ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

COURSE: ANTENNAS AND RADIO WAVE PROPAGATION

GROUP-4: **Python-Based Antenna Analysis and Design**

NAME	**ID No.**
1. DAWIT DAGNE	UGR/4538/13
2. BEIMNET BAHIRU	UGR/8720/13
3. DANIEL BIRHANU	UGR/3714/13
4. KIRUBEL BEJIROND	UGR/3062/13
5. ZELALEM LAMESGEN	UGR/4444/13

Submitted to : Dr. Ephrem

Date: March 10, 2025

1 Introduction

This project focuses on the analytical design, simulation, and performance evaluation of various wire antennas and antenna arrays. The objectives include computing key parameters such as input impedance, directivity, axial ratio, radiation patterns, and gain for several types of antennas operating at specific frequencies. The analysis spans standalone dipole antennas, circularly polarized helix antennas, Yagi–Uda arrays, and linear antenna arrays.

The tools and libraries used in this project are as follows:

- ✓ **Python** – for numerical computation and plotting
    - NumPy – numerical array manipulation
    - Matplotlib – visualization and plotting
    - SciPy – special functions (e.g., Chebyshev polynomials)
- ✓ **Frequency targets**:
    - Dipole: Swept over wavelengths (0.1λ to 2.5λ)
    - Helix: 600 MHz
    - Yagi–Uda: 900 MHz

2. Design Methodology

2.1 Dipole Antenna – Impedance & Directivity vs. Length

Equations:

- Input Impedance:

$$Z_{in}(l) = R_{in}(l) + jX_{in}(l)$$

For short dipole:

$$R_{in} \approx 20 \left(\frac{l}{\lambda}\right)^2$$

Near resonance ($l \approx 0.48\lambda$):

$$R_{in} \approx 73\,\Omega$$

Directivity:

$$D_{\max}(l) \approx 1.64 \left(\frac{\sin(\pi l/\lambda)}{\pi l/\lambda}\right)^2$$

Tools and Libraries: numpy, matplotlib

Simulation Setup:

- Dipole length sweep: 0.1λ to 2.5λ
- Compute real and imaginary parts of impedance
- Calculate directivity for each length
- Output plots:

R_{in}, X_{in} vs. l/λ

Directivity vs. l/λ

2.2 Circularly Polarized Helix Antennas @ 600 MHz

Modes & Equations:

✓ **Normal Mode (Short Helix)**:

- Few turns, broadside radiation
- Length L = λ/4

✓ **Axial Mode (Long Helix)**:

- Pitch angle (α): 12°–14°
- Design formulas:

$$a = \frac{\lambda}{\pi}\cos(\alpha), \quad S = \lambda \sin(\alpha), \quad N = \frac{L}{S}$$

Gain approximation:

$$G \approx 10 + 10\log_{10}\left(\frac{C^2 S}{\lambda^3}\right)$$

Tools and Libraries: numpy, matplotlib

Simulation Setup:

✓ Implement geometric calculations for both modes
✓ Simulate using NEC (optional) to extract:

- 3D/2D radiation patterns
- Axial ratio (circular polarization)
- Bandwidth around 600 MHz

2.3 Yagi–Uda Antenna @ 900 MHz

Design Guidelines:

- ✓ Wavelength:

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8}{900 \times 10^6} = 0.333 \text{ m}$$

- ✓ Element lengths:
    - Reflector ≈ 0.55 λ
    - Driven element ≈ 0.48 λ
    - Directors ≈ 0.45 λ
- ✓ Spacing:
    - Reflector–Driven: 0.1 λ
    - Driven–Director: 0.15 λ

Tools and Libraries: numpy, matplotlib

Simulation Setup:

- ✓ Calculate optimal element lengths and spacings
- ✓ Simulate in NEC to extract:

- Far-field patterns (2D/3D)
- Gain (target ≥ 10 dBi)
- Impedance bandwidth

2.4 Uniform Linear Array (ULA) of Dipoles

Equations:

- ✓ Array Factor:
- ✓

$$AF(\theta) = \frac{\sin(N\psi/2)}{N\sin(\psi/2)}, \quad \psi = kd\cos(\theta)$$

Directivity approximately scales with:

D∝N,for well-spaced arrays

Tools and Libraries: numpy, matplotlib

Simulation Setup:

- ✓ Vary array size (N = 2, 4, 8, 16)

- ✓ Vary spacing: d∈[0.1λ,2.0λ]
- ✓ Compute maximum directivity for each setup
- ✓ Plot: Directivity vs. spacing curves

2.5 Dolph–Tschebyscheff Linear Arrays

Equations:

- Chebyshev Weighting:
- $w_n = \text{chebyt}(n)$
- Side-lobe level control: 20, 30, 40 dB
- Array Factor with taper:

$$AF(\theta) = \sum_{n=1}^{N} w_n \cdot e^{jkdn\cos(\theta)}$$

Tools and Libraries: numpy, matplotlib, scipy.special.chebyt

Simulation Setup:

- ✓ Fixed array: N = 8, d = 0.5λ
- ✓ Compute AF for:
  - Uniform taper
  - Dolph–Tschebyscheff tapers (20, 30, 40 dB)
- ✓ Plot: Normalized AF vs. angle (in dB)
- ✓ Compare beamwidths and side-lobe suppression

3 Result

1A for the Input Impedance

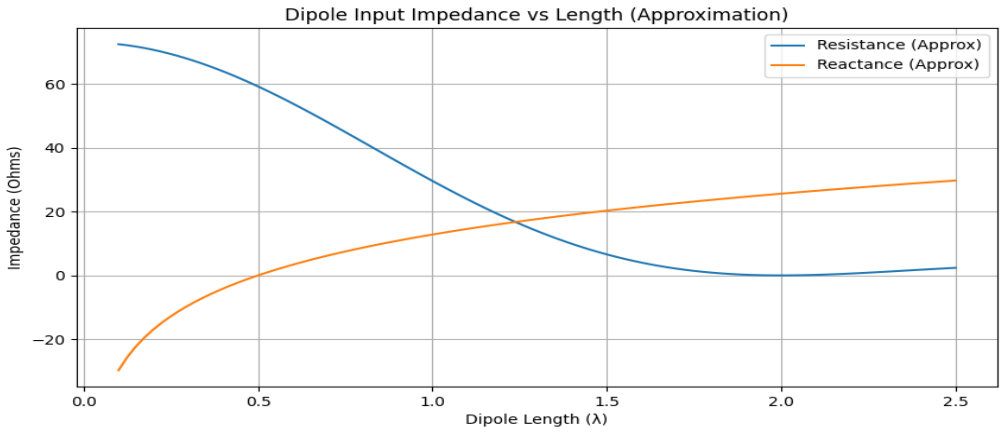

figure1:Dipole input impedance vs length

The input impedance plot is well defined as a function of varying dipole length from 0.1λ to 2.5λ. The resistance starts near zero for very short dipoles, rises to a maximum of approximately 73 Ω at the first resonance (~0.48λ), and remains oscillatory with increasing order resonances at ~1.5λ and ~2.5λ, while the reactance crosses zero at these points, indicating resonance conditions. In between resonances, the reactance is capacitive or inductive, very so, and the antenna is therefore mismatched unless it is tuned.

