Функцията **np.linspace** от библиотеката NumPy в Python се използва за генериране на последователност от равномерно разпределени числа в зададен интервал. Тя връща равномерно разпределени числа върху зададен интервал.

**Синтаксис**

**python**

**Copy code**

**numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)**

**Параметри**

* start: Началната стойност на интервала.
* stop: Крайната стойност на интервала.
* num: Броят на числата в генерираната последователност (по подразбиране е 50).
* endpoint: Ако е True, крайната стойност (stop) ще бъде включена в последователността. Ако е False, крайната стойност няма да бъде включена.
* retstep: Ако е True, функцията връща и стъпката между последователните числа.
* dtype: Типът на върнатите числа (например numpy.float64).
* axis: Оста в резултатния масив по която ще се поставят новите стойности.

Кодът ax = **plt.subplot**(111, projection='polar') в Python използва библиотеката Matplotlib, за да създаде полярен subplot (подграфика) в съществуваща фигура. Нека разясним всяка част от този код:

1. **plt.subplot(111)**:
   * Функцията plt.subplot създава нов subplot в съществуваща фигура. Параметърът 111 е специален код, който указва размерността на subplot-а.
   * 111 се интерпретира като 1x1 грид (мрежа) с 1 subplot (подграфика). Трите цифри представляват броя на редовете, броя на колоните и индекса на subplot-а.
     + Първата цифра 1 означава 1 ред.
     + Втората цифра 1 означава 1 колона.
     + Третата цифра 1 означава, че това е първият subplot в тази мрежа.
2. **projection='polar'**:
   * Този параметър указва, че subplot-ът трябва да бъде полярен. Полярните графики са вид диаграми, които се използват за визуализация на данни в полярни координати.
   * Полярната координатна система е двуизмерна координатна система, в която всяка точка в равнината се определя от ъгъл и разстояние от централна точка (полюса).

**Как работи това на практика**

Когато този код се изпълнява, се създава полярен subplot и се връща обект Axes, който може да бъде използван за добавяне на данни към графиката. Този обект Axes предоставя методи за настройка и манипулиране на полярната графика.

Кодът **theta\_full = np.concatenate((theta, np.pi - theta))** използва библиотеката NumPy, за да разшири модела на ъглите, така че да покрие цялата сфера чрез огледално отражение на оригиналния модел. Нека разгледаме по-подробно какво прави този код:

**Обяснение на кода:**

1. **theta**:
   * theta е масив, който съдържа ъглови стойности (обикновено в радиани), покриващи горната полусфера или някакъв сегмент от сферата.
2. **np.pi - theta**:
   * np.pi е константа, която представлява стойността на π (пи).
   * np.pi - theta създава нов масив, като от всяка стойност в theta се изважда π. Това огледално отразява ъгловите стойности спрямо π. Например, ако theta съдържа стойности от 0 до π, то np.pi - theta ще съдържа стойности от π до 0.
3. **np.concatenate((theta, np.pi - theta))**:
   * np.concatenate е функция на NumPy, която комбинира два или повече масива в един.
   * (theta, np.pi - theta) са двата масива, които искаме да комбинираме. Първият масив е оригиналният theta, а вторият масив е огледално отразеният масив np.pi - theta.

**Как работи този код на практика:**

* Първоначалният масив theta съдържа ъглови стойности, които покриват определен сегмент от сферата.
* Масивът np.pi - theta съдържа огледално отразените стойности на theta.
* Комбинирането на двата масива с np.concatenate създава нов масив theta\_full, който включва както оригиналните стойности на ъглите, така и огледалните им стойности. Това разширява покритието на ъгловите стойности, така че да обхваща цялата сфера.

**Примерен код:**

За да стане още по-ясно, нека разгледаме примерен код, който използва np.concatenate:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

# Генериране на ъглови стойности от 0 до π (горна полусфера)

theta = np.linspace(0, np.pi, 100)

# Разширяване на модела за долната полусфера чрез огледално отражение

theta\_full = np.concatenate((theta, np.pi - theta))

# Примерни радиални стойности за визуализация

r = np.abs(np.sin(theta\_full))

# Създаване на полярна графика

ax = plt.subplot(111, projection='polar')

ax.plot(theta\_full, r)

# Настройка на заглавието

ax.set\_title("Полярна графика с разширен модел на ъгли")

plt.show()

В този пример:

* theta съдържа ъглови стойности от 0 до π.
* theta\_full съдържа стойности от 0 до π, последвани от π до 0.
* r е масив от радиални стойности, които се използват за визуализация.
* Полярната графика показва разширения модел на ъглите, като обхваща цялата сфера.

