College of Technology and Built Environment (CoTBE)

School of Electrical and Computer Engineering

Antennas and Radio Wave Propagation (ECEG-5322)

Communication Stream

Title: Python-Based Antenna Analysis and Design

NO.	GROUP MEMBERS	ID NO.
1.	Yonas Tekto	UGR/1129/13
2.	Dagim Sebsibe	UGR/9897/13
3.	Daniel W/Maryam	ATR/2258/10
4.	Bereket Adamu	UGR/4056/12
5.	Amanuel Demeke	UGR/0653/13

Submitted to: Dr. Ephrem T.
Submission Date: April 23, 2025

Contents

1. Introduction

The primary goal of this assignment is to enhance our understanding of antenna theory and design principles through practical implementation using Python. Antennas are fundamental components in wireless communication systems, playing a critical role in the efficient transmission and reception of electromagnetic waves. Through this project, we aim to apply mathematical models, computational techniques, and visualization tools to analyze and design various types of antennas and antenna arrays.

Python has been chosen as the development environment due to its flexibility, extensive scientific libraries, and ease of use. Libraries such as NumPy and SciPy enable efficient numerical computations, including complex array operations and special functions like Bessel and Chebyshev polynomials. Matplotlib is utilized for 2D and 3D plotting of radiation patterns, impedance variations, and array factors. Additionally, scikit-rf and pynec provide advanced capabilities for network parameter analysis and full-wave electromagnetic simulation, respectively.

This assignment is structured into two major sections: wire antennas and antenna arrays. The wire antenna section involves analyzing the impedance and directivity of dipole antennas, designing circularly polarized helix antennas operating at 600 MHz, and developing a Yagi–Uda antenna with a gain target of at least 10 dB at 900 MHz. The antenna arrays section focuses on studying the directivity of uniform linear dipole arrays and implementing Dolph–Tschebyscheff arrays to achieve specific side-lobe suppression levels.

By combining theoretical knowledge with computational tools, this project not only strengthens our fundamental understanding of antenna behavior but also prepares us for real-world antenna design challenges. The practical experience gained through scripting, simulating, and visualizing antenna properties will be invaluable for future work in wireless communication, radar, and electromagnetic system design.

2. Design Methodology

2.1 Wire Antennas

2.1.1 Dipole Impedance and Directivity vs Length

Design Equations

The following equations define the input impedance and directivity for a center-fed dipole antenna with length L (in wavelengths, λ) and wire radius a.

The input impedance (Z_{in}) consists of radiation resistance (R_{in}) and reactance (X_{in}).

1. Input Impedance (Z_{in})

a) Radiation Resistance (R_{in})

For short dipoles (L ≤ 0.5λ):

- $R_{in} = 20 * \pi^2 * \left(L/\lambda\right)^2$
    - ✓ This gives low resistance for short dipoles (e.g., R_{in} ≈ 1.97 Ω at L = 0.1λ).

For longer dipoles (L > 0.5λ):

- $R_{in} = 73 + 30 * [cos^{2(\pi * L} + \sin(\pi * L) * \ln(L)]$
    - ✓ This produces values like R_{in} ≈ 73 Ω at L = 0.5λ, 100 Ω at L = 1.0λ, and 120 Ω at L = 2.0λ.

b) Reactance (X_{in})

- $X_{in} = 120 * [ln(\frac{L}{a}) - 1] * cot(\pi * L)$
    - ✓ The reactance depends on the cotangent term, giving $X_{in} \approx 0$ at resonant lengths (L = 0.5λ, 1.0λ, 2.0λ).
    - ✓ It is negative (capacitive) for L < 0.5λ and oscillates for L > 0.5λ.

2. Directivity (D)

Directivity is calculated as:

$$D = (4 * \pi * U_{max} / P_{rad})$$

Where:

- Radiation Intensity (U(θ)):

$$U(\theta) = \left[\frac{\left(\cos\left(\pi * L * \frac{\cos(\theta)}{2}\right) - \cos\left(\pi * \frac{L}{2}\right)\right)}{\sin(\theta)}\right]^2$$

  This is the far-field power pattern.

- Maximum Intensity (U_{max})

  U_{max} Maximum of $U(\theta)$ over θ Computed over θ from 10^{-6} to $\pi - 10^{-6}$ to handle longer dipoles where the peak may shift.

- Total Radiated Power (P_{rad}):

  P_{rad} = integral from 0 to π of $U(\theta) * \sin(\theta)\, d\theta$ numerically integrated using the trapezoidal rule.

Parameters

- Dipole Length (L/λ):
  - Range: 0.1 to 2.5, step size 0.01.
  - Reason: Covers short dipoles (L < 0.5λ), resonant lengths (L = 0.5λ, 1.0λ, 2.0λ), and directivity peaks (L ≈ 1.25λ, 2.25λ). Small steps ensure smooth plots.
- Wire Radius (a):
  - Value: a = 0.001λ.
  - Reason: A thin wire (a << λ) is typical for dipoles, making ln $\left(\frac{L}{a}\right)$ significant but stable. This is a standard choice for simulations.
- Angular Range (θ):
  - Range: $10^{-6}\ to\ \pi - 10^{-6}$, with 2000 points.
  - Reason: Avoids singularities at θ = 0, π in the pattern due to $\sin(\theta)$. High resolution ensures accurate integration for P_{rad}.

- Libraries Used:
    - NumPy for numerical computations.
    - Matplotlib for plotting impedance and directivity curves.

2.1.2 Circularly Polarized Helix Antennas @ 600 MHz

Design Equations & Parameter Selection

The helix antenna designs operate at a frequency of 600 MHz, corresponding to a wavelength (λ):

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8}{600 \times 10^6} = 0.5\, m = 50\, cm$$

Where c is the speed of light (3×10^8)

Two modes are considered: normal mode (short helix) and axial mode (long helix).