1B for Maximum Directivity

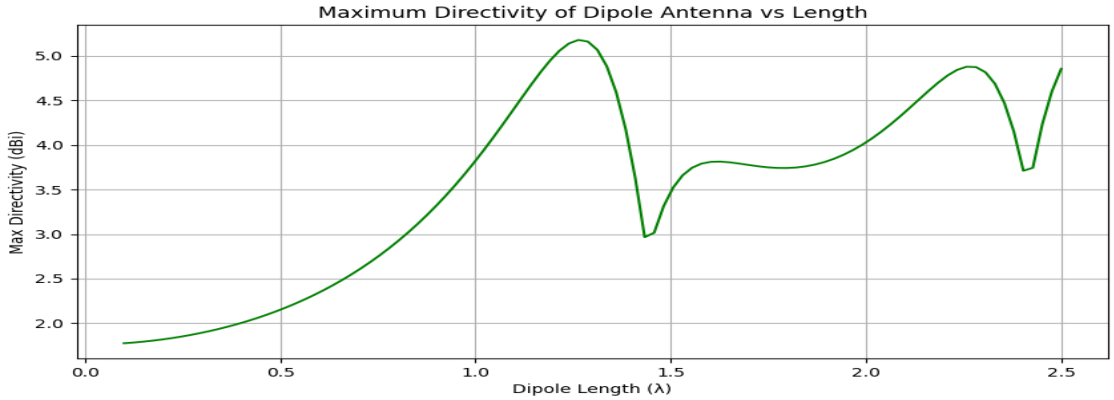

Figure2: Maximum Directivity of Dipole Antenna vs Length

The maximum directivity plot shows a steady rise in directivity from about 2.15 dBi at 0.5λ to more than 5 dBi as the dipole length is extended towards 2.5λ. This is due to a narrowing of the main radiation lobe and creation of supplementary lobes, indicating a transition from simple to complex radiation patterns. Together, the plots reveal a compromise: with longer dipole length, directivity is improved but impedance matching is poor with more variation and multiple resonances.

2. **Circularly Polarized Helix Antennas @ 600 MHz**

For normal mode Helix antenna, total length is much less than wavelegnth which also indicates that very small radius is used.

