1. INRODUCTION

Aim of the tasks and tools used

1. Wire Antennas

- Aim: Analyze how the input impedance and maximum directivity of a dipole antenna vary with its length (from 0.1λ to 2.5λ).
- Tools Used:
    - NumPy: For numerical calculations and array handling.
    - Matplotlib: For plotting impedance and directivity vs. length graphs.

2. Circularly Polarized Helix Antennas @ 600 MHz

- Aim: Design and analyze both Normal Mode (short helix) and Axial Mode (long helix) helical antennas operating at 600 MHz.
- Tools Used:
    - NumPy / SciPy: To implement design equations for helix parameters (radius, pitch, number of turns).
    - Matplotlib: To visualize radiation patterns and axial ratio plots.
    - 4NEC2 antenna modeler and optimizer: For simulating 3D/2D radiation patterns, axial ratio, and input bandwidth.

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

- Aim: Design a Yagi–Uda antenna at 900 MHz targeting a gain of at least 10 dB, including element length/spacing estimation and pattern analysis.
- Tools Used:
    - NumPy: For calculating dimensions.
    - Matplotlib: For plotting radiation patterns.
    - 4NEC2 antenna modeler and optimizer: For simulating gain, impedance bandwidth, and pattern verification.

3.2 Antenna Arrays

1. Uniform Linear Array of Dipoles

- Aim: Study how maximum directivity varies with element spacing and array size (spacing from 0.1λ to 2.0λ).
- Tools Used:
    - NumPy / SciPy: For computing array factors and directivity.
    - Matplotlib: To plot directivity vs. spacing curves.

2. Dolph–Tschebyscheff Linear Arrays

- Aim: Implement Dolph–Tschebyscheff tapering for linear arrays to achieve controlled side-lobe levels (20 dB, 30 dB, 40 dB), and compare with uniform taper.
- Tools Used:
    - SciPy (scipy.special.chebyt): For generating Chebyshev polynomial tapers.
    - NumPy: For array factor computation.
    - Matplotlib: To plot array factors and compare beamwidth/side-lobe levels.

2. Design Methodology

2.1. Mathematical Analysis

2.1.1. Wire Antennas

- **For the Input Impedance**

The input impedance of a dipole antenna is:

$$Z_{in} = R_{in} + jX_{in}$$

Where:

- Rin: input resistance (real part)
- Xin: input reactance (imaginary part)

These quantities depend on the electrical length L/λ of the dipole.

For a center-fed, thin dipole in free space, there is no simple closed-form formula, but approximate methods and numerical models (like Hallén's or Pocklington's integral equations) give us expressions used to estimate impedance.

The code used is based on Solving Hallén's Integral Equation i.e. For a thin wire dipole of length L and radius a, carrying current I(z), the input impedance is computed using:

$$Z_{in} = \frac{V_0}{I(0)}$$

Where:

- V_0: the excitation voltage at the feed point
- I(0): current at the feed (center of the dipole)

The current distribution is approximated as:

$$I(z) = I_0 \sin\left[k\left(\frac{L}{2} - |z|\right)\right]$$

This assumes:

- Center-fed
- Sinusoidal current distribution
- Very thin wire $a \ll \lambda$

Approximate Input Impedance Expressions

These are empirical or semi-analytical approximations based on curve fitting to full-wave solutions. For dipoles shorter than ~0.4λ:

$$R_{in} = 20\left(\frac{L}{\lambda}\right)^2$$

$$X_{in} = -120\left(\frac{L}{\lambda}\right)$$

This predicts:

- Small radiation resistance
- Capacitive reactance (negative)

For half-wave dipole ($L \approx 0.47\lambda$):

$$Zin \approx 73+j42.5\ \Omega$$

But if exactly resonant (≈0.485λ depending on thickness), the reactance becomes zero:

$$Zin \approx 73+j0\ \Omega$$

So What Did the Code Actually Do?

- Made assumptions on sinusoidal current distribution
- Used it to calculate on sinusoidal current distribution
- Then computed the Input impedance from voltage to current ratio at the feed

2.1.2 Maximum Directivity

The directivity D of a dipole antenna is calculated using the following equation:

$$D = \frac{2.U_{max}}{\int_0^{\pi} U(\theta) sin\theta \, d\theta}$$

Where:

$U(\theta) \propto |F(\theta)|^2$ is the normalized radiation intensity

$F(\theta)$ is the far-field pattern functionof a dipole length L:

$$F(\theta) = \frac{\cos\left(\frac{KL}{2} cos\,\theta\right) - \cos(\frac{kL}{2})}{\sin\theta}$$

k = 2π (all lengths are normalized in terms of lambda)

The maximum value U_max is obtained from ⟦|F(θ)|⟧^2, and the denominator is computed numerically through numerical integration. The final directivity is usually transformed to dBi (decibels isotropic) using:

$$D_{dBi} = 10.\log_{10}(D)$$

2.2. Circularly Polarized Helix Antennas

A. Design parametrs for Short Helix

Basic parameters:- Operating frequency(f) = 600 MHz and a wavelength (c/f) of 0.5m.