**J1J\_1J1​ е първа функция на Бесел от първи вид**. Функциите на Бесел са решения на диференциалното уравнение на Бесел, което има следния вид:

x2d2ydx2+xdydx+(x2−n2)y=0x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - n^2)y = 0x2dx2d2y​+xdxdy​+(x2−n2)y=0

където nnn е неотрицателно цяло число, което определя реда на функцията на Бесел. За n=1n = 1n=1 получаваме първа функция на Бесел от първи вид J1J\_1J1​.

**Свойства на първа функция на Бесел от първи вид**

Функциите на Бесел от първи вид, Jn(x)J\_n(x)Jn​(x), са дефинирани чрез техните силови редове:

Jn(x)=∑m=0∞(−1)mm!Γ(m+n+1)(x2)2m+nJ\_n(x) = \sum\_{m=0}^{\infty} \frac{(-1)^m}{m! \Gamma(m+n+1)} \left(\frac{x}{2}\right)^{2m+n}Jn​(x)=∑m=0∞​m!Γ(m+n+1)(−1)m​(2x​)2m+n

където Γ\GammaΓ е гамата функция. Специфично за J1J\_1J1​ имаме:

J1(x)=∑m=0∞(−1)mm!Γ(m+2)(x2)2m+1J\_1(x) = \sum\_{m=0}^{\infty} \frac{(-1)^m}{m! \Gamma(m+2)} \left(\frac{x}{2}\right)^{2m+1}J1​(x)=∑m=0∞​m!Γ(m+2)(−1)m​(2x​)2m+1

**Приложения**

Функциите на Бесел намират приложения в много области на физиката и инженерството, особено при анализ на вълнови процеси и дифракция, като например при:

1. Анализ на кръгови вълни и цилиндрични резонатори.
2. Решения на задачата за разпространение на топлина в цилиндрични обекти.
3. Моделиране на вибрациите на кръгли мембрани.

**Примерен код за визуализация на J1(x)J\_1(x)J1​(x) с Python**

Използвайки библиотеката scipy, можем лесно да визуализираме първа функция на Бесел от първи вид:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

from scipy.special import jn

# Генериране на стойности за x

x = np.linspace(0, 20, 1000)

# Изчисляване на J\_1(x)

y = jn(1, x)

# Плотиране на J\_1(x)

plt.figure(figsize=(10, 6))

plt.plot(x, y, label=r'$J\_1(x)$')

plt.title(r'First-order Bessel function of the first kind: $J\_1(x)$')

plt.xlabel(r'$x$')

plt.ylabel(r'$J\_1(x)$')

plt.legend()

plt.grid(True)

plt.show()

Този код ще генерира графика на първа функция на Бесел от първи вид J1(x)J\_1(x)J1​(x), показвайки нейните осцилиращи свойства и затихване с увеличаване на xxx.

**Wavelength (arbitrary units)** се използва в контекста на вълни и описва разстоянието между две съседни точки на една вълна, които имат една и съща фаза на трептене (например, два последователни върха или две последователни долини).

Когато използваме термина "**arbitrary units" (произволни единици), това означава, че не сме конкретизирали или фиксирали физическа единица за измерване на дължината на вълната.** Вместо това, можем да работим с дължината на вълната в относителни или нормализирани стойности, които са подходящи за конкретния контекст или математически модел.

**Примери за използване на "arbitrary units"**

1. **Моделиране и симулации**: В симулации и теоретични изследвания, често се използват произволни единици, за да се опростят уравненията и да се избегне зависимостта от конкретни измервателни системи. Например, ако разглеждаме интерференционен модел, можем да зададем дължината на вълната на една вълна като 1, и всички други дължини ще бъдат сравнени с нея.
2. **Абстрактни теоретични модели**: В някои теоретични изследвания, като квантова механика или оптика, използването на произволни единици може да улесни анализа и разбирането на различни явления без нуждата от конкретни стойности.
3. **Нормализация**: В спектроскопията и анализа на вълни, нормализацията на дължината на вълната може да се използва за сравнение на различни системи или експерименти. Например, можем да зададем най-късата дължина на вълната в спектъра като 1 и да изразим всички други дължини на вълната спрямо нея.