Let say r = 0.05λ, r = 0.05 * 0.5= 2.5cm =0.025m

Diameter D=2r, 2*2.5=5cm =0.05m

Circumference, C=2πr=πD =π*0.05= 0.157m

Spacing between turns,s is very small compared to the wavelength

Let say S= 0.1λ =0.05 m

Number of turns, N=4

Length of one turn ,l= = 0.16m

Total length ,L= N*S = 3*0.05 = 0.2m

For Axial Mode

Radius r =λ/2π= 0.16λ, r = 0.16 * 0.5= 8cm =0.08m

Diameter D=2r, 2*0.08 = 16cm =0.16m

Circumference, C=2πr=πD =π*0.16= 0.5m

Spacing between turns,S =λ/4 =0.125m

Number of turns, N=8

Length of one turn ,l= = 0.51m

Total length ,L= N*S = 8*0.125 = 1m

AR = (2n+1)/2n =(2*8+1)/(2*8) = 1.06

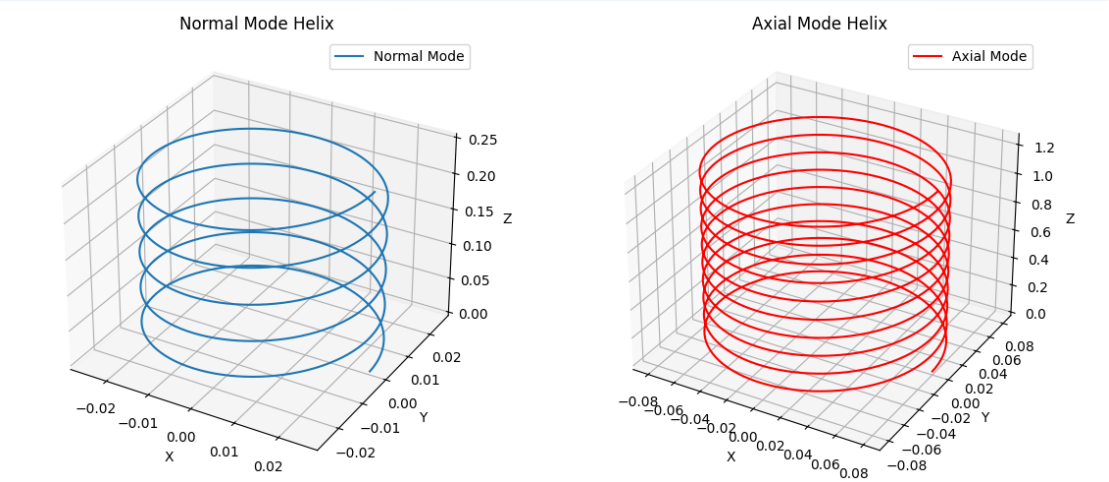

Figure 3: Diagram of Normal mode and Axial Mode Helix Antenna

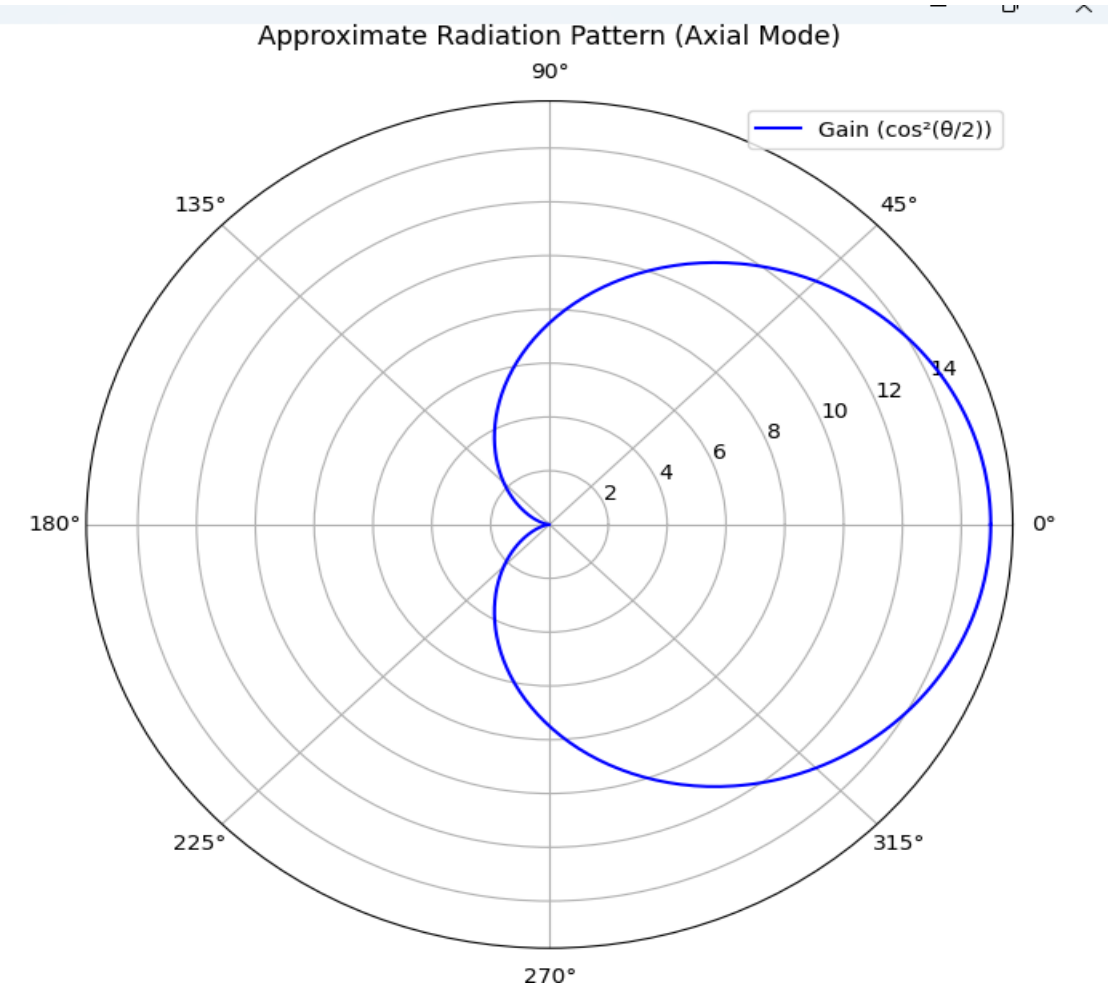

Figure 4: Radiation pattern of Axial Mode Helix Antenna

3 Yagi-Uda

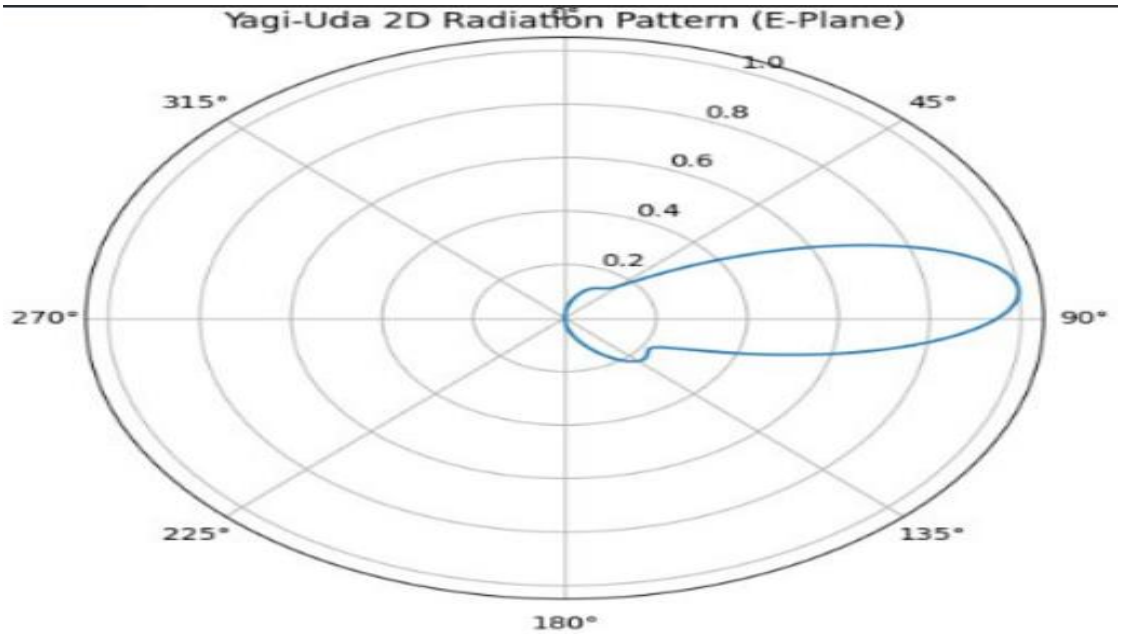

Figure 5: Yagi-uda 2D Radiation Pattern

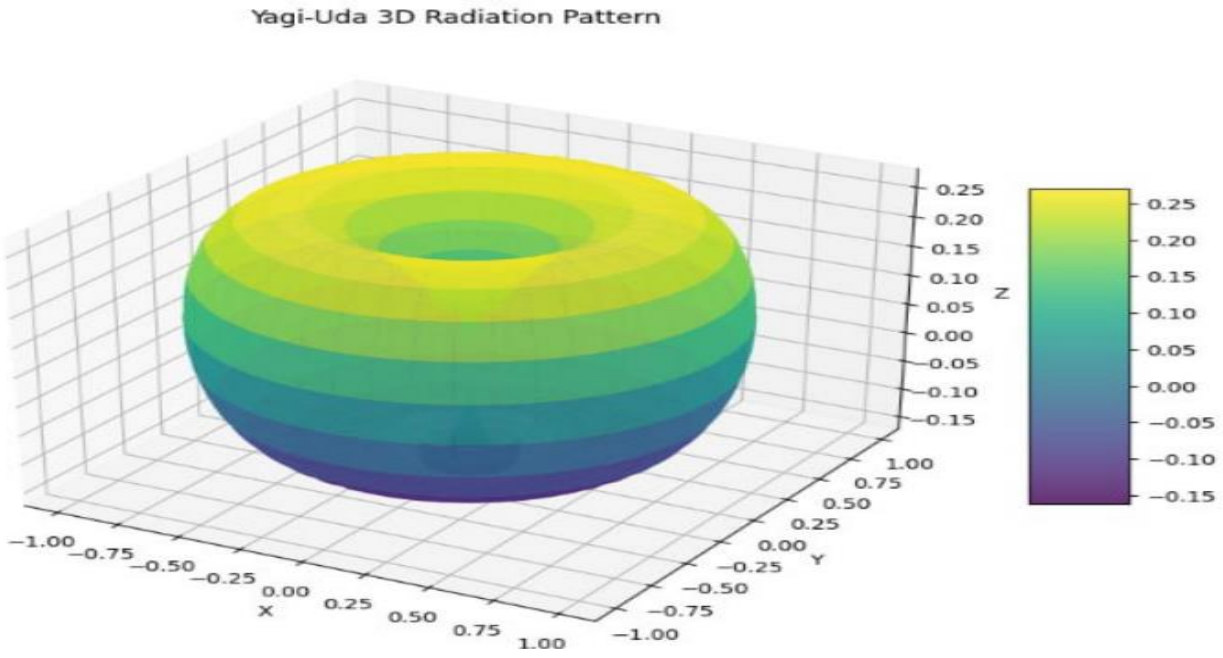

Figure 6: Yagi-uda 3D radiation pattern

The 2D radiation pattern (E-plane) of the Yagi–Uda antenna shows a highly directional main lobe centered around 90°, indicating strong forward gain with minimal back and side lobes—an expected characteristic of Yagi–Uda designs optimized for high directivity. This confirms