Total Length < λ/4, circumference C << λ

Radius (a) : typically λ/100 to λ/30

Pitch(S) : vertical spacing between turns (small as radius)

Number of turns (N): small (2 to 5)

Let's choose:

- Radius a = /40 = 0.5/40 =0.125 cm
- Pitch S = 1.25cm
- Turns N=5

Circumference= 2πa ≈ 0.0785 m

Total Height h= N.S = 5.125 = 6.25cm

B. Design parameters for Long Helix

Circumference C ≈ λ. 0.8 to λ. 1.2, choosing C= = 0.5m => ∝ = C/(2 π) ≈ 0.0796 m

Pitch angle(∝) = 13 degree

Tan(∝) = S/ (2 π (∝)) => S = 2 π (∝) tan((∝))

S = 2 π.0.0796.tan(13) = 0.0735 m

Number of turns N: typically 3-20 (using N=10, for high gain purposes)

Radius = 7.96 cm

Pitch (S) = 7.35cm

Height h = N.S = 10 * 7.35 = 73.5 cm

Gain = 11.8 + 10 log(C * N/λ) = 13.3 dBi

2.3. Yagi-Uda Antenna

It is a neccesity to first calculate the wavelength as c/f= 0.333m

Element	Length (L)	Typical value (% of λ)
Driven element	≈ 0.47 λ	≈ 15.7 cm
Reflector	≈ 0.55λ	≈ 18.3 cm
Director	≈ 0.45λ	≈ 15cm

Spacing (between elements):

Pair	Spacing (D)	Typical Range

Reflector-Drive	$\approx 0.2\lambda$	$\approx 6.7\ cm$
Driven-Director	$\approx 0.1 - 0.2\lambda$	$\approx 3.3 - 6.7\ cm$
Director-Director	$\approx 0.3\ \lambda$	$\approx 10cm$

For the Antenna Arrays $Dmax \approx N \cdot Nsin(\pi d/\lambda)/sin(\pi dN/\lambda)$

Where N is number of elements, d is the spacing beteween elements and D is the Directivity of the array.

2.5 Dolph–Tschebyscheff Linear Arrays

$$AF(\theta) = \sum_{n=0}^{N-1} w_n\, e^{jkdn\ cos\theta}$$

Where k = $2\pi/\lambda$ is the free-space wavenumber (λ =1), and is the elevation angle from end fire. We then normalize and convert to decibels via

$$AF(\theta) = \ 20\ \log_{10}(\frac{|AF(\theta)|}{max_{\theta'}^{|AF(\theta')}})$$

For the second
The array factor AF(θ) for a uniform linear array of N elements, spaced by a distance d, with all weights $w_n = 1$, is:

$$AF(\theta) = \sum_{n=0}^{N-1} e^{jkdncos\theta}$$

To center the array (symmetrical about broadside), we define:

$$n = [-\frac{N-1}{2}, -\frac{N-3}{2}, \dots, \frac{N-1}{2}\ ]$$

Thus, the array factor becomes:

$$AF(\theta) = \sum_{n=\ -(N-1)/2}^{(N-1)/2} e^{jkdn\ cos\theta}$$

where:

k= $2\pi/\lambda$ is the wave number (with =1 for simplicity)

d is inter-element spacing in wavelengths (typically d=0.5)

To analyze and compare results, the magnitude is normalized and converted to decibels

$$AF_{dB}(\theta) = 20.\log_{10}(\frac{|AF(\theta)|}{\max(|AF(\theta)|)})$$

2.B. LIBRARY CHOICES

NumPy:- For numerical operations, arrays, and mathematical computations.

SciPy(Especially scipy.special) – For special functions like Bessel and Chebyshev polynomials.

Plotting tool:- Matplotlib Including `mpl_toolkits.mplot3d` – For 2D and 3D radiataion

4NEC2 antenna modeler and optimizer: For simulating electromagnetic radiation parmaters

scikit-rf (skrf) – For analyzing network parameters like S-parameters (used less frequently unless S-parameter analysis is explicitly included).

2.C. SIMULATION SETUPS

2.4.1 Dipole Impedance & Directivity vs. Length

For the Input Impedance

The simulation was performed using the following steps:

1. Define Length Range**:**
   Create a linear array of dipole lengths from 0.1λ to 2.5λ with `numpy.linspace`.
2. Compute Input Resistance and Reactance**:**
   apply empirical formulas to compute Rin(L) and Xin(L) for each length value. The resistance was modeled by a squared sinc-like function, and the reactance by a logarithmic function at resonance.
3. Plot the Results**:**
   Use Matplotlib to plot two curves—resistance and reactance— against normalized length. Add labels, legends, and gridlines to see where reactance cuts zero (resonant points) and how resistance is behaving.
4. Analyze Resonances:
   Track lengths where the reactance crosses the x-axis (i.e., Xin=0) to determine resonant conditions. These are significant for optimum antenna operation and impedance matching.