**Примерна графика с произволни единици за дължина на вълната**

Представете си, че имаме различни вълнови форми и искаме да ги изобразим в един и същи график, използвайки произволни единици за дължина на вълната:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

# Генериране на данни

x = np.linspace(0, 4 \* np.pi, 1000)

y1 = np.sin(x) # Вълна с дължина на вълната 2π (единици)

y2 = np.sin(2 \* x) # Вълна с дължина на вълната π (единици)

# Плотиране на вълните

plt.figure(figsize=(10, 6))

plt.plot(x, y1, label='Wavelength = 2π (arbitrary units)')

plt.plot(x, y2, label='Wavelength = π (arbitrary units)')

plt.title('Waves with Different Wavelengths (Arbitrary Units)')

plt.xlabel('Distance (arbitrary units)')

plt.ylabel('Amplitude')

plt.legend()

plt.grid(True)

plt.show()

Тази графика ще покаже две синусоиди с различни дължини на вълната, изразени в произволни единици. Така можем да сравним и анализираме различни вълнови модели без да се нуждаем от конкретни физически измервания.

A Yagi antenna, also known as a Yagi-Uda antenna, is a directional antenna that consists of multiple parallel elements in a line. The direction of the main lobe of a Yagi antenna can be determined using its geometric and electrical properties.

**Formula for the Direction of the Main Lobe**

The direction of the main lobe of a Yagi antenna is generally along the axis of the antenna, typically from the driven element (usually a dipole) towards the directors. The back reflector enhances this directivity.

To find the angle of maximum radiation (θ\thetaθ):

θ=arctan⁡(yx)\theta = \arctan\left(\frac{y}{x}\right)θ=arctan(xy​)

However, for a Yagi antenna, the angle is primarily aligned with the axis, so this is simplified to the direction along the boom of the antenna.

**Python Code for Directional Pattern**

Here is a basic Python script to calculate and plot the radiation pattern of a Yagi antenna using the numpy and matplotlib libraries.

1. **Install necessary packages**:

bash

Copy code

pip install numpy matplotlib

1. **Python script to plot the directional pattern**:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

def radiation\_pattern(num\_elements, spacing, length):

theta = np.linspace(0, 2 \* np.pi, 360)

psi = 2 \* np.pi \* spacing \* np.cos(theta) / length

E = np.abs(np.sin(num\_elements \* psi / 2) / np.sin(psi / 2))

E = E / np.max(E) # Normalize the radiation pattern

return theta, E

# Parameters for the Yagi antenna

num\_elements = 5 # Number of elements

spacing = 0.2 # Spacing between elements in wavelengths

length = 1.0 # Length of the elements in wavelengths

theta, E = radiation\_pattern(num\_elements, spacing, length)

# Plot the radiation pattern

plt.figure(figsize=(8, 6))

ax = plt.subplot(111, projection='polar')

ax.plot(theta, E)

ax.set\_title("Radiation Pattern of a Yagi Antenna")

plt.show()

**Explanation:**

* num\_elements: Number of elements in the Yagi antenna (reflector, driven element, and directors).
* spacing: Distance between the elements in wavelengths.
* length: Effective length of each element in wavelengths.
* theta: Array of angles from 0 to 2π to represent the complete circle.
* psi: Phase difference based on the angle and spacing.
* E: Calculated radiation pattern, normalized for plotting.

**Steps:**

1. Define the number of elements, spacing, and length of the elements.
2. Compute the phase difference and resultant electric field intensity for each angle.
3. Normalize the radiation pattern.
4. Plot the radiation pattern in a polar plot.

This script provides a basic visualization of the directional pattern of a Yagi antenna. Adjust num\_elements, spacing, and length parameters to match the specific Yagi antenna design you are interested in.

need formula for direction wave of Dipole Antenna and python code

ChatGPT

**Formula for the Directional Pattern of a Dipole Antenna**

The radiation pattern of a dipole antenna is typically represented in terms of the elevation angle θ\thetaθ (the angle measured from the axis perpendicular to the dipole). For a half-wave dipole antenna, the radiation pattern in the far field can be approximated by the following formula:

E(θ)=cos⁡(π2cos⁡θ)sin⁡θE(\theta) = \cos \left(\frac{\pi}{2} \cos \theta \right) \sin \thetaE(θ)=cos(2π​cosθ)sinθ

where:

* E(θ)E(\theta)E(θ) is the electric field strength at an angle θ\thetaθ.
* θ\thetaθ is the elevation angle.

