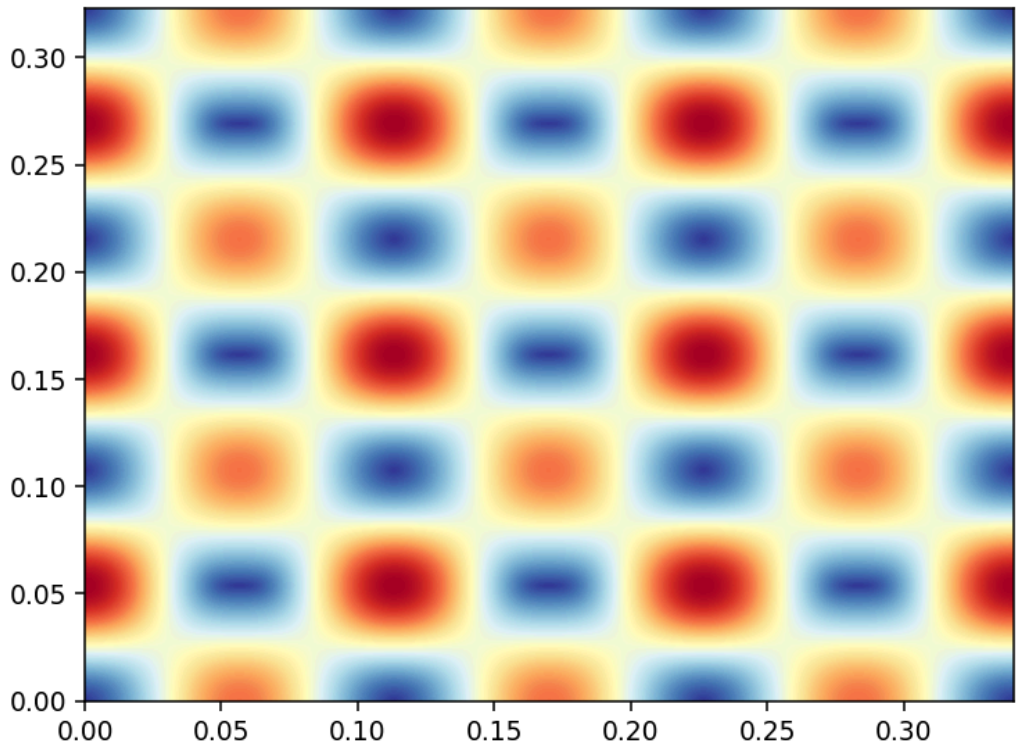
In [40]:
```
# I first wanted to first visualize the poynting vector (hotspots) in the microwave
# at a random slice
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

%matplotlib notebook

# Constants
u_0 = 4 * np.pi * 10**(-7)
L_x = .34
L_y = L_x * .95
L_z = L_x * .8
n_x = 3
n_y = 3
n_z = 3
z_use = .3


k_x = (n_x * np.pi) / L_x
k_y = (n_y * np.pi) / L_y
k_z = (n_z * np.pi) / L_z

# Speed of light and angular frequency
c = 3 * 1e8
w = c * np.sqrt(k_x**2 + k_y**2 + k_z**2)

# Get poynting vector in every direction
def S_x(w, k_x, k_y, k_z, t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_x * x) * np.si
           (((k_x * (np.cos(k_y * y))**2 * (np.sin(k_z * z))**2) -
             (k_y * np.cos(k_y * y)**2 * np.sin(k_z * z)**2)) -
            (k_y * np.sin(k_y * y)**2 * np.cos(k_z * z)**2) +
            (k_x * np.sin(k_y * y)**2 * np.cos(k_z * z)**2))


def S_y(w, k_x, k_y, k_z, t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_y * y) * np.si
           (((k_y * (np.cos(k_z * z))**2 * (np.sin(k_x * x))**2) -
             (k_z * np.cos(k_z * z)**2 * np.sin(k_x * x)**2)) -
            (k_x * np.sin(k_z * z)**2 * np.cos(k_x * x)**2) +
            (k_y * np.sin(k_z * z)**2 * np.cos(k_x * x)**2))

def S_z(w, k_x, k_y, k_z, t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_z * z) * np.si
           (((k_z * (np.cos(k_x * x))**2 * (np.sin(k_y * y))**2) -
             (k_x * np.cos(k_x * x)**2 * np.sin(k_y * y)**2)) -
            (k_y * np.sin(k_x * x)**2 * np.cos(k_y * y)**2) +
            (k_z * np.sin(k_x * x)**2 * np.cos(k_y * y)**2))

# Create a grid for the spatial coordinates (x, y), and fix at a given z
x_vals = np.linspace(0, L_x, 500)
y_vals = np.linspace(0, L_y, 500)
z_vals = np.full_like(x_vals, z_use)  # Create a z array with the same size as x_vals

# Create mesh grid
X, Y = np.meshgrid(x_vals, y_vals)
```

```
# Compute the magnitude of the Poynting vector (S)
def mag(t):
    Z = np.full_like(X, z_use)
    Sx = S_x(w, k_x, k_y, k_z, t, X, Y, Z)
    Sy = S_y(w, k_x, k_y, k_z, t, X, Y, Z)
    Sz = S_z(w, k_x, k_y, k_z, t, X, Y, Z)
    return np.sqrt(Sx**2 + Sy**2 + Sz**2)

# Function to update the plot for each frame
def update(t):
    # Compute the magnitude of the Poynting vector at time t
    data = mag(t)
    # Update the contour plot
    contour_plot = ax.contourf(X, Y, data, cmap='RdYlBu_r', levels = 1000)
    return contour_plot.collections
# Time steps for the animation
t_steps = np.arange(0, 2*np.pi, np.pi/60 )

fig, ax = plt.subplots()

# Set up the animation
ani = FuncAnimation(fig, update, frames=t_steps, interval=1000, blit=False)

# plot
plt.show()
```




```
15932284239.568277
```