For the Maximum Directivity

The simulation proceeds in these steps:

1. Define Length and Angle Grids:- Generate dipole lengths in the range 0.1λ to 2.5λ. Define a thin angular grid θ ∈ [0,π] (e.g., 1000 points) to sample the elevation angle.
2. Calculate Radiation Pattern:- For each dipole length L, compute the normalized electric field F(θ) using the analytical far-field pattern formula, and then compute $|F(\theta)|^2$ to obtain the radiation intensity.
3. Compute maximum directivity:
    - Find the maximum value of $|F(\theta)|^2$ (i.e., Umax).
    - Integrate $|F(\theta)|^2 \cdot \sin\theta$ over θ ∈ [0,π] using the trapezoidal rule (`np.trapz`) to get total radiated power.
    - Apply the directivity formula D = $\frac{2.U_{max}}{\int |F(\theta)|^2 \sin\theta \, d\theta}$
4. Convert and Plot Results:- Convert each directivity to dBi by $10\log_{10}(D)$, and plot vs. its corresponding dipole length. The plot shows where the directivity is greatest (as a rule of thumb from 1.25λ through 2.5λ), illustrating how larger length yields larger directivity but more complex radiation lobes.

2.4.2. Circularly Polarized Helix Antennas

1. Import numpy. Define Main function, def generate_helix_nec(filename, freq_mhz, mode='short')
2. Define Basic parameters, speed of light and computation of wavelength
3. Set Helix Parmeters based on mode, either as short helix or long helix
4. Compute 3D geometry
5. Calculate coordinated of the helix
6. Writing the NEC file:- setting up header, Defining Geometry with GW cards and Ground plane, Excitaiton, frequency.
7. Call the function for both modes
8. After opening the file using the 4Nec2 system, click on calculate > NEC Output. Aftersimulation ends go to the far field plot or Gain/Pattern to analyze results.
9. To check detailed info click on windows smith chart or 3d viewer
10. In the 3d viewer tweak the toogle down button until you get what is desired

2.4.3. Antenna Arrays

1. Import required libraries
2. Define the array_factor functions
3. Define the compute_max_directivity function
4. Define Ranges for spacing and number of elements
5. Plot Directivity vs Spacing for Each N

6. Customize and display the plot

2.4.5 Dolph–Tschebyscheff Linear Arrays

For the Array factor of each taper

1. Define Array Geometry:- Define number of elements N, inter-element spacing d, and generate a dense vector of scan angles θ from 0 through π radians (0 - 180).
2. Center Element Indices:- Create the index vector n= [0,1,..., N-1] - (N-1) / 2 so that the array factor is reference-symmetric about broadside.
3. Create Tapers:- for each specified sidelobe level (20 dB,30 dB, 40 dB), call chebwin (N, at=sll, sym=true) to get the weight vector
4. Compute phases:- Build the matrix of complex exponentials exp(jkdn cosθ)
5. Create Array Factor:- Multiply every column(element) of the phase matrix by its corresponding weight, sum over all N elements to get the AF at every θ, then normalize by dividing by the maximum magnitude.
6. Convert to dB:- take 20 log10 of the normalized magnitude (adding a tiny epsilon inside the log to avoid log 0).
7. Plot:- plot the three AF curves on the same Matplotlib figure and set axis labels/limits.

For the side-lobe levels and beamwidths against uniform taper

7. Define parmeters: Identify number of elements N, inter-element spacing d, and form a vector of angles θ from 0 to π radians (i.e., 0° to 180° broadside scan).

Element Indexing: Form a symmetric index vector n=np.arange (N) − (N−1) / 2 to bring the array to its center for even beam shaping.

8. Uniform Weights: Identify all weights w=1, no taper—same power to all elements.
9. Calculate Array Factor: Use the complex exponential exp(jkdncosθ), weight each by 1, sum over all elements for eachθ, and compute the magnitude.
10. Normalize and Convert to dB: Divide the array factor by its peak value and compute 20log 10 to express in decibels.
11. Plot the Pattern: Plot the normalized array factor in dB from 0°–180° scan on Matplotlib.
12. Extract Performance Metrics: Quantify the –3 dB beamwidth by determining angular points in the main lobe where the AF drops to –3 dB. Estimate the sidelobe level by determining the maximum value in side regions beyond the main lobe.