This formula describes the normalized radiation pattern of a half-wave dipole antenna. The maximum radiation occurs perpendicular to the axis of the dipole (at θ=90∘\theta = 90^\circθ=90∘ and θ=270∘\theta = 270^\circθ=270∘), and there is no radiation along the axis of the dipole (at θ=0∘\theta = 0^\circθ=0∘ and θ=180∘\theta = 180^\circθ=180∘).

**Python Code to Plot the Radiation Pattern of a Dipole Antenna**

Here is a Python code snippet that uses numpy and matplotlib to plot the radiation pattern of a half-wave dipole antenna:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

# Define the elevation angles (theta) in radians

theta = np.linspace(0, 2 \* np.pi, 360)

# Calculate the radiation pattern (normalized)

E\_theta = np.cos((np.pi / 2) \* np.cos(theta)) \* np.sin(theta)

# Normalize the pattern

E\_theta /= np.max(E\_theta)

# Create a polar plot

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta, E\_theta)

ax.set\_title("Radiation Pattern of a Half-Wave Dipole Antenna", va='bottom')

# Display the plot

plt.show()

**Explanation of the Code**

1. **Import Libraries**:
   * numpy for numerical calculations.
   * matplotlib.pyplot for plotting.
2. **Define Elevation Angles**:

python

Copy code

theta = np.linspace(0, 2 \* np.pi, 360)

This creates an array of angles from 000 to 2π2\pi2π radians (0 to 360 degrees).

1. **Calculate Radiation Pattern**:

python

Copy code

E\_theta = np.cos((np.pi / 2) \* np.cos(theta)) \* np.sin(theta)

This uses the formula for the radiation pattern of a half-wave dipole antenna.

1. **Normalize the Pattern**:

python

Copy code

E\_theta /= np.max(E\_theta)

This ensures the pattern is normalized to a maximum value of 1.

1. **Create a Polar Plot**:

python

Copy code

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta, E\_theta)

ax.set\_title("Radiation Pattern of a Half-Wave Dipole Antenna", va='bottom')

1. **Display the Plot**:

python

Copy code

plt.show()

Running this code will display the normalized radiation pattern of a half-wave dipole antenna in a polar plot, showing how the radiated power varies with the elevation angle. The plot will have maximum radiation perpendicular to the dipole axis and nulls along the dipole axis.

4o

I need formula for direction wave of Monopole Antenna and python code

ChatGPT

**Formula for the Directional Pattern of a Monopole Antenna**

A monopole antenna is essentially half of a dipole antenna, usually placed over a ground plane. The radiation pattern of a quarter-wave monopole antenna is similar to that of a half-wave dipole antenna but only in the upper hemisphere. The formula for the radiation pattern of a monopole antenna above a perfect ground plane can be represented as:

E(θ)=sin⁡(θ)E(\theta) = \sin(\theta)E(θ)=sin(θ)

where:

* E(θ)E(\theta)E(θ) is the electric field strength at an angle θ\thetaθ.
* θ\thetaθ is the elevation angle from the zenith (ranging from 0 to π/2\pi/2π/2).

**Python Code to Plot the Radiation Pattern of a Monopole Antenna**

Here is a Python code snippet that uses numpy and matplotlib to plot the radiation pattern of a quarter-wave monopole antenna:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

# Define the elevation angles (theta) in radians from 0 to π/2 (90 degrees)

theta = np.linspace(0, np.pi / 2, 180)

# Calculate the radiation pattern (normalized)

E\_theta = np.sin(theta)

# Normalize the pattern

E\_theta /= np.max(E\_theta)

# Extend the pattern to the lower hemisphere for visualization (by mirroring the pattern)

theta\_full = np.concatenate((theta, np.pi - theta))

E\_theta\_full = np.concatenate((E\_theta, E\_theta))

# Create a polar plot

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta\_full, E\_theta\_full)

ax.set\_title("Radiation Pattern of a Quarter-Wave Monopole Antenna", va='bottom')

# Display the plot

plt.show()

**Explanation of the Code**

1. **Import Libraries**:
   * numpy for numerical calculations.
   * matplotlib.pyplot for plotting.
2. **Define Elevation Angles**:

python

Copy code

theta = np.linspace(0, np.pi / 2, 180)

This creates an array of angles from 000 to π/2\pi/2π/2 radians (0 to 90 degrees), covering the upper hemisphere.