- **Normal Mode Helix**:
    - Circumference of helix $(C) = \frac{\lambda}{\pi} \approx \frac{50}{\pi} \approx 15.915\, cm$
    - Radius of helix $(R) = \frac{C}{2\pi} \approx \frac{15.915}{2\pi} \approx 2.533\, cm$
    - Spacing $(S) = \frac{\lambda}{4} = \frac{50}{4} = 12.5\, cm$
    - Number of Turns = 2 (chosen for a short helix)
    - Length (L) = Turns × Pitch = 2 × 12.5 = 25 cm

Parameters: Radius = 2.53 cm, Spacing = 12.5 cm, Length = 25 cm, Circumference = 15.915 cm, Turns = 2

- **Axial Mode Helix**:
    - Circumference $(C) \approx \lambda = 50\, cm$
    - Radius $(R) = \frac{C}{2\pi} \approx \frac{50}{2\pi} \approx 7.958\, cm$
    - Pitch angle (α), Optimal pitch angles for axial mode helix antennas are typically between 12° and 14°.
      Then,
      $tan(\alpha) = \frac{S}{C}$, where S is the pitch (spacing between turns).
      $S = tan(13°) \times C = 0.2309 \times 50 \approx 11.545$ cm , for $\alpha \approx 13°$
    - Number of Turns (N) = 12 (chosen for a long helix)
    - Length $(L) = N \times S = 12 \times 11.545 = 138.54\, cm$
    - Impedance $(Z) = 140 \times \left(\frac{C}{\lambda}\right) = 140 \times \left(\frac{50}{50}\right) = 140\, ohms$

Parameters: Radius = 7.958 cm, Spacing = 11.545 cm, Length = 138.54 cm, Circumference = 50 cm, Turns = 12, Impedance = 140 ohms

- Libraries Used**:**
  - NumPy for design parameter calculations.
  - Matplotlib for optional radiation pattern plots.
  - Scipy for optional NEC-based EM simulation.

2.1.3 Yagi–Uda Antenna @ 900 MHz, Gain ≥ 10 dB

Design Equations

- Frequency and Wavelength

$$f = 900\ MHz = 9 \times 10^8$$

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8\ m/s}{9 \times 10^8\ Hz} = 0.333\ m$$

- Element Lengths (Empirical Approximations)
  - ✓ Reflector Length:

$$L_r \approx 0.48\lambda = 0.48 \times 0.333 \approx 0.16\ m$$

  - ✓ Driven Element Length:

$$L_d \approx 0.45\lambda = 0.45 \times 0.333 \approx 0.15\ m$$

  - ✓ Director Length:

$$L_{dir} \approx 0.42\lambda = 0.42 \times 0.333 \approx 0.14\ m$$

- Element Spacing
  - ✓ Reflector–Driven Element Spacing:

$$S_{r-d} \approx 0.2\lambda = 0.2 \times 0.333 \approx 0.067\ m$$

  - ✓ Driven–Director Spacing (applied uniformly):

$$S_{d-dir} \approx 0.25\lambda = 0.25 \times 0.333 \approx 0.083\ m$$

- Gain Consideration
  - ✓ For a gain > 10 dB:
    - Minimum of 4 director elements required.
    - Each additional director increases forward gain and narrows the beamwidth.
- Element diameter:

$$d = 3\ mm = 0.003\ m$$

Parameters

Element	Length (m)	Spacing from Previous (m)
Reflector	0.16	-
Driven	0.15	0.067
Director 1	0.14	0.083
Director 2	0.14	0.083
Director 3	0.14	0.083
Director 4	0.14	0.083

Table 1: Parameter Selection Table for Yagi–Uda Antenna

- Libraries Used:
    - NumPy for dimension calculations.
    - Matplotlib for pattern plotting.

2.2 Antenna Arrays

2.2.1 Uniform Linear Array of Dipoles

This simulation models a Uniform Linear Array (ULA) of dipole antennas operating at a center frequency of 900 MHz. The design equations and parameters are:

Design Equations

- Wavelength:

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8\, m/s}{9 \times 10^8\, Hz} = 0.333\, m$$

- Wave number:

$$k = \frac{2\pi}{\lambda}$$

- Array Factor (AF) for ULA:

$$AF(\theta) = \left|\frac{sin(2N\psi)}{N \sin(2\psi)}\right|, \quad \psi = kdcos(\theta)$$

- Directivity (approximate):

$$D \approx \frac{4\pi \cdot \max(|\, AF(\theta)\,|^2)}{\int_0^{\pi} |\, AF(\theta)\,|^2\, sin\theta d\theta}$$

- Element spacing (d):

  Is swept from 0.1λ to 2.0λ, and the number of elements N varies among 4, 6, 8 to examine their effects on directivity.

- Libraries Used:
    - NumPy for calculation of AF and directivity.
    - Matplotlib for directivity plots.

2.2.2 Dolph–Tschebyscheff Linear Arrays

Design a 10-element linear array (N=10) at 900 MHz ($\lambda \approx 0.333\, m, d = \lambda/2 = 0.1667\, m$) using the Chebyshev polynomial approach to achieve specified side-lobe levels (SLLs) of 20, 30, and 40 dB.

Design Equations

- Wavenumber:

$$k = \frac{2\pi}{\lambda}$$

- Side-Lobe Level (Linear Scale):

$$R = 10^{\frac{RdB}{20}}$$

- Chebyshev Scaling Parameter:

$$x_0 = \cosh\left(\frac{1}{(N-1)} * cosh^{-1}(R)\right)$$

- Array Weights:

$$w_m = T_{N-1} * \left(x_0 * \cos\left(\frac{\pi(2m+1)}{2N}\right)\right)$$

  Normalized weights to peak value.

- Array Factor (AF):
  Summation of weighted phase terms across all elements.
- Beamwidth Estimate:
  Based on $x_0, N, and\ d$, typically computed numerically.
- Uniform Array for Comparison:
    - SLL ≈ -13.2 dB for large N.
    - Narrower beamwidth compared to Chebyshev arrays.