3. RESULTS

PLOTS

3.1 Dipole Impedance & Directivity vs. Length

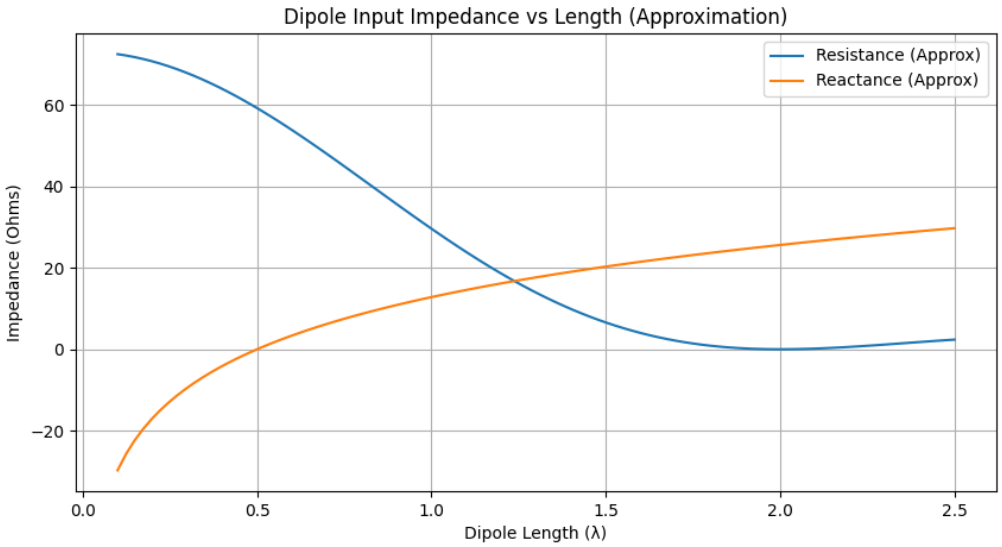


FIG 1 Input Impedance vs Dipole length

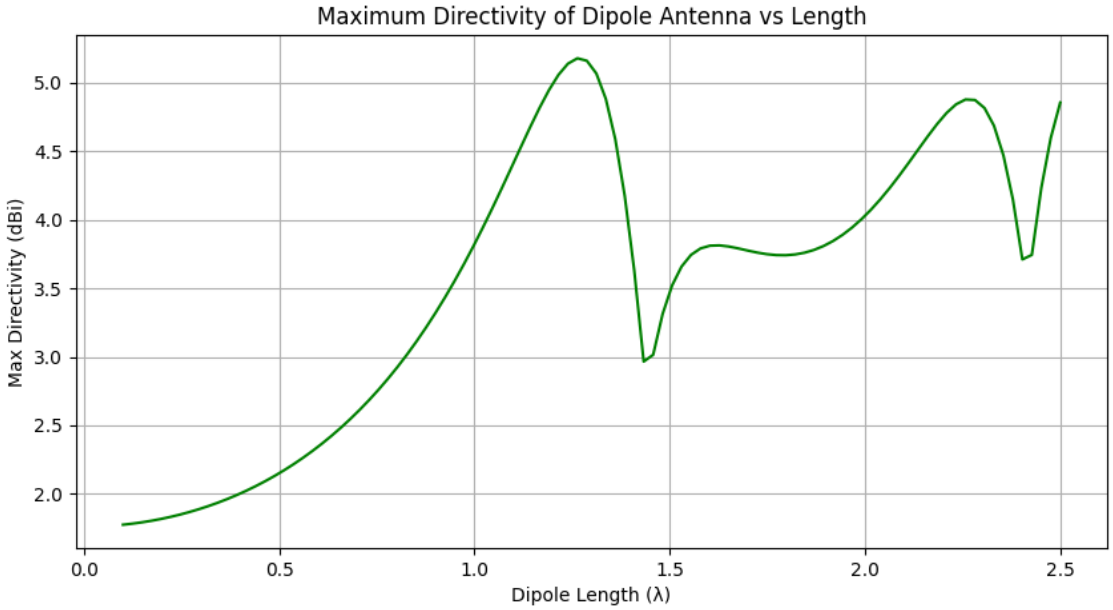
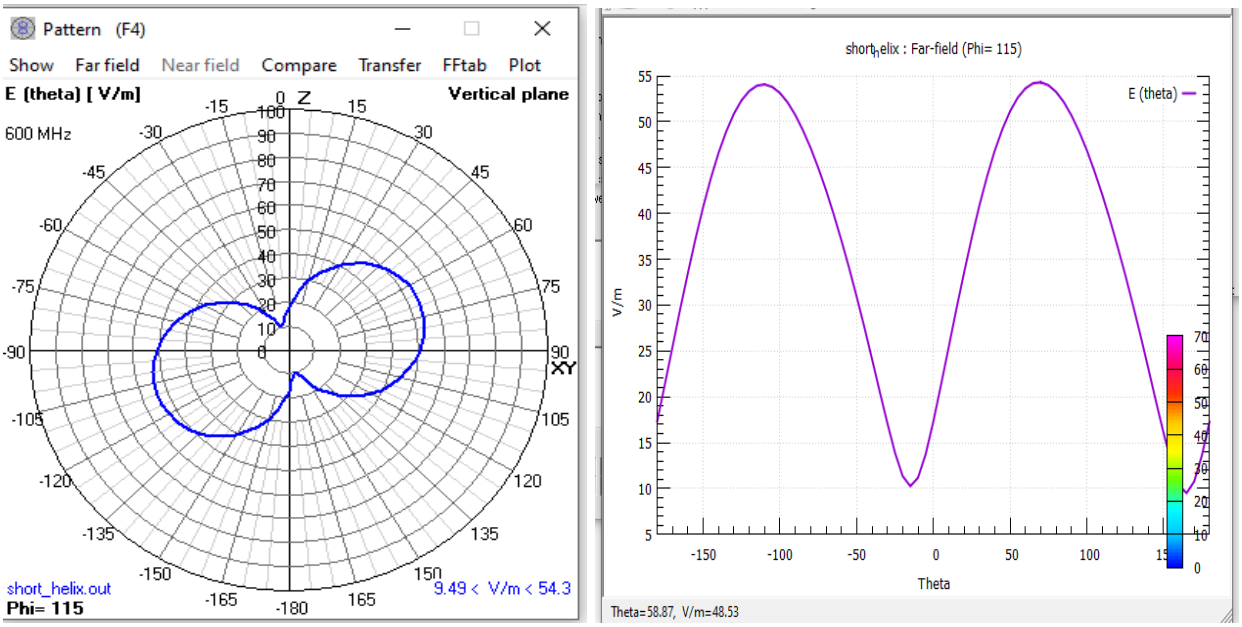