1. **Calculate Radiation Pattern**:

python

Copy code

E\_theta = np.sin(theta)

This uses the formula for the radiation pattern of a monopole antenna.

1. **Normalize the Pattern**:

python

Copy code

E\_theta /= np.max(E\_theta)

This ensures the pattern is normalized to a maximum value of 1.

1. **Extend the Pattern for Visualization**:

python

Copy code

theta\_full = np.concatenate((theta, np.pi - theta))

E\_theta\_full = np.concatenate((E\_theta, E\_theta))

This mirrors the pattern to visualize it in a full 360-degree polar plot.

1. **Create a Polar Plot**:

python

Copy code

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta\_full, E\_theta\_full)

ax.set\_title("Radiation Pattern of a Quarter-Wave Monopole Antenna", va='bottom')

1. **Display the Plot**:

python

Copy code

plt.show()

Running this code will display the normalized radiation pattern of a quarter-wave monopole antenna in a polar plot. The plot will show how the radiated power varies with the elevation angle, demonstrating the monopole's characteristic radiation pattern with maximum radiation in the plane perpendicular to the antenna and nulls along the axis of the antenna.

4o

I need formula for direction wave of Parabolic Reflectors Antenna and python code

ChatGPT

**Formula for the Directional Pattern of a Parabolic Reflector Antenna**

Parabolic reflector antennas, commonly known as dish antennas, are highly directional antennas used in applications such as satellite communication, radar, and radio telescopes. The main characteristic of a parabolic reflector is its high gain and narrow beamwidth.

The radiation pattern of a parabolic reflector antenna is typically modeled using the aperture distribution and the physical dimensions of the dish. However, a common approximation of the far-field radiation pattern can be described using the Airy pattern (diffraction pattern of a circular aperture):

E(θ)=(2J1(kasin⁡θ)kasin⁡θ)E(\theta) = \left( \frac{2 J\_1(k a \sin \theta)}{k a \sin \theta} \right)E(θ)=(kasinθ2J1​(kasinθ)​)

where:

* E(θ)E(\theta)E(θ) is the electric field strength at an angle θ\thetaθ.
* J1J\_1J1​ is the first-order Bessel function of the first kind.
* kkk is the wave number, given by k=2πλk = \frac{2\pi}{\lambda}k=λ2π​, where λ\lambdaλ is the wavelength.
* aaa is the radius of the parabolic reflector.
* θ\thetaθ is the angle from the axis of the dish (elevation angle).

**Python Code to Plot the Radiation Pattern of a Parabolic Reflector Antenna**

Here is a Python code snippet that uses numpy and matplotlib to plot the radiation pattern of a parabolic reflector antenna:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

from scipy.special import j1 # Import the first-order Bessel function of the first kind

# Define constants

wavelength = 0.03 # Example wavelength in meters (e.g., 3 cm for microwave frequency)

radius = 1.0 # Radius of the parabolic reflector in meters

# Calculate the wave number k

k = 2 \* np.pi / wavelength

# Define the elevation angles (theta) in radians from 0 to 2π

theta = np.linspace(0, 2 \* np.pi, 360)

# Calculate the radiation pattern (normalized)

E\_theta = (2 \* j1(k \* radius \* np.sin(theta)) / (k \* radius \* np.sin(theta)))\*\*2

# Handle the case where theta is zero to avoid division by zero

E\_theta[theta == 0] = 1.0 # Manually set the value at theta = 0 to the maximum

# Normalize the pattern

E\_theta /= np.max(E\_theta)

# Create a polar plot

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta, E\_theta)

ax.set\_title("Radiation Pattern of a Parabolic Reflector Antenna", va='bottom')

# Display the plot

plt.show()

**Explanation of the Code**

1. **Import Libraries**:
   * numpy for numerical calculations.
   * matplotlib.pyplot for plotting.
   * scipy.special.j1 for the first-order Bessel function of the first kind.
2. **Define Constants**:
   * Wavelength and radius of the parabolic reflector.

python

Copy code

wavelength = 0.03 # Example wavelength in meters (e.g., 3 cm for microwave frequency)

radius = 1.0 # Radius of the parabolic reflector in meters

1. **Calculate Wave Number**:

python

Copy code

k = 2 \* np.pi / wavelength

1. **Define Elevation Angles**:

python

Copy code

theta = np.linspace(0, 2 \* np.pi, 360)