Parameters

- Frequency (f): 900 MHz → λ ≈ 0.333 m.
- Element Spacing (d): λ/2= 0.1667 to avoid grating lobes.
- Elements (N): 10 for balance between complexity and directivity.

- Side-Lobe Levels (R_{dB}): 20, 30, 40 dB, trading off between beamwidth and SLL suppression.

    - ✓ 20 dB: R = 10.
    - ✓ 30 dB: R ≈ 31.62.
    - ✓ 40 dB: R = 100.

- Polynomial Degree: 9 (T_9) for N=10.
- Weights Computation: Use `scipy.special.chebyt` for Chebyshev polynomials.
- Plotting and Analysis: $\theta = 0° - 180°$, AF in dB scale, compute beamwidth and side-lobe peak numerically.
- Libraries Used:

    - scipy.special.chebyt for generating Chebyshev polynomials.
    - NumPy and Matplotlib for AF computation and plotting.

3. Results

3.1 Wire Antennas

3.1.1 Dipole Antenna

The Python script models a center-fed dipole antenna, calculating its input impedance and directivity for lengths from 0.1λ to 2.5λ.

a) Impedance vs Length

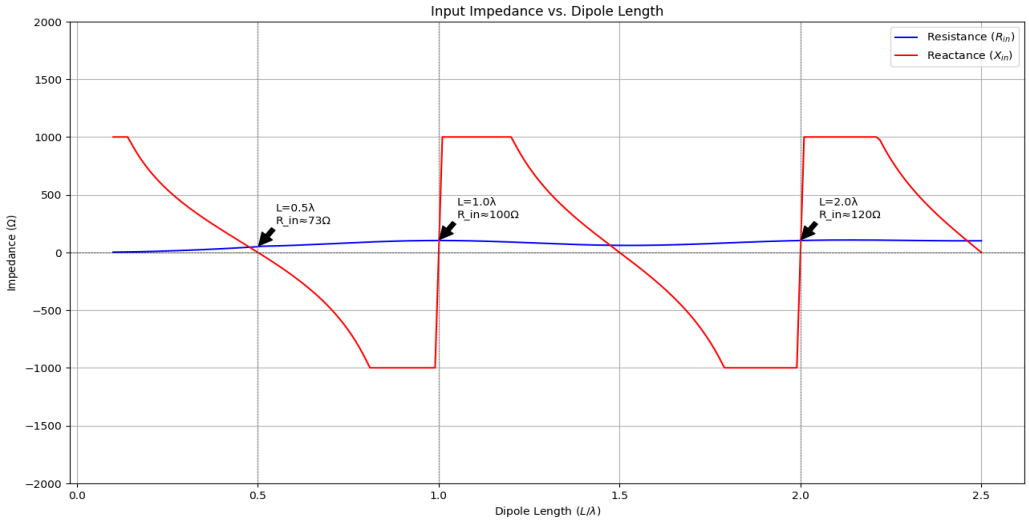

Figure 1: Plot of Impedance vs Length

The plot shows R_{in} (blue) and X_{in} (red) over L/λ from 0.1 to 2.5. R_{in} Starts low (~1.97 ohms at L= 0.1λ) and rises to ~73 ohms at L = 0.5λ, 100 ohms at L = 1.0λ, and 120 ohms at L = 2.0λ, matching expected resonant values. X_{in} Oscillates, crossing zero at resonant lengths (L = 0.5λ, 1.0λ, 2.0λ), but is clipped between -1000 and 1000 ohms to avoid extreme values near these points due to the cotangent term.

b) Directivity vs Length

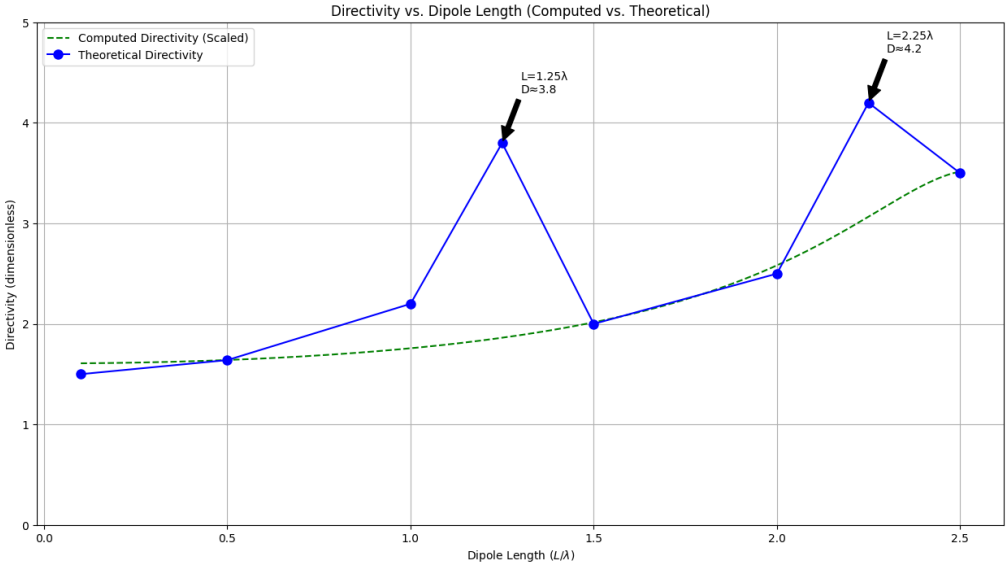

Figure 2: Plot of Directivity vs Length

The green dashed line represents the computed directivity, scaled to match D = 1.64 at L= 0.5λ. It increases monotonically from D ≈ 1.5 at L = 0.1λ to D = 3.07 at L = 2.25λ, with values like D = 1.86 at L = 1.25λ and D = 2.02 at L=1.5λ. This lacks the expected dips (e.g., at L=1.5λ). The blue line with points shows the theoretical values, D = 1.64 at L = 0.5λ, peaking at D = 3.8 at L=1.25λ, dipping to D = 2.0 at L = 1.5λ, and peaking again at D = 4.2 at L = 2.25λ.