In [34]:
```
# Import everything I wanted
import numpy as np
import matplotlib.pyplot as plt
```

```
%matplotlib notebook

# Constants
u_0 = 4 * np.pi * 10**(-7)
L_x = .33 # These lengths were from one of my reference papers
L_y = .32
L_z = .27
n_x = 3 # Nodes were chose to best match roughly the angular frequency of a typical mi
n_y = 3
n_z = 3

# Wave numbers
k_x = (n_x * np.pi)/L_x  # Wave number in x-direction
k_y = (n_y * np.pi)/L_y  # Wave number in y-direction
k_z = (n_z * np.pi)/L_z # Wave number in z-direction

# Speed of light and angular frequency
c = 3 * 1e8
w = c * np.sqrt(k_x**(2) + k_y**(2) + k_z**(2))
print(w)

# Functions used to represent poynting vector in x,y,z direction
def S_x(t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_x * x) * np.si
             (k_y * np.cos(k_y * y) ** 2 * np.sin(k_z * z) ** 2)) -
            (k_y * np.sin(k_y * y) ** 2 * np.cos(k_z * z) ** 2) +
            (k_x * np.sin(k_y * y) ** 2 * np.cos(k_z * z) ** 2))


def S_y(t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_y * y) * np.si
           (((k_y * (np.cos(k_z * z)) ** (2) * (np.sin(k_x * x)) ** (2)) -
             (k_z * np.cos(k_z * z) ** (2) * np.sin(k_x * x) ** (2))) -
            (k_x * np.sin(k_z * z) ** (2) * np.cos(k_x * x) ** 2) +
            (k_y * np.sin(k_z * z) ** (2) * np.cos(k_x * x) ** 2))

def S_z(t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_z * z) * np.si
           (((k_z * (np.cos(k_x * x)) ** 2 * (np.sin(k_y * y)) ** 2) -
             (k_x * np.cos(k_x * x) ** 2 * np.sin(k_y * y) ** 2)) -
            (k_y * np.sin(k_x * x) ** 2 * np.cos(k_y * y) ** 2) +
            (k_z * np.sin(k_x * x) ** 2 * np.cos(k_y * y) ** 2))

# Create a linspace for the coordinates with lengths of 10,000 points
x_vals = np.linspace(0, L_x, 10000)
y_vals = np.linspace(0, L_y, 10000)
z_vals = np.linspace(0, L_z, 10000)

# Create a meshgrid for each plane possible
xy, yx = np.meshgrid(x_vals,y_vals,sparse = True)
yz, zy = np.meshgrid(y_vals,z_vals,sparse = True)
zx, xz = np.meshgrid(x_vals,z_vals,sparse = True)

# Create the average power of the poynting vector in each direction
def surface_X(S, y_vals,z_vals,y_i,y_f,z_i,z_f,x):
    power = S(np.pi/8,x,y_vals,z_vals)[y_i:y_f][z_i:z_f]
    return power
def surface_Y(S, x_vals,z_vals,x_i,x_f,z_i,z_f,y):
    power = S(np.pi/8,x_vals,y,z_vals)[x_i:x_f][z_i:z_f]
    return power
```

```
def surface_Z(S, x_vals,y_vals,x_i,x_f,y_i,y_f,z):
    power = S(np.pi/8,x_vals,y_vals,z)[x_i:x_f][y_i:y_f]
    return power

# Create the actual function for gathering the power over our project
def power(x1,x2,y1,y2,z1,z2):
    power = 0
    top_surface = surface_Z(S_z,xy,yx,x1,x2,y1,y2,z1)
    power += np.sum(abs(top_surface))
    bottom_surface = surface_Z(S_z,xy,yx,x1,x2,y1,y2,z2)
    power += np.sum(abs(bottom_surface))

    front_surface = surface_Y(S_y,xz,zx,x1,x2,z1,z2,y1)
    power += np.sum(abs(front_surface))

    rear_surface = surface_Y(S_y,xz,zx,x1,x2,z1,z2,y2)
    power += np.sum(abs(rear_surface))

    right_surface = surface_X(S_x,yz,zy,y1,y2,z1,z2,x1)
    power += np.sum(abs(right_surface))

    left_surface = surface_X(S_x,yz,zy,y1,y2,z1,z2,x2)
    power += np.sum(abs(left_surface))

    return power
# This shows the power of the entire box on average
x = power(0,10000,0,10000,0,10000)
print(x)
```