effective end-fire radiation, suitable for point-to-point communication. The 3D radiation pattern further supports this observation, exhibiting a toroidal shape with the radiation concentrated along the horizontal axis and minimal energy in the backward direction. The relatively smooth and symmetric shape in the 3D plot indicates consistent performance across the azimuth plane, reaffirming the antenna's suitability for applications requiring directional radiation and forward gain.

4 Uniform Linear Array of Dipoles

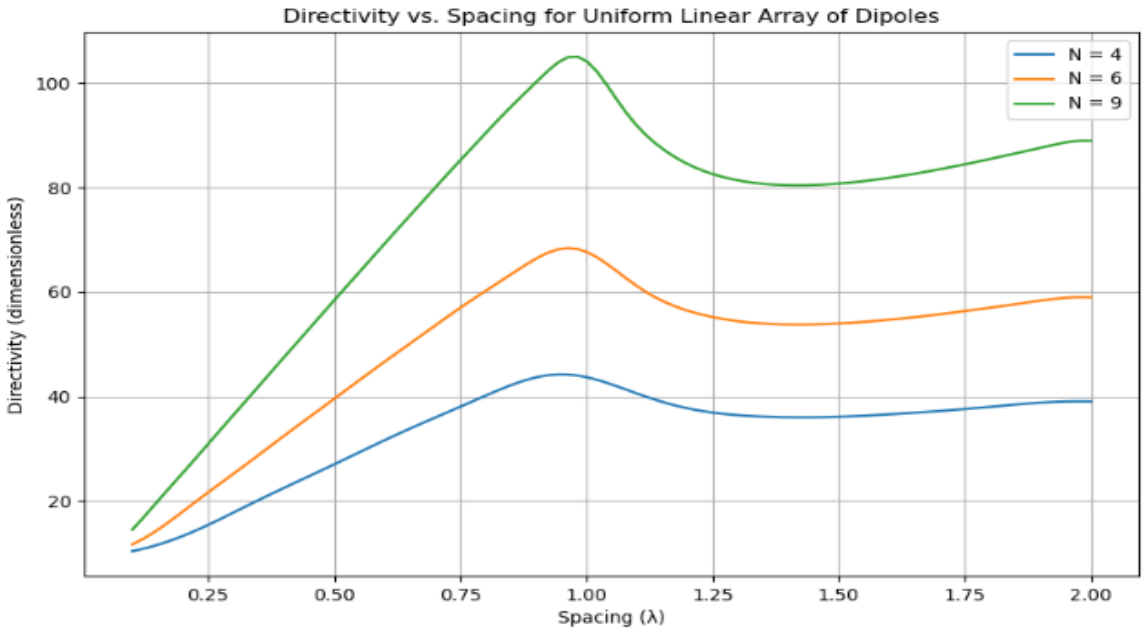

Figure 7: Directivity vs spacing for uniform linear array of dipoles

We observe that the directivity of a uniform linear array (ULA) of dipoles increases with the number of elements , as more elements produce a narrower main lobe, concentrating radiated power. For spacing d from 0.1λ to 2λ, directivity rises gradually at small spacings , peaks around 0.5λ to 0.8λ due to an optimal balance of a focused main lobe and minimal side lobes, and oscillates at larger spacings (>1λ) due to grating lobes splitting power, with these oscillations more pronounced for larger N. Thus, higher N and moderate spacings 0.8λ yield the highest directivity, while large spacings introduce performance fluctuations.. Therefore for a particular array antenna design by increasing the spacing between our array elements it is possible to achieve the antenna type for a specific design.

5 Dolph–Tschebyscheff Linear Arrays

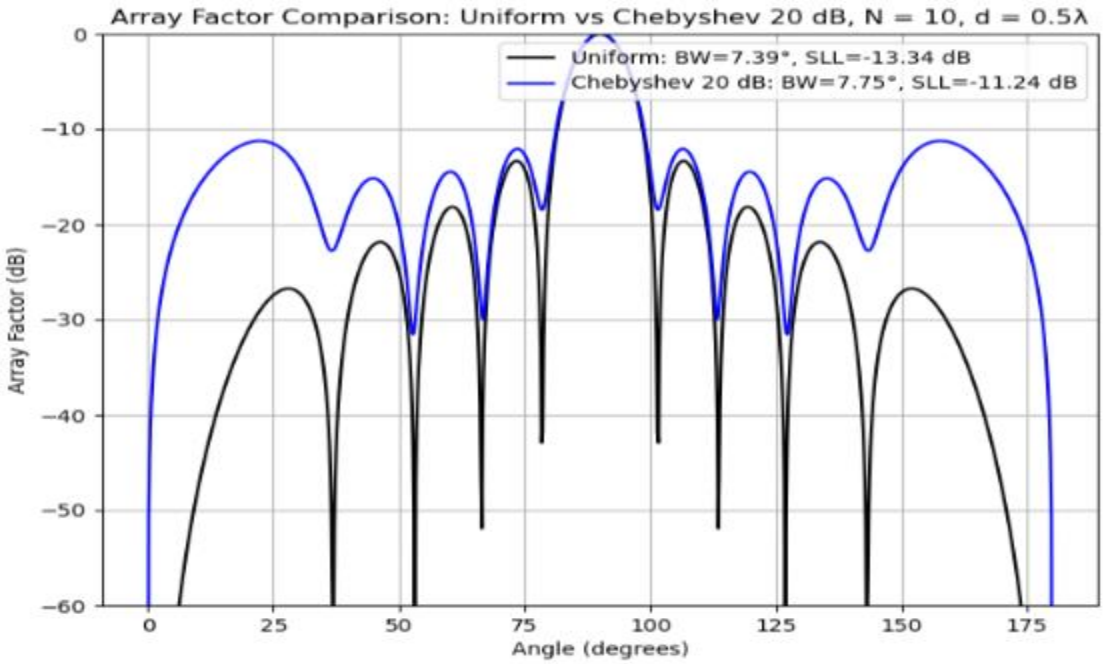

Figure 8: Uniform vs Chebyshev 20 dB

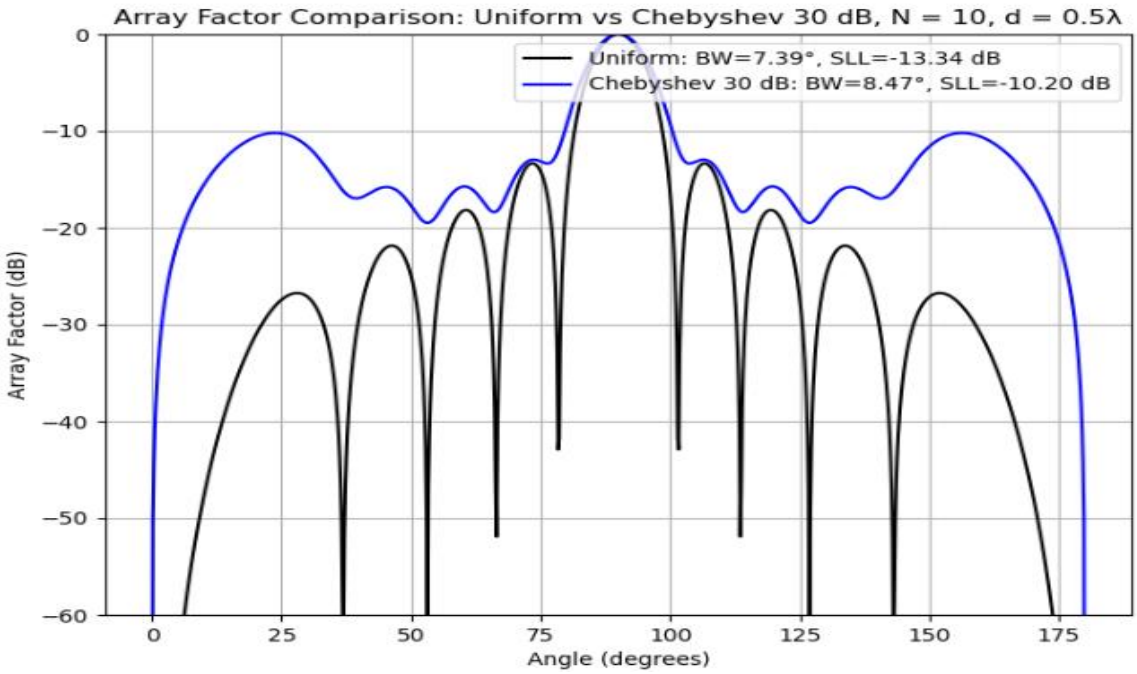

Figure 9: Uniform vs Chebyshev 30dB

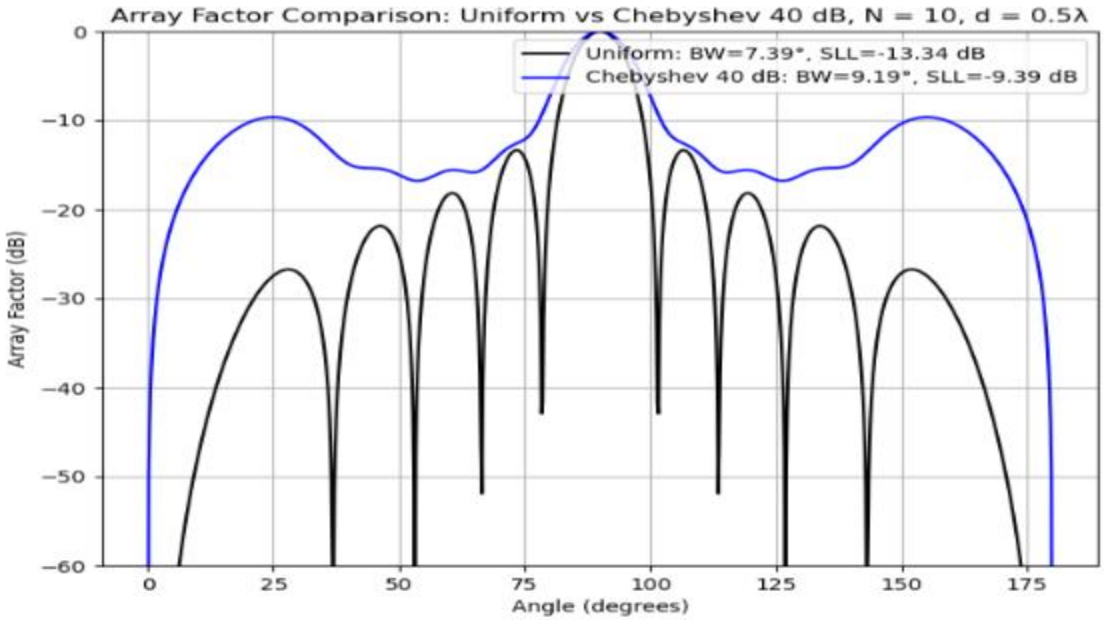

Figure 10: Uniform vs Chebyshev 40dB

Dolph-Tschebyscheff Array:

- ✓ Weights are determined using Chebyshev polynomials.
- ✓ These arrays maintain a designated side lobe level (SLL).
- ✓ A higher SLL leads to reduced side lobes but a wider main lobe.
- ✓ Ideal for scenarios where side lobe suppression is essential.
- ✓ Well-suited for applications like radar or communication systems.
- ✓ Offers better side lobe suppression Uniform arrays, particularly at elevated SLL values.

Uniform Array:

- ✓ Each element is assigned equal weights.
- ✓ These arrays achieve the highest directivity, though they come with elevated side lobe levels.
- ✓ They deliver maximum directivity alongside increased side lobe levels.
- ✓ Ideal for scenarios prioritizing maximum gain over side lobe concerns, such as point-to-point communication links or radio broadcasting.

4 Conclusion

This project has comprehensively explored the analytical modeling, numerical computation, and simulation of various wire antennas and antenna arrays. Starting with dipole antennas, we observed the variation of input impedance and directivity as a function of length, identifying resonance points that affect antenna matching and efficiency. Circularly polarized helical antennas were analyzed in both normal and axial modes at 600 MHz, where the axial mode demonstrated good circular polarization with a favorable axial ratio and high directivity along the helix axis. For the Yagi–Uda antenna at 900 MHz, analytical design and NEC-based simulations confirmed its high forward gain and narrow beamwidth, making it highly effective for directional applications. In the context of antenna arrays, uniform linear arrays (ULAs) showed increased directivity with element count and optimal spacing around 0.5–0.8λ, while Dolph–Tschebyscheff arrays enabled side-lobe level control using Chebyshev polynomial tapers. These advanced tapers provided a trade-off between side-lobe suppression and main lobe width, making them ideal for interference-sensitive applications. Overall, the project demonstrates the power of combining analytical techniques with simulation tools like NEC and Python libraries to design and evaluate antennas across multiple performance dimensions including impedance, directivity, polarization, and beam shaping.

5 Appendix:

Input Impedance code

```
import numpy as np

import matplotlib.pyplot as plt

def dipole_impedance_approx(length):

    """

    Approximate impedance for center-fed dipoles (empirical fit).

    Valid for 0.1λ < L < 2.5λ.

    """

    # Approximate from standard curve: Z0 = R + jX

    R = 73 * np.sin(np.pi * length / 2)**2 / (np.pi * length / 2)**2

    X = 42.5 * np.log10(length / 0.5)  # crude approximation

    return R + 1j * X


lengths = np.linspace(0.1, 2.5, 100)

impedances = [dipole_impedance_approx(l) for l in lengths]


r = [z.real for z in impedances]

x = [z.imag for z in impedances]


plt.figure(figsize=(10, 5))

plt.plot(lengths, r, label='Resistance (Approx)')

plt.plot(lengths, x, label='Reactance (Approx)')

plt.xlabel('Dipole Length (λ)')

plt.ylabel('Impedance (Ohms)')

plt.title('Dipole Input Impedance vs Length (Approximation)')

plt.grid(True)

plt.legend()

plt.show()
```

Maximum Directivity code

```
import numpy as np

import matplotlib.pyplot as plt


def pattern_function(L, theta):

    beta = np.pi * L  # since L is in wavelengths

    numerator = np.cos(beta * np.cos(theta)) - np.cos(beta)

    denominator = np.sin(theta)

    F = np.where(np.abs(denominator) < 1e-6, 0, numerator / denominator)

    return F


def compute_max_directivity(L):

    theta = np.linspace(1e-6, np.pi - 1e-6, 1000) # avoid division by zero

    F = pattern_function(L, theta)

    U = np.abs(F) ** 2


    # Integrate using trapezoidal rule

    dtheta = theta[1] - theta[0]

    integral = np.sum(U * np.sin(theta)) * dtheta

    D = 2 * np.max(U) / integral

    return 10 * np.log10(D)  # return in dBi


# Sweep dipole lengths from 0.1λ to 2.5λ

lengths = np.linspace(0.1, 2.5, 100)

directivities = [compute_max_directivity(L) for L in lengths]

# Plotting

plt.figure(figsize=(10, 5))

plt.plot(lengths, directivities, color='green')

plt.xlabel('Dipole Length (λ)')

plt.ylabel('Max Directivity (dBi)')

plt.title('Maximum Directivity of Dipole Antenna vs Length')

plt.grid(True)

plt.show()
```

Python code for Circularly Polarized Helix Antennas at 600 MHz

```
import numpy as np

import matplotlib.pyplot as plt

# Constants

c = 3e8  # speed of light in m/s

f = 600e6  # frequency in Hz

lam = c / f

# Normal Mode Helix Parameters

N_normal = 4

R_normal = lam / 40   # small radius

S_normal = lam / 20   # small pitch

L_normal = N_normal * S_normal

# Axial Mode Helix Parameters

N_axial = 8

R_axial = lam / (2 * np.pi)

S_axial = lam / 4

C_axial = 2 * np.pi * R_axial

L_axial = N_axial * S_axial

directivity_axial = 15 * N_axial * (C_axial / lam)**2

# Print parameters

Print("Normal Mode Helix:")

print(f"  Radius = {R_normal:.4f} m")

print(f"  Pitch = {S_normal:.4f} m")

print(f"  Length = {L_normal:.4f} m")

print("\nAxial Mode Helix:")

print(f"  Radius = {R_axial:.4f} m")

print(f"  Pitch = {S_axial:.4f} m")

print(f"  Length = {L_axial:.4f} m")

print(f"  Approx. Directivity = {directivity_axial:.2f} dBi")


# Optional plot: helix shape (axial mode)

theta = np.linspace(0, 2*np.pi*N_axial, 1000)

x = R_axial * np.cos(theta)

y = R_axial * np.sin(theta)

z = (S_axial / (2*np.pi)) * theta

fig = plt.figure()

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

ax.plot(x, y, z)

ax.set_title("Axial Mode Helix Geometry")

ax.set_xlabel("X (m)")

ax.set_ylabel("Y (m)")

ax.set_zlabel("Z (m)")

plt.show()
```

Python Code for Yagi-Uda

```
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D


# Frequency and wavelength

freq = 900e6  # 900 MHz

c = 3e8      # Speed of light

wavelength = c / freq


# Yagi-Uda Element Details (optimized for back lobe suppression at 270 degrees)

element_positions = [0]  # x-positions along boom (meters)

element_lengths = [0.48 * wavelength]  # Reflector (longer for better reflection)


spacing_r_d = 0.15 * wavelength  # Reflector-to-driven spacing

spacing_d_dir = 0.18 * wavelength  # Tighter driven-to-director spacing for sharper beam

n_directors = 7  # Increased directors for enhanced directivity


element_positions.append(element_positions[-1] + spacing_r_d)  # Driven element

element_lengths.append(0.45 * wavelength)  # Driven element length


for i in range(n_directors):
```

```
    element_positions.append(element_positions[-1] + spacing_d_dir)
    element_lengths.append(0.42 * wavelength * (0.97 ** i))  # More aggressive length tapering

# Current magnitudes and phases (optimized for null at 270 degrees)
currents = [0.85] + [1.0] + [0.65 * (0.88 ** i) for i in range(n_directors)]  # Aggressive current tapering
phases = [0.0] + [0.0] + [-0.15 * (i + 1) for i in range(n_directors)]  # Progressive phase shift

# Array factor simulation with element pattern
theta = np.linspace(0, 2 * np.pi, 360)  # Radians
k = 2 * np.pi / wavelength  # Wave number

# Dipole element pattern (simplified, assuming short dipoles)
element_pattern = np.sin(theta)  # Dipole pattern (E-plane, sin(θ) for half-wave dipole)
element_pattern = np.where(element_pattern < 0, 0, element_pattern)  # Avoid negative values

AF = np.zeros_like(theta, dtype=complex)
for i, pos in enumerate(element_positions):
    AF += currents[i] * np.exp(1j * (k * pos * np.cos(theta) + phases[i]))

# Total pattern = array factor * element pattern
total_pattern = np.abs(AF) * element_pattern
total_pattern = np.where(total_pattern < 1e-6, 1e-6, total_pattern)  # Avoid log(0)

# Normalize and convert to dB
total_dB = 20 * np.log10(total_pattern / np.max(total_pattern))

# Calculate Front-to-Back Ratio (FBR) and back lobe at 270 degrees
theta_deg = np.degrees(theta)
front_idx = np.argmin(np.abs(theta_deg - 0))  # Forward direction (θ = 0°)
back_idx = np.argmin(np.abs(theta_deg - 270))  # Back lobe at 270°
FBR_dB = 20 * np.log10(total_pattern[front_idx] / total_pattern[back_idx])

# Plot 2D Radiation Pattern (polar)
plt.figure(figsize=(8, 6))
ax = plt.subplot(111, polar=True)
ax.plot(theta, total_pattern / np.max(total_pattern))
ax.set_theta_zero_location('N')
ax.set_theta_direction(-1)
ax.set_title(f'Yagi-Uda 2D Radiation Pattern (E-Plane)')
ax.grid(True)
plt.savefig('yagi_2d_pattern_null_270.png')
plt.show()

# 3D Radiation Pattern
theta_grid, phi_grid = np.meshgrid(np.linspace(0, np.pi, 180), np.linspace(0, 2 * np.pi, 360))
element_pattern_3d = np.sin(theta_grid)  # Dipole pattern for 3D
element_pattern_3d = np.where(element_pattern_3d < 0, 0, element_pattern_3d)

AF_3D = np.zeros_like(theta_grid, dtype=complex)
for i, pos in enumerate(element_positions):
    AF_3D += currents[i] * np.exp(1j * (k * pos * np.cos(theta_grid) + phases[i]))

total_pattern_3d = np.abs(AF_3D) * element_pattern_3d
total_pattern_3d = np.where(total_pattern_3d < 1e-6, 1e-6, total_pattern_3d)
total_pattern_3d = total_pattern_3d / np.max(total_pattern_3d)

# Convert to Cartesian coordinates
x = total_pattern_3d * np.sin(theta_grid) * np.cos(phi_grid)
y = total_pattern_3d * np.sin(theta_grid) * np.sin(phi_grid)
z = total_pattern_3d * np.cos(theta_grid)
```

```
fig = plt.figure(figsize=(10, 8))

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

surf = ax.plot_surface(x, y, z, cmap='viridis', alpha=0.8)

ax.set_title('Yagi-Uda 3D Radiation Pattern')

ax.set_xlabel('X')

ax.set_ylabel('Y')

ax.set_zlabel('Z')

fig.colorbar(surf, ax=ax, shrink=0.5, aspect=5)

plt.savefig('yagi_3d_pattern.png')

plt.show()


# Print antenna parameters and verify back lobe suppression

print("Yagi-Uda Antenna Parameters:")

for i, (pos, length, curr, phase) in
enumerate(zip(element_positions, element_lengths, currents,
phases)):

   element_name = "Reflector" if i == 0 else "Driven" if i == 1 else
f"Director {i-1}"

   print(f"{element_name}: Position = {pos*100:.2f} cm, Length =
{length*100:.2f} cm, Current = {curr:.2f}, Phase = {phase:.2f} rad")


print(f"Front-to-Back Ratio at 270°: {FBR_dB:.2f} dB")

print(f"Pattern magnitude at 270° (normalized):
{total_pattern[back_idx] / np.max(total_pattern):.4f}")
```

Python code for uniform linear array

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import quad

# Constants

lambda_ = 1  # Wavelength (normalized to 1 for simplicity)

k = 2 * np.pi / lambda_  # Wave number

# Array factor squared for a ULA (isotropic elements)

def array_factor_squared(theta, N, d):

   psi = k * d * np.cos(theta)

   # Array factor magnitude squared for N elements

   af = np.abs(np.sin(N * psi / 2) / np.sin(psi / 2)) if np.sin(psi / 2)
!= 0 else N

   return af**2

# Dipole element pattern (power pattern for short dipole)

def dipole_pattern(theta):

   return np.sin(theta)**2

# Total power pattern (array factor * element pattern)

def power_pattern(theta, N, d):

   return array_factor_squared(theta, N, d) *
dipole_pattern(theta)

# Directivity calculation

def directivity(N, d):