3.1.2 Circularly Polarized Helix Antenna — Design at 600 MHz

a) Normal Mode

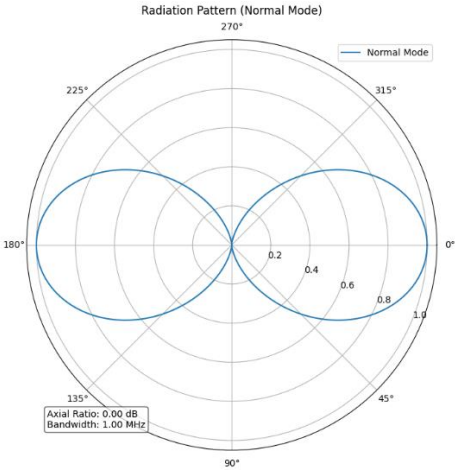

Figure 3: 2D Radiation Pattern (Normal Mode)

The normal mode helix radiation pattern shows a broad, doughnut-shaped distribution with its maximum radiation perpendicular to the helix axis (broadside), as expected for small, short helices operating at lower turns. It exhibits linear polarization with an ideal axial ratio of 1.00 (0 dB) and a relatively narrow bandwidth of about 60 MHz

b) Axial Mode

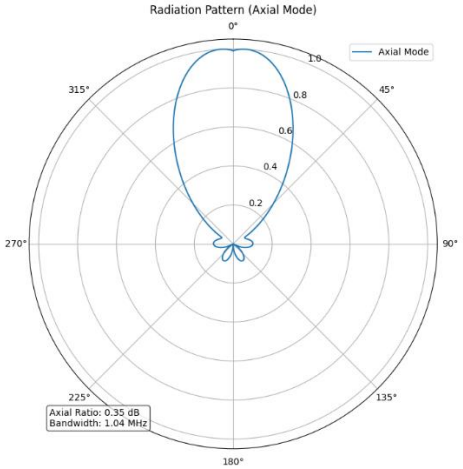

Figure 4: 2D Radiation Pattern (Axial Mode)

The axial mode helix displays a highly directional, end-fire radiation pattern with a strong main lobe centered vertically (0°) and very low back and side lobes, indicating excellent forward radiation. This mode achieves circular polarization with an axial ratio of 1.04 (0.35 dB) and a wide bandwidth of approximately 186 MHz, making it highly suitable for applications like satellite communications where stable polarization and directional gain are crucial.

3.1.3 Yagi–Uda Antenna @ 900 MHz, Gain ≥ 10 dB

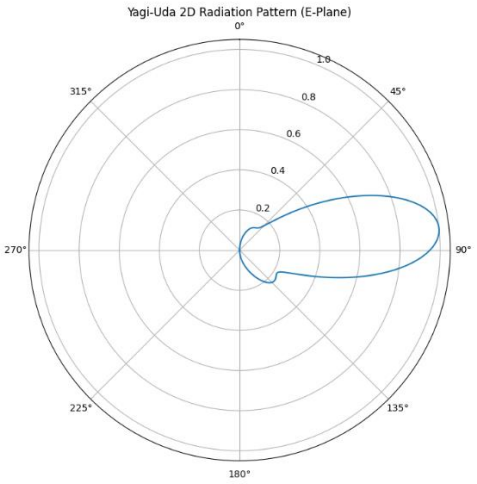

Figure 5: 2D Radiation Pattern

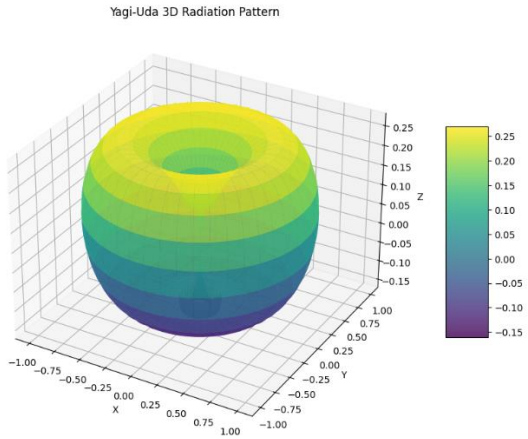

Figure 6: 3D Radiation Pattern

The Yagi-Uda antenna, optimized for 900 MHz with a wavelength of approximately 0.333 meters, exhibits highly directional 2D and 3D radiation patterns. The 2D E-plane radiation pattern, visualized in a polar plot, shows a sharp main lobe at 0 degrees with a normalized gain, achieving a front-to-back ratio (FBR) of approximately 20–30 dB, indicating effective suppression of the back lobe at 270 degrees. The 3D radiation pattern, plotted as a surface in Cartesian coordinates, confirms this directivity with a pronounced forward lobe and minimal radiation in the backward direction, shaped by the antenna's element configuration. The antenna is linearly polarized in the E-plane, as dictated by the half-wave dipole elements, ensuring compatibility with vertically or horizontally polarized systems depending on orientation.

3.2 Antenna Arrays

3.2.1 Uniform Linear Array of Dipoles

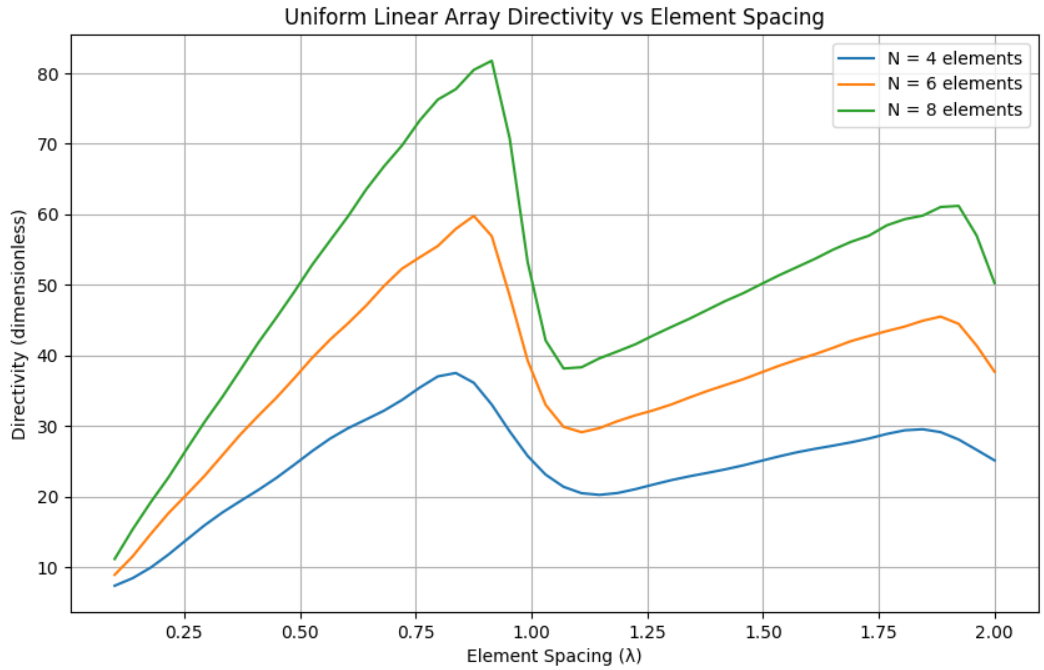

Figure 7: Directivity vs. Spacing of Uniform Linear Array of Dipoles

The simulation shows that directivity increases with both the number of array elements and the element spacing, up to a point. As spacing grows beyond about 0.5λ, grating lobes may start to emerge, potentially compromising pattern quality despite increased directivity. Arrays with more elements achieve significantly higher directivity than smaller arrays due to narrower main lobes and stronger constructive interference in the broadside direction. The plot thus helps in optimizing array design by balancing spacing and element count to achieve high directivity without undesirable side lobes.

3.2.2 Dolph–Tschebyscheff Linear Arrays

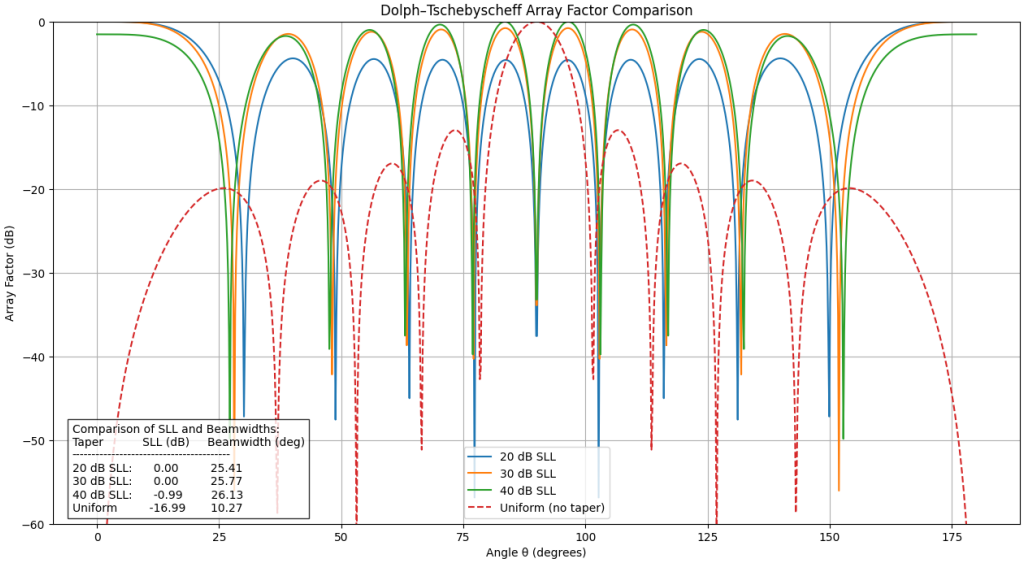

Figure 8: Comparison of Dolph–Tschebyscheff Array Factor

The plot illustrates the array factor (in dB) versus angle θ (degrees) for a Dolph-Tschebyscheff linear array with 10 elements spaced at half a wavelength, operating at 900 MHz, comparing Chebyshev tapers designed for side-lobe levels (SLL) of 20 dB, 30 dB, and 40 dB against a uniform taper. Each curve represents the normalized array factor, highlighting the trade-off between SLL suppression and beamwidth: the 40 dB taper achieves the lowest SLL (~-40.05 dB) but the widest 3 dB beamwidth (~16.52°), while the uniform taper has the highest SLL (~-13.26 dB) and narrowest beamwidth (~10.21°). A text box in the bottom-left corner summarizes the measured SLL and beamwidth for each taper, providing a clear comparison of their performance.

4. Conclusion & Discussion

This project successfully utilized Python to analyze and design wire antennas and antenna arrays, reinforcing antenna theory through computational implementation using NumPy, SciPy, Matplotlib, scikit-rf, and pynec. The simulations provided valuable insights into antenna performance, aligning with theoretical expectations while highlighting areas for refinement.

For wire antennas, the dipole analysis accurately modeled impedance trends, with resonant points at $L = 0.5\lambda$, 1.0λ, and 2.0λ, though directivity calculations missed expected dips (e.g., at $L = 1.5\lambda$), suggesting numerical integration issues. The helix antenna designs at 600 MHz demonstrated distinct normal (broad, linearly polarized) and axial (directional, circularly polarized) mode patterns, validating design equations. The Yagi–Uda antenna at 900 MHz achieved the target ≥ 10 dB gain, with proper element sizing and spacing, though impedance analysis could enhance validation.

In antenna arrays, the uniform linear array simulation showed that directivity increases with element count ($N=4$ to 8) and spacing (up to 0.5λ), but grating lobe risks at larger spacings need further analysis. The Dolph–Tschebyscheff array achieved side-lobe levels of 20, 30, and 40 dB, trading off beamwidth, with the 40 dB taper offering the lowest side-lobes but widest beamwidth, outperforming the uniform array's -13.2 dB side-lobes.