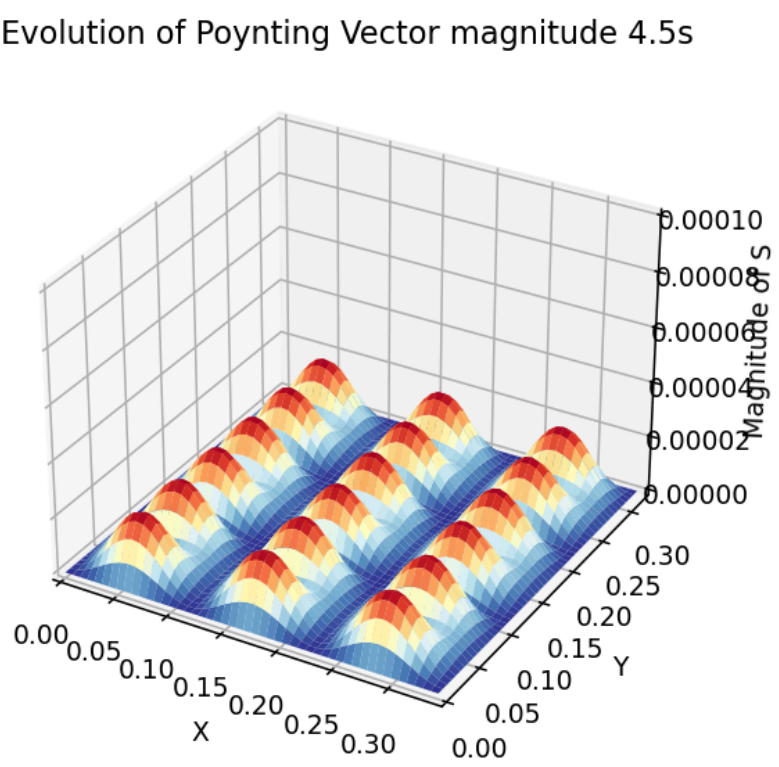
```
13847274417.213652
11851.336122089944
```