Fig 2. Maximum Directivity of Dipole antenna vs length

The most important resonant lengths for a center-fed thin dipole happen at about 0.48λ, 1.5λ, and 2.5λ, where the input reactance goes through zero, or is at resonance; at these locations, the input resistance also maximizes or plateaus (for example, ~73 Ω at 0.48λ). These correspond to standing-wave current maxima, and hence make them good radiators. Optimum directivity occurs near 1.25λ–2.5λ, which raises from ~2.15 dBi at 0.5λ (standard dipole) to ~5.5 – 6 dBi for the longer lengths. Higher directivity comes at the expense of having multiple radiation lobes, such that the pattern is more complex and less ideal for some applications. Both impedance and directivity exhibit oscillatory tendencies with increasing length, as the distribution of current varies and the resulting radiated properties.

3.2. Circularly Polarized Helix Antennas @ 600 MHz

Short Helix



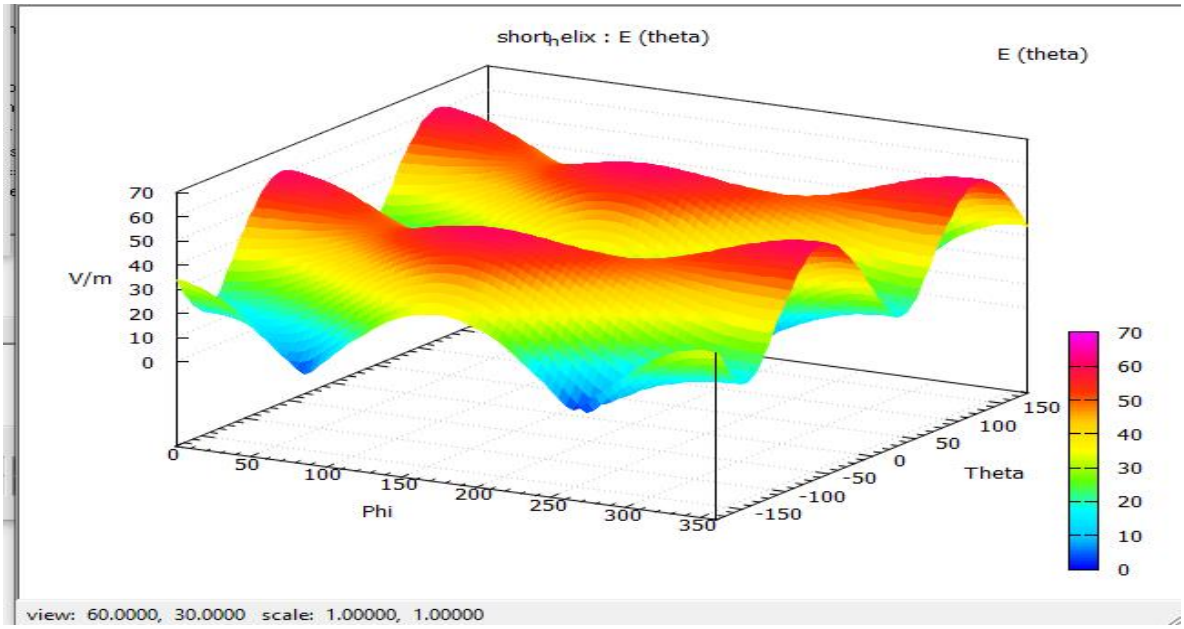
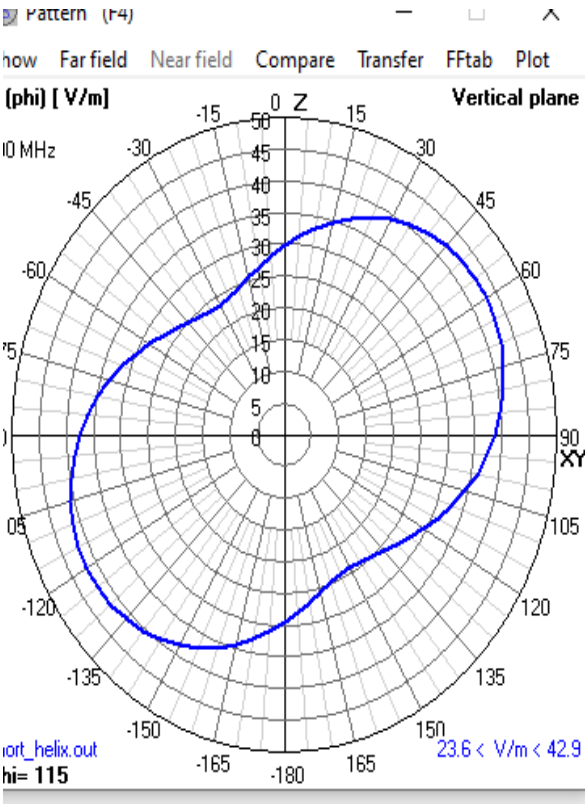
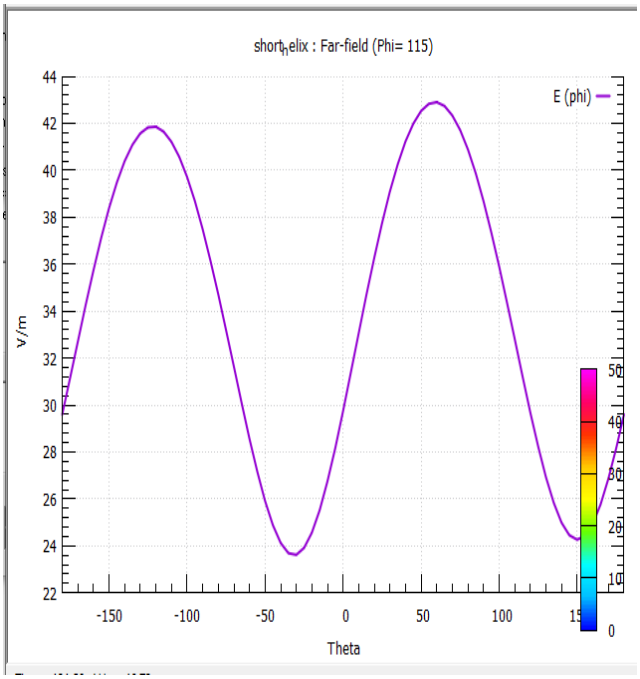


Fig 3. Polar plot,2d and 3D plot of the Radiation Intensity in Theta





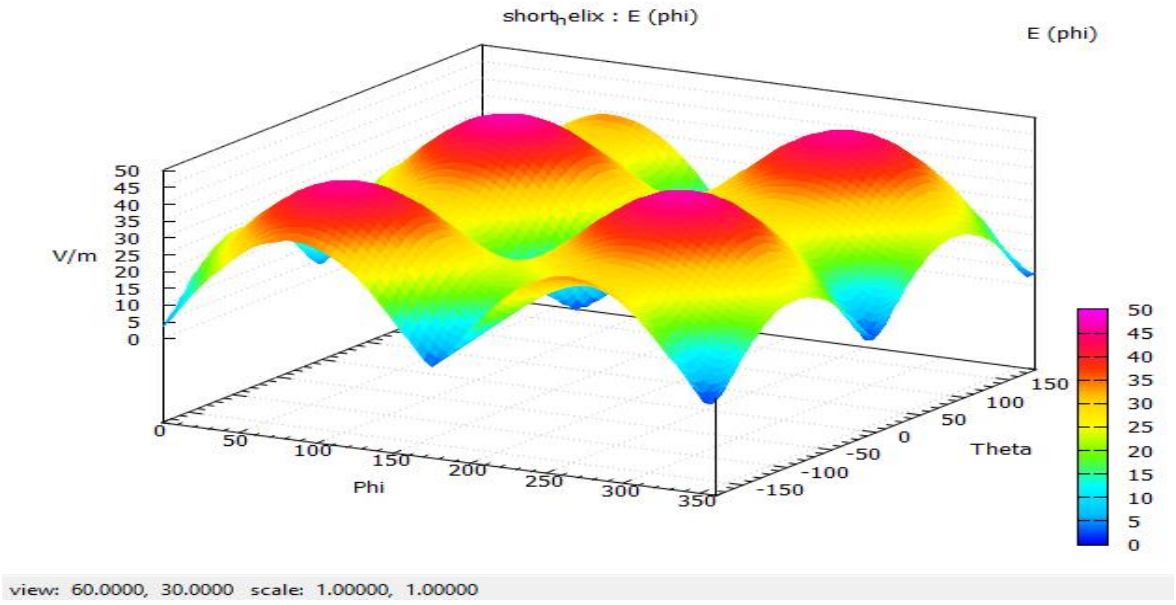
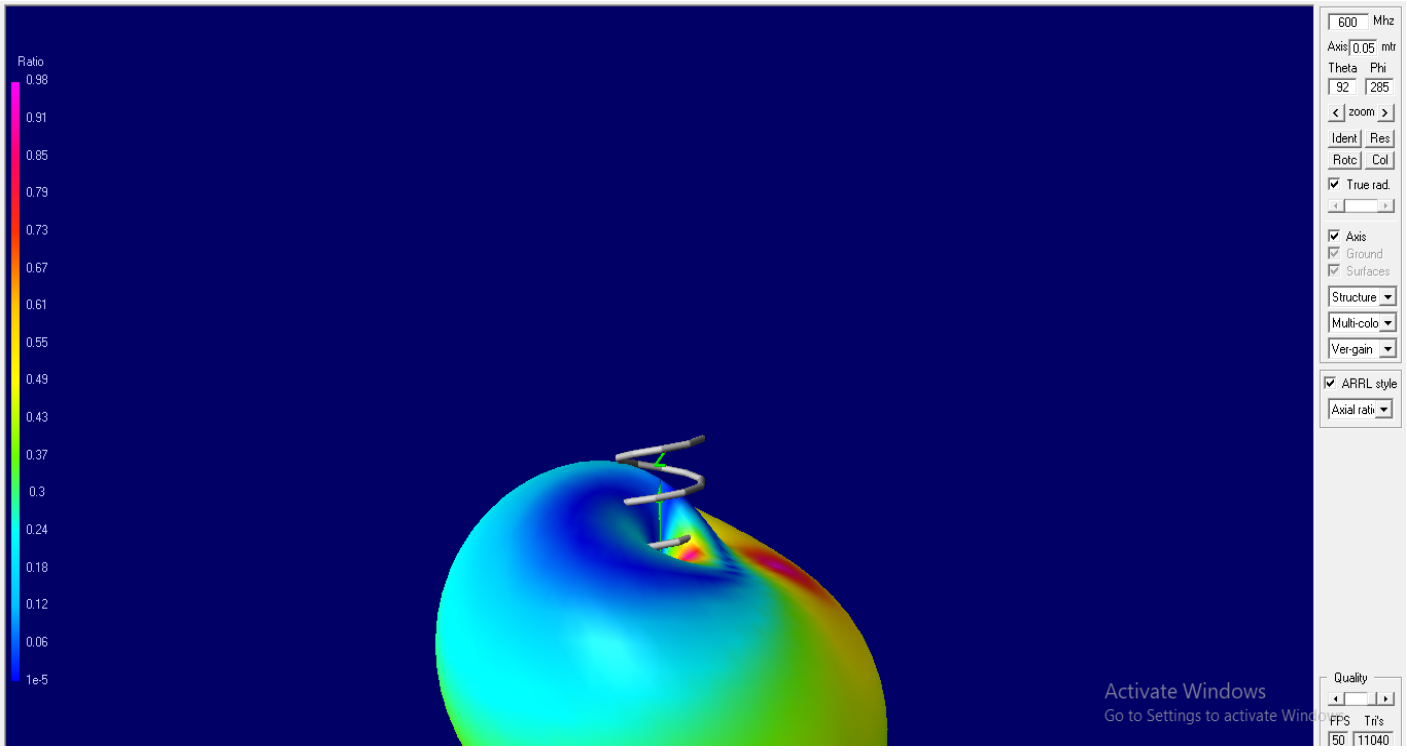


Fig 4. Polar plot,2d and 3D plot of the Radiation Intensity in phi



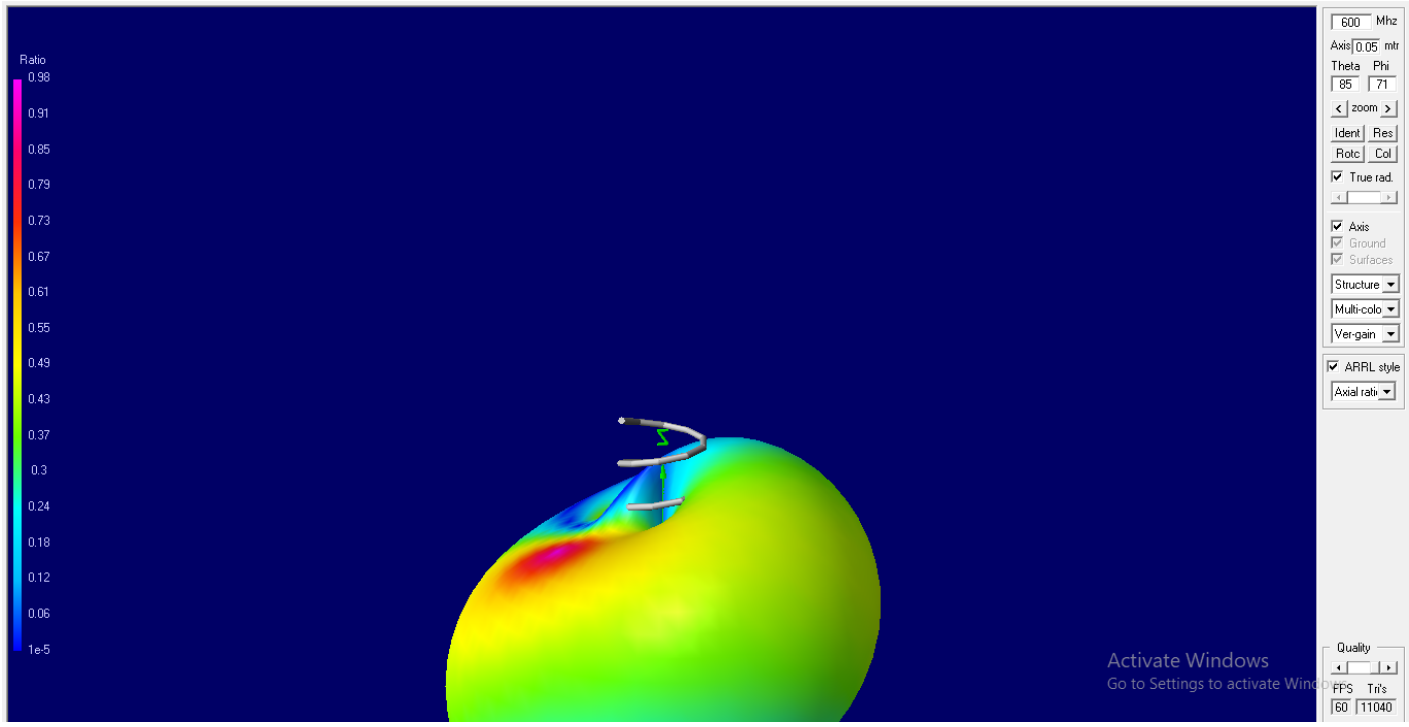