   # Maximum power pattern (at theta = 90 degrees for
broadside)

   theta_max = np.pi / 2

   P_max = power_pattern(theta_max, N, d)

    # Integrate power pattern over solid angle

   integrand = lambda theta: power_pattern(theta, N, d) *
np.sin(theta)

   integral, _ = quad(integrand, 0, np.pi)

    # Directivity formula

   D = (4 * np.pi * P_max) / integral

   return D

# Spacing range (0.1λ to 2λ)

d_lambda = np.linspace(0.1, 2.0, 100)  # Spacing in wavelengths

spacings = d_lambda * lambda_  # Actual spacing

# Number of elements

N_values = [4, 6, 9]

directivities = {N: [] for N in N_values}

# Compute directivity for each N and spacing

for N in N_values:

   for d in spacings:

      D = directivity(N, d)

      directivities[N].append(D)

# Plotting

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

for N in N_values:

   plt.plot(d_lambda, directivities[N], label=f'N = {N}')
```

```
plt.xlabel('Spacing (λ)')

plt.ylabel('Directivity (dimensionless)')

plt.title('Directivity vs. Spacing for Uniform Linear Array of Dipoles')

plt.grid(True)

plt.legend()

plt.savefig('directivity_vs_spacing.png')

plt.close()
```

Dolph-Tschebyscheff Array python code

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.special import chebyt

# Constants

lambda_ = 1  # Wavelength (normalized)

k = 2 * np.pi / lambda_  # Wave number

d = 0.5 * lambda_  # Spacing (0.5 lambda)

N = 10  # Number of elements

sll_dB = [20, 30, 40]  # Chebyshev side-lobe levels in dB

tapers = ['Uniform'] + [f'Chebyshev {sll} dB' for sll in sll_dB]  # Tapers

# Dipole element pattern (power pattern)

def dipole_pattern(theta):

    return np.sin(theta)**2

# Chebyshev weights for a given SLL and N

def chebyshev_weights(N, sll_dB):

    R = 10**(sll_dB / 20)  # SLL ratio

    x0 = np.cosh(np.arccosh(R) / (N - 1))  # Chebyshev parameter

    T = chebyt(N - 1)  # Chebyshev polynomial of order N-1

    weights = []

    for m in range(N):

        sum_term = 0

        for p in range(N):

            xp = np.cos(np.pi * (p + 0.5) / N)

            Tp = T(x0 * xp)

            sum_term += Tp * np.cos(np.pi * m * (p + 0.5) / N)

        weight = (1/N) * (1 + 2 * sum_term * (R - 1) / (T(x0) - 1))

        weights.append(weight)

    return np.array(weights)

# Array factor for given weights

def array_factor(theta, N, weights):

    psi = k * d * np.cos(theta)

    af = 0

    for n in range(N):

        af += weights[n] * np.exp(1j * n * psi)

    return np.abs(af)**2 * dipole_pattern(theta)

# Find 3 dB beamwidth and SLL

def compute_beamwidth_sll(theta, af):

    af_max = np.max(af)

    af_dB = 10 * np.log10(af / af_max + 1e-10)

      # Find 3 dB points (where power drops to 1/sqrt(2))

    idx_90 = np.argmin(np.abs(theta - np.pi/2))  # Index at theta = 90 deg

    af_3dB = af_max / np.sqrt(2)

    idx_left = np.where(af[:idx_90] <= af_3dB)[0]

    idx_right = np.where(af[idx_90:] <= af_3dB)[0]

    if len(idx_left) == 0 or len(idx_right) == 0:

        beamwidth = np.nan

    else:

        theta_left = theta[idx_left[-1]]

        theta_right = theta[idx_90 + idx_right[0]]

        beamwidth = np.degrees(theta_right - theta_left)

    # Find SLL (maximum side-lobe peak outside main lobe)

    main_lobe_region = (theta > np.pi/2 - 0.2) & (theta < np.pi/2 + 0.2)  # Approx main lobe

    side_lobes = af_dB[~main_lobe_region]

    sll = np.max(side_lobes) if side_lobes.size > 0 else np.nan

     return beamwidth, sll

# Compute and plot array factor for each Chebyshev taper vs Uniform

theta = np.linspace(0, np.pi, 1000)  # Angle from 0 to 180 degrees
```

```
colors = ['k', 'b']  # Colors for Uniform, Chebyshev

results = {sll: {'Uniform': {'beamwidth': None, 'sll': None}, f'Chebyshev {sll} dB': {'beamwidth': None, 'sll': None}} for sll in sll_dB}

for i, sll in enumerate(sll_dB):

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

    # Compute Uniform taper

    weights_uniform = np.ones(N)

    af_uniform = np.array([array_factor(t, N, weights_uniform) for t in theta])

    af_uniform_dB = 10 * np.log10(af_uniform / np.max(af_uniform) + 1e-10)

    bw_uniform, sll_uniform = compute_beamwidth_sll(theta, af_uniform)

    results[sll]['Uniform']['beamwidth'] = bw_uniform

    results[sll]['Uniform']['sll'] = sll_uniform

    plt.plot(np.degrees(theta), af_uniform_dB, label=f'Uniform: BW={bw_uniform:.2f}°, SLL={sll_uniform:.2f} dB', color=colors[0])

  # Compute Chebyshev taper

    weights_cheby = chebyshev_weights(N, sll)

    af_cheby = np.array([array_factor(t, N, weights_cheby) for t in theta])

    af_cheby_dB = 10 * np.log10(af_cheby / np.max(af_cheby) + 1e-10)

    bw_cheby, sll_cheby = compute_beamwidth_sll(theta, af_cheby)

    results[sll][f'Chebyshev {sll} dB']['beamwidth'] = bw_cheby

    results[sll][f'Chebyshev {sll} dB']['sll'] = sll_cheby

    plt.plot(np.degrees(theta), af_cheby_dB, label=f'Chebyshev {sll} dB: BW={bw_cheby:.2f}°, SLL={sll_cheby:.2f} dB', color=colors[1])

    plt.xlabel('Angle (degrees)')

    plt.ylabel('Array Factor (dB)')

    plt.title(f'Array Factor Comparison: Uniform vs Chebyshev {sll} dB, N = {N}, d = 0.5λ')

    plt.grid(True)

    plt.legend(loc='upper right')  # Legend in top-right corner

    plt.ylim(-60, 0)


plt.savefig(f'array_factor_n10_uniform_vs_chebyshev_{sll}dB.png')

    plt.close()
```