In [5]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

%matplotlib notebook

# Constants
u_0 = 4 * np.pi * 10**(-7)
L_x = .34
L_y = L_x * .95
L_z = L_x * .8
n_x = 3
n_y = 3
n_z = 3
z_use = 0

k_x = (n_x * np.pi) / L_x
k_y = (n_y * np.pi) / L_y
k_z = (n_z * np.pi) / L_z

# Speed of light and angular frequency
c = 3 * 1e8
w = c * np.sqrt(k_x**2 + k_y**2 + k_z**2)

# Create poynting vectors for every direction
def S_x(w, k_x, k_y, k_z, t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_x * x) * np.si
           (((k_x * (np.cos(k_y * y))**2 * (np.sin(k_z * z))**2) -
```

```
             (k_y * np.cos(k_y * y)**2 * np.sin(k_z * z)**2)) -
            (k_y * np.sin(k_y * y)**2 * np.cos(k_z * z)**2) +
            (k_x * np.sin(k_y * y)**2 * np.cos(k_z * z)**2))


def S_y(w, k_x, k_y, k_z, t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_y * y) * np.si
           (((k_y * (np.cos(k_z * z))**2 * (np.sin(k_x * x))**2) -
             (k_z * np.cos(k_z * z)**2 * np.sin(k_x * x)**2)) -
            (k_x * np.sin(k_z * z)**2 * np.cos(k_x * x)**2) +
            (k_y * np.sin(k_z * z)**2 * np.cos(k_x * x)**2))


def S_z(w, k_x, k_y, k_z, t_0, x, y, z):
    return ((np.cos(w * t_0) * np.sin(w * t_0) / (u_0 * w)) * (np.cos(k_z * z) * np.si
           (((k_z * (np.cos(k_x * x))**2 * (np.sin(k_y * y))**2) -
             (k_x * np.cos(k_x * x)**2 * np.sin(k_y * y)**2)) -
            (k_y * np.sin(k_x * x)**2 * np.cos(k_y * y)**2) +
            (k_z * np.sin(k_x * x)**2 * np.cos(k_y * y)**2))

# Create a grid for the spatial coordinates
x_vals = np.linspace(0, L_x, 50)
y_vals = np.linspace(0, L_y, 50)
z_vals = np.full_like(x_vals, z_use)
X, Y = np.meshgrid(x_vals, y_vals)

# Compute the magnitude of the Poynting vector (S)
def mag(t):
    Z = np.full_like(X, z_use)  # Create Z array of the same shape as X, Y (set z = L_
    Sx = S_x(w, k_x, k_y, k_z, t, X, Y, Z)
    Sy = S_y(w, k_x, k_y, k_z, t, X, Y, Z)
    Sz = S_z(w, k_x, k_y, k_z, t, X, Y, Z)
    return np.sqrt(Sx**2 + Sy**2 + Sz**2)

# Function to update the plot for each frame
def update(t):
    # Compute the magnitude of the Poynting vector at time t
    data = mag(t)

    # Clear the previous surface
    ax.cla()
    # Plot the new surface
    ax.plot_surface(X, Y, data, cmap='RdYlBu_r', edgecolor='none', alpha=1)

    # Set axis labels
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Magnitude of S')
    ax.set_xlim([0, L_x])
    ax.set_ylim([0, L_y])
    ax.set_zlim([0, .0001])
    ax.set_title(f"Evolution of Poynting Vector magnitude {t:.1f}s" )
    return [ax]

# Time steps
t_steps = np.arange(0, 2*np.pi, np.pi/60)


fig = plt.figure()
# 3D!
```

```
ax = fig.add_subplot(111, projection='3d')

# Set up the animation
ani = FuncAnimation(fig, update, frames=t_steps, interval=1000, blit=False)

# Plot
plt.show()
```



In [ ]: