paso  0  de  500
Creando animación, paso  50  de  500
Creando animación, paso  100  de  500
Creando animación, paso  150  de  500
Creando animación, paso  200  de  500
Creando animación, paso  250  de  500
Creando animación, paso  300  de  500
Creando animación, paso  350  de  500
Creando animación, paso  400  de  500
Creando animación, paso  450  de  500
```



Actividades

Elegir una de entre las siguientes propuestas:

- Modificar el código proporcionado para que un onda incidente (visible) incida sobre un material i.h.l no dispersivo $\varepsilon = cte \neq 1$. Elegir los parámetros materiales de forma realista.
- Modificar el código proporcionado para que un onda incidente (microondas *f~2.5GHz*) incida sobre un material i.h.l no dispersivo $\varepsilon = cte \neq 1$ y con conductividad $\sigma = cte \neq 0$. Elegir los parámetros materiales de forma realista.
- Las ondas electromagnéticas en vacío no cumplen la misma relación de dispersión en las ecuaciones de Maxwell discretizadas (FDTD) que en su versión continua. Averiguar el porqué y discutir la cuestión.
- En el código proporcionado, calcular la potencia promedio generada por la onda plana y comparar su valor con el teórico.
- A partir del código proporcionado, modificarlo para simular una cavidad 1D con paredes perfectamente reflejantes.
- Hemos utilizado las leyes de Faraday y Ampère. ¿Qué pasa con la ley de Gauss para campo eléctrico? ¿Sigue siendo la divergencia del campo magnético igual a cero? Demostrar ambas situaciones.
- Introducir una fuente con iluminación gaussiana. Calcular numéricamente la transformada de Fourier y comparar el resultado numérico con el analítico.
- En el código buscar *# Condiciones de absorbente en los extremos. ¿Qué hacen las 4 líneas de código de debajo? ¿En que idea "física" se fundamentan?*
- ***...***