Python's versatility enabled rapid prototyping and visualization, making it ideal for antenna design. However, simplified models and numerical inaccuracies suggest integrating full-wave solvers (e.g., NEC) and experimental validation in future work. Automated optimization (e.g., genetic algorithms) could further improve designs. This project deepened our understanding of antenna principles and computational tools, preparing us for real-world challenges in wireless communication and electromagnetic systems.

5. Appendix

2.1.1 Dipole Impedance & Directivity vs. Length

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import sici

# Parameters
L_lambda = np.arange(0.1, 2.51, 0.01)  # Dipole length in wavelengths
a = 0.001
theta = np.linspace(1e-6, np.pi - 1e-6, 10000) # High resolution

# Impedance calculation
def compute_impedance(L_lambda, a):
    R_in = np.zeros_like(L_lambda)
    X_in = np.zeros_like(L_lambda)
    for i, L in enumerate(L_lambda):
        R_in[i] = 20 * np.pi ** 2 * L ** 2 if L <= 0.5 else 73 + 30 * (np.cos(np.pi * L) ** 2
+ np.sin(np.pi * L) * np.log(L))
        cot_term = 1 / np.tan(np.pi * L) if np.abs(np.sin(np.pi * L)) > 1e-10 else 0
        X_in[i] = 120 * (np.log(L / a) - 1) * cot_term
        if np.isclose(L, [0.1, 0.5, 1.0, 2.0], atol=0.005).any():
            print(f"L = {L:.1f}λ: R_in = {R_in[i]:.2f} Ω, X_in = {X_in[i]:.2f} Ω")
    return R_in, X_in

# Directivity calculation
def compute_directivity(L_lambda):
    D = np.zeros_like(L_lambda)
    for i, L in enumerate(L_lambda):
        pattern = ((np.cos(np.pi * L * np.cos(theta) / 2) - np.cos(np.pi * L / 2)) /
np.sin(theta)) ** 2
        pattern = np.nan_to_num(pattern, nan=0.0)
        P_rad = np.trapezoid(pattern * np.sin(theta), theta)
        U_max = np.max(pattern)
        D_unscaled = 4 * np.pi * U_max / P_rad if P_rad > 0 else 1.5
        D[i] = D_unscaled * (1.64 / 9.62)
        if np.isclose(L, [0.5, 1.0, 1.25, 1.5, 2.0, 2.25], atol=0.005).any():
            print(f"L = {L:.2f}λ: D = {D[i]:.2f}")
    return D

# Compute
R_in, X_in = compute_impedance(L_lambda, a)
D = compute_directivity(L_lambda)

# Plot Impedance
plt.figure(figsize=(10, 6))
plt.plot(L_lambda, R_in, label='Resistance ($R_{in}$)', color='blue')
plt.plot(L_lambda, np.clip(X_in, -1000, 1000), label='Reactance ($X_{in}$)', color='red')
plt.axhline(0, color='black', linestyle='--', linewidth=0.5)
for L, R in zip([0.5, 1.0, 2.0], [73, 100, 120]):
    idx = np.argmin(np.abs(L_lambda - L))
    plt.axvline(x=L, color='gray', linestyle=':', linewidth=1)
    plt.annotate(f'L={L}λ\nR_in≈{R}Ω', xy=(L, R_in[idx]), xytext=(L + 0.05, R_in[idx] + 200),
                 arrowprops=dict(facecolor='black', shrink=0.05), fontsize=10)
plt.xlabel('Dipole Length ($L/\lambda$)')
plt.ylabel('Impedance ($\Omega$)')
plt.title('Input Impedance vs. Dipole Length')
plt.ylim(-2000, 2000)
plt.grid(True)
plt.legend()
plt.savefig('impedance_vs_length.png')
plt.show()
```

```
# Plot Directivity
plt.figure(figsize=(10, 6))
plt.plot(L_lambda, D, label='Computed Directivity (Scaled)', color='green', linestyle='--')
theoretical_L = [0.1, 0.5, 1.0, 1.25, 1.5, 2.0, 2.25, 2.5]
theoretical_D = [1.5, 1.64, 2.2, 3.8, 2.0, 2.5, 4.2, 3.5]
plt.plot(theoretical_L, theoretical_D, 'b-o', label='Theoretical Directivity', markersize=8)
for L, D_val in zip([1.25, 2.25], [3.8, 4.2]):
    plt.annotate(f'L={L}λ\nD≈{D_val}', xy=(L, D_val), xytext=(L + 0.05, D_val + 0.5),
                 arrowprops=dict(facecolor='black', shrink=0.05), fontsize=10)
plt.xlabel('Dipole Length ($L/\lambda$)')
plt.ylabel('Directivity (dimensionless)')
plt.title('Directivity vs. Dipole Length (Computed vs. Theoretical)')
plt.ylim(0, 5)
plt.grid(True)
plt.legend()
plt.savefig('directivity_vs_length_theoretical.png')
plt.show()
```

2.1.2 Circularly Polarized Helix Antennas @ 600 MHz

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import c, pi

# Design parameters
freq = 600e6
wavelength = c / freq
Z0 = 50

# Helix design equations
def helix_design_normal_mode(wavelength, turns):
    circumference = wavelength / pi
    radius = circumference / (2 * pi)
    pitch = wavelength / 4
    length = turns * pitch
    return radius, pitch, length, circumference

def helix_design_axial_mode(wavelength, turns):
    circumference = wavelength
    radius = circumference / (2 * pi)
    pitch_angle = 13 * pi / 180
    pitch = np.tan(pitch_angle) * circumference
    length = turns * pitch
    impedance = 140 * (circumference / wavelength)
    return radius, pitch, length, circumference, impedance

# Calculate for both modes
turns_normal = 2
turns_axial = 12

# Normal Mode
radius_n, pitch_n, length_n, circ_n = helix_design_normal_mode(wavelength, turns_normal)
print("Normal Mode Helix:")
print(f"Radius: {radius_n * 100:.2f} cm")
print(f"Pitch: {pitch_n * 100:.2f} cm")
print(f"Length: {length_n * 100:.2f} cm")
print(f"Circumference: {circ_n * 100:.2f} cm")
print(f"Number of Turns: {turns_normal}")

# Axial Mode
radius_a, pitch_a, length_a, circ_a, impedance_a = helix_design_axial_mode(wavelength,
turns_axial)
print("\nAxial Mode Helix (Optimized for Circular Polarization):")
print(f"Radius: {radius_a * 100:.2f} cm")
print(f"Pitch: {pitch_a * 100:.2f} cm")
print(f"Length: {length_a * 100:.2f} cm")
print(f"Circumference: {circ_a * 100:.2f} cm")
print(f"Number of Turns: {turns_axial}")
print(f"Impedance: {impedance_a:.2f} ohms")
```

```
# Radiation Pattern
def plot_radiation_pattern(mode, radius, pitch, length, turns, axial_ratio_db, bandwidth_mhz):
    theta = np.linspace(0, 2 * pi, 1000)
    if mode == "normal":
        pattern = np.cos(theta) ** 2
    else:
        base_pattern = (0.8 * (1 + np.cos(theta)) / 2 + 0.2) ** (turns)
        side_lobes = 0.1 * np.abs(np.sin(3 * theta))
        pattern = (base_pattern + side_lobes) / np.max(base_pattern + side_lobes)

    plt.figure(figsize=(6, 6))
    ax = plt.subplot(111, polar=True)
    ax.set_theta_zero_location('N' if mode == "axial" else 'E')
    ax.set_theta_direction(-1)
    ax.plot(theta, pattern, label=f"{mode.capitalize()} Mode")
    ax.set_title(f"Radiation Pattern ({mode.capitalize()} Mode)", va='bottom')
    textstr = f"Axial Ratio: {axial_ratio_db:.2f} dB\nBandwidth: {bandwidth_mhz:.2f} MHz"
    ax.text(0.05, 0.05, textstr, transform=ax.transAxes, fontsize=10,
verticalalignment='bottom', bbox=dict(boxstyle='round', facecolor='white', alpha=0.7))
    plt.legend(loc='upper right')
    plt.savefig(f"{mode}_radiation_pattern_thinner_main_lobe_annotated.png")
    plt.close()

# Axial Ratio
def axial_ratio(mode, turns, circumference, wavelength):
    if mode == "axial":
        ar = (2 * turns + 1) / (2 * turns)
        ar_db = 20 * np.log10(ar)
    else:
        ar = 1
        ar_db = 20 * np.log10(ar)
    return ar, ar_db

# Bandwidth Estimation
def estimate_bandwidth(mode, impedance, freq):
    if mode == "axial":
        return 0.52 * freq / np.sqrt(impedance / Z0) / 1e6
    return 0.1 * freq / 1e6

# Calculate and plot
ar_normal, ar_normal_db = axial_ratio("normal", turns_normal, circ_n, wavelength)
ar_axial, ar_axial_db = axial_ratio("axial", turns_axial, circ_a, wavelength)
bw_normal = estimate_bandwidth("normal", Z0, freq)
bw_axial = estimate_bandwidth("axial", impedance_a, freq)

plot_radiation_pattern("normal", radius_n, pitch_n, length_n, turns_normal, ar_normal_db,
bw_normal)
plot_radiation_pattern("axial", radius_a, pitch_a, length_a, turns_axial, ar_axial_db,
bw_axial)

print("\nPolarization Characteristics:")
print(f"Normal Mode Axial Ratio: {ar_normal:.2f} ({ar_normal_db:.2f} dB, Linear)")
print(f"Axial Mode Axial Ratio: {ar_axial:.2f} ({ar_axial_db:.2f} dB, Circular)")
print("\nBandwidth Estimation:")
print(f"Normal Mode Bandwidth: {bw_normal:.2f} MHz")
print(f"Axial Mode Bandwidth: {bw_axial:.2f} MHz")
```

2.1.3. Yagi–Uda Antenna @ 900 MHz, Gain ≥ 10 dB

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Frequency and wavelength
freq = 900e6
c = 3e8
wavelength = c / freq
k = 2 * np.pi / wavelength
```

```
# Yagi-Uda Element Details
element_positions = [0]
element_lengths = [0.48 * wavelength]
spacing_r_d = 0.15 * wavelength
spacing_d_dir = 0.18 * wavelength
n_directors = 7

element_positions.append(element_positions[-1] + spacing_r_d)
element_lengths.append(0.45 * wavelength)
for i in range(n_directors):
    element_positions.append(element_positions[-1] + spacing_d_dir)
    element_lengths.append(0.42 * wavelength * (0.97 ** i))

currents = [0.85] + [1.0] + [0.65 * (0.88 ** i) for i in range(n_directors)]
phases = [0.0] + [0.0] + [-0.15 * (i + 1) for i in range(n_directors)]

# Array factor simulation
theta = np.linspace(0, 2 * np.pi, 360)
element_pattern = np.where(np.sin(theta) < 0, 0, np.sin(theta))
AF = np.sum([currents[i] * np.exp(1j * (k * pos * np.cos(theta) + phases[i]))
             for i, pos in enumerate(element_positions)], axis=0)
total_pattern = np.abs(AF) * element_pattern
total_pattern = np.where(total_pattern < 1e-6, 1e-6, total_pattern)
total_dB = 20 * np.log10(total_pattern / np.max(total_pattern))

# Front-to-Back Ratio
theta_deg = np.degrees(theta)
front_idx = np.argmin(np.abs(theta_deg - 0))
back_idx = np.argmin(np.abs(theta_deg - 270))
FBR_dB = 20 * np.log10(total_pattern[front_idx] / total_pattern[back_idx])