1. **Calculate Radiation Pattern**:

python

Copy code

E\_theta = (2 \* j1(k \* radius \* np.sin(theta)) / (k \* radius \* np.sin(theta)))\*\*2

1. **Handle Division by Zero**:

python

Copy code

E\_theta[theta == 0] = 1.0 # Manually set the value at theta = 0 to the maximum

1. **Normalize the Pattern**:

python

Copy code

E\_theta /= np.max(E\_theta)

1. **Create a Polar Plot**:

python

Copy code

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta, E\_theta)

ax.set\_title("Radiation Pattern of a Parabolic Reflector Antenna", va='bottom')

1. **Display the Plot**:

python

Copy code

plt.show()

Running this code will display the normalized radiation pattern of a parabolic reflector antenna in a polar plot. The plot will show how the radiated power varies with the elevation angle, demonstrating the highly directional nature of the parabolic reflector antenna with a narrow beamwidth and high gain.

4o

I need formula for direction wave of Patch Antennas and python code

ChatGPT

**Formula for the Directional Pattern of a Patch Antenna**

Patch antennas, also known as microstrip antennas, are commonly used in various applications such as mobile phones, GPS devices, and satellite communication due to their low profile, low cost, and ease of fabrication. The radiation pattern of a rectangular patch antenna is typically characterized by its broadside radiation pattern (radiation perpendicular to the patch surface).

For a rectangular patch antenna, the far-field radiation pattern can be approximated using the following equations. The E-plane (principal plane) radiation pattern is typically given by:

E(θ)=cos⁡(π2sin⁡θ)E(\theta) = \cos \left( \frac{\pi}{2} \sin \theta \right)E(θ)=cos(2π​sinθ)

where:

* E(θ)E(\theta)E(θ) is the electric field strength at an angle θ\thetaθ.
* θ\thetaθ is the elevation angle from the broadside direction.

**Python Code to Plot the Radiation Pattern of a Patch Antenna**

Below is a Python code snippet using numpy and matplotlib to plot the radiation pattern of a rectangular patch antenna:

python

Copy code

import numpy as np

import matplotlib.pyplot as plt

# Define the elevation angles (theta) in radians from 0 to π

theta = np.linspace(0, np.pi, 360)

# Calculate the E-plane radiation pattern (normalized)

E\_theta = np.cos((np.pi / 2) \* np.sin(theta))

# Normalize the pattern

E\_theta /= np.max(E\_theta)

# Extend the pattern for visualization (mirroring the pattern for a full plot)

theta\_full = np.concatenate((theta, np.pi + theta))

E\_theta\_full = np.concatenate((E\_theta, E\_theta))

# Create a polar plot

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta\_full, E\_theta\_full)

ax.set\_title("Radiation Pattern of a Rectangular Patch Antenna", va='bottom')

# Display the plot

plt.show()

**Explanation of the Code**

1. **Import Libraries**:
   * numpy for numerical calculations.
   * matplotlib.pyplot for plotting.
2. **Define Elevation Angles**:

python

Copy code

theta = np.linspace(0, np.pi, 360)

This creates an array of angles from 000 to π\piπ radians (0 to 180 degrees), covering the upper hemisphere.

1. **Calculate Radiation Pattern**:

python

Copy code

E\_theta = np.cos((np.pi / 2) \* np.sin(theta))

This uses the formula for the E-plane radiation pattern of a rectangular patch antenna.

1. **Normalize the Pattern**:

python

Copy code

E\_theta /= np.max(E\_theta)

This ensures the pattern is normalized to a maximum value of 1.

1. **Extend the Pattern for Visualization**:

python

Copy code

theta\_full = np.concatenate((theta, np.pi + theta))

E\_theta\_full = np.concatenate((E\_theta, E\_theta))

This mirrors the pattern to visualize it in a full 360-degree polar plot.

1. **Create a Polar Plot**:

python

Copy code

plt.figure()

ax = plt.subplot(111, projection='polar')

ax.plot(theta\_full, E\_theta\_full)

ax.set\_title("Radiation Pattern of a Rectangular Patch Antenna", va='bottom')

1. **Display the Plot**:

python

Copy code

plt.show()

Running this code will display the normalized radiation pattern of a rectangular patch antenna in a polar plot. The plot will show how the radiated power varies with the elevation angle, demonstrating the characteristic broadside radiation pattern of the patch antenna with maximum radiation in the direction perpendicular to the patch surface.

# Define the elevation angles (theta) in radians from 0 to 2π