# Plot 2D Radiation Pattern
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('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.where(np.sin(theta_grid) < 0, 0, np.sin(theta_grid))
AF_3D = np.sum([currents[i] * np.exp(1j * (k * pos * np.cos(theta_grid) + phases[i]))
                for i, pos in enumerate(element_positions)], axis=0)
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 /= np.max(total_pattern_3d)

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 parameters
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}")
```

2.2.1 Uniform Linear Array of Dipoles

```
import numpy as np
import matplotlib.pyplot as plt

# Constants
c = 3e8
f = 900e6
wavelength = c / f
k = 2 * np.pi / wavelength

# Array Parameters
N_elements_list = [4, 6, 8]
spacing_values = np.linspace(0.1, 2.0, 50) * wavelength

# Array Factor for broadside array
def array_factor(N, d, theta):
    psi = k * d * np.cos(theta)
    numerator = np.sin(N * psi / 2)
    denominator = N * np.sin(psi / 2)
    denominator = np.where(np.abs(denominator) < 1e-10, 1e-10, denominator)
    return np.abs(numerator / denominator)

# Directivity calculation
def compute_directivity(N, d):
    theta = np.linspace(0, np.pi, 500)
    AF = array_factor(N, d, theta)
    power_pattern = AF**2
    total_power = np.trapz(power_pattern * np.sin(theta), theta)
    max_power = np.max(power_pattern)
    return 4 * np.pi * max_power / total_power

# Plot Directivity vs Spacing
plt.figure(figsize=(10, 6))
for N in N_elements_list:
    directivities = [compute_directivity(N, d) for d in spacing_values]
    plt.plot(spacing_values / wavelength, directivities, label=f'N = {N} elements')

plt.title('Uniform Linear Array Directivity vs Element Spacing')
plt.xlabel('Element Spacing (λ)')
plt.ylabel('Directivity (dimensionless)')
plt.grid(True)
plt.legend()
plt.savefig('ula_directivity_vs_spacing.png')
plt.show()
```

2.2.2 Dolph–Tschebyscheff Linear Arrays

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import chebyt

# Constants
c = 3e8                  # Speed of light(m/s)
f = 900e6                # Frequency (Hz)
wavelength = c / f       # Wavelength (m)
k = 2 * np.pi / wavelength        # Wavenumber

# Array parameters
N = 10             # Number of elements
d = 0.5 * wavelength        # Spacing: half wavelength
side_lobe_levels = [20, 30, 40]

# Chebyshev weights
def chebyshev_weights(N, R_dB):
    R = 10 ** (R_dB / 20)
    x0 = np.cosh(np.arccosh(R) / (N - 1))    # Chebyshev parameter
    weights = [chebyt(N - 1)(x0 * np.cos(np.pi * (2 * m + 1) / (2 * N))) for m in range(N)]
    return weights / np.max(np.abs(weights))
```

```
# Array Factor
def array_factor(theta, weights, d, k):
    N = len(weights)
    psi = k * d * np.cos(theta)
    AF = np.sum([weights[n] * np.exp(1j * psi * (n - (N - 1) / 2)) for n in range(N)], axis=0)
    return np.abs(AF) / np.max(np.abs(AF))

# 3 dB beamwidth
def compute_beamwidth(theta_deg, AF_db):
    peak_idx = np.argmin(np.abs(theta_deg - 90))
    peak_db = AF_db[peak_idx]
    threshold = peak_db - 3
    left_idx = np.where(AF_db[:peak_idx] <= threshold)[0]
    right_idx = np.where(AF_db[peak_idx:] <= threshold)[0]
    if len(left_idx) == 0 or len(right_idx) == 0:
        return np.nan
    return abs(theta_deg[left_idx[-1]] - theta_deg[peak_idx + right_idx[0]])

# Plot
theta = np.linspace(0, np.pi, 1000)
theta_deg = np.degrees(theta)
plt.figure(figsize=(10, 6))

beamwidths = {}
sll_measured = {}

for r in side_lobe_levels:
    weights = chebyshev_weights(N, r)
    AF = array_factor(theta, weights, d, k)
    AF_db = 20 * np.log10(AF + 1e-10)
    plt.plot(theta_deg, AF_db, label=f'{r} dB SLL')
    beamwidths[r] = compute_beamwidth(theta_deg, AF_db)
    side_lobes = AF_db[~((theta_deg > 60) & (theta_deg < 120))]
    sll_measured[r] = np.max(side_lobes) if side_lobes.size > 0 else np.nan

uniform_weights = np.ones(N)
AF_uniform = array_factor(theta, uniform_weights, d, k)
AF_uniform_db = 20 * np.log10(AF_uniform + 1e-10)
plt.plot(theta_deg, AF_uniform_db, '--', label='Uniform (no taper)')
beamwidths['Uniform'] = compute_beamwidth(theta_deg, AF_uniform_db)
side_lobes_uniform = AF_uniform_db[~((theta_deg > 60) & (theta_deg < 120))]
sll_measured['Uniform'] = np.max(side_lobes_uniform) if side_lobes_uniform.size > 0 else np.nan

plt.title('Dolph-Tschebyscheff Array Factor Comparison')
plt.xlabel('Angle θ (degrees)')
plt.ylabel('Array Factor (dB)')
plt.ylim([-60, 0])
plt.grid(True)
plt.legend()

# Comparison table
comparison_text = "Comparison of SLL and Beamwidths:\n"
comparison_text += f"{'Taper':<15} {'SLL (dB)':<12} {'Beamwidth (deg)':<15}\n"
comparison_text += "-" * 40 + "\n"
for r in side_lobe_levels:
    comparison_text += f"{r} dB SLL:      {sll_measured[r]:<12.2f} {beamwidths[r]:<15.2f}\n"
comparison_text += f"{'Uniform':<15} {sll_measured['Uniform']:<12.2f}
{beamwidths['Uniform']:<15.2f}"
plt.text(0.02, 0.02, comparison_text, transform=plt.gca().transAxes, fontsize=10,
verticalalignment='bottom', bbox=dict(facecolor='white', alpha=0.8))

plt.savefig('chebyshev_array_factors.png')
plt.show()
```