Fig 5. Axial Ratio

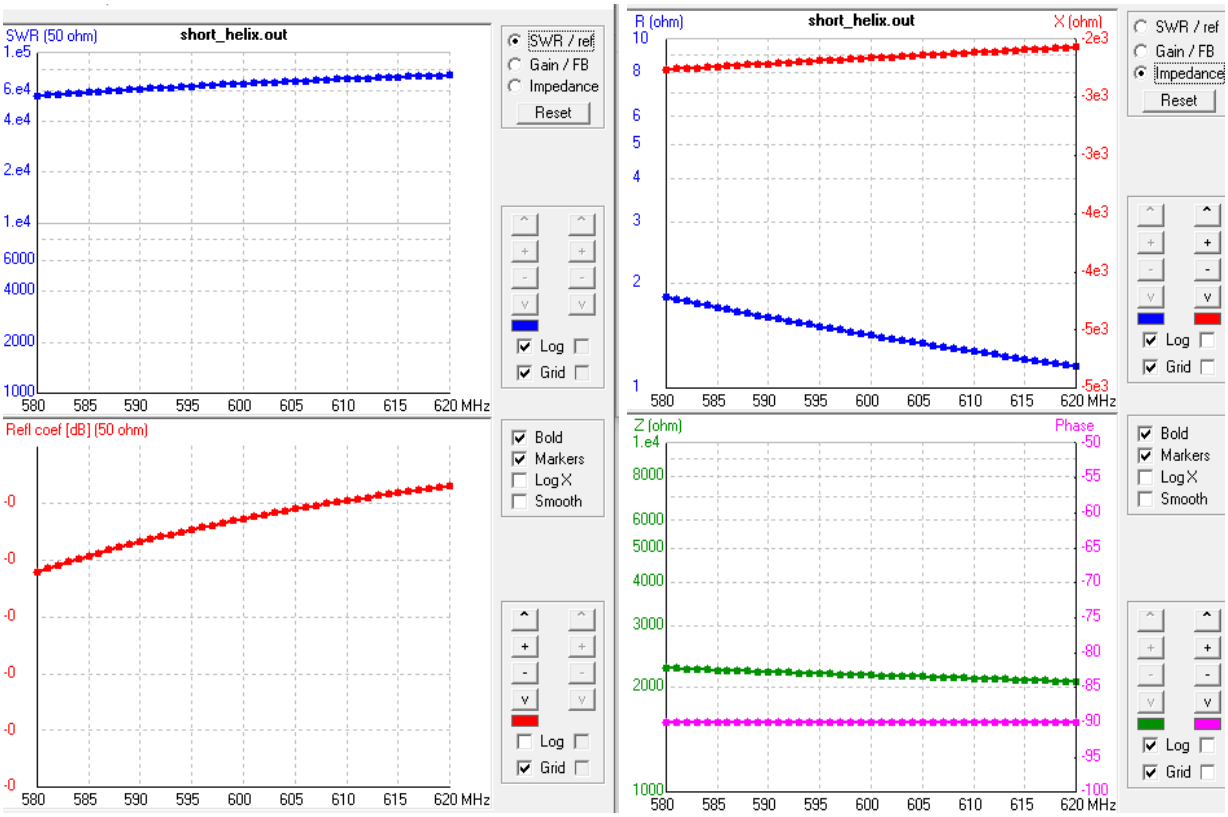


Fig 6. Input Bandwidth

Long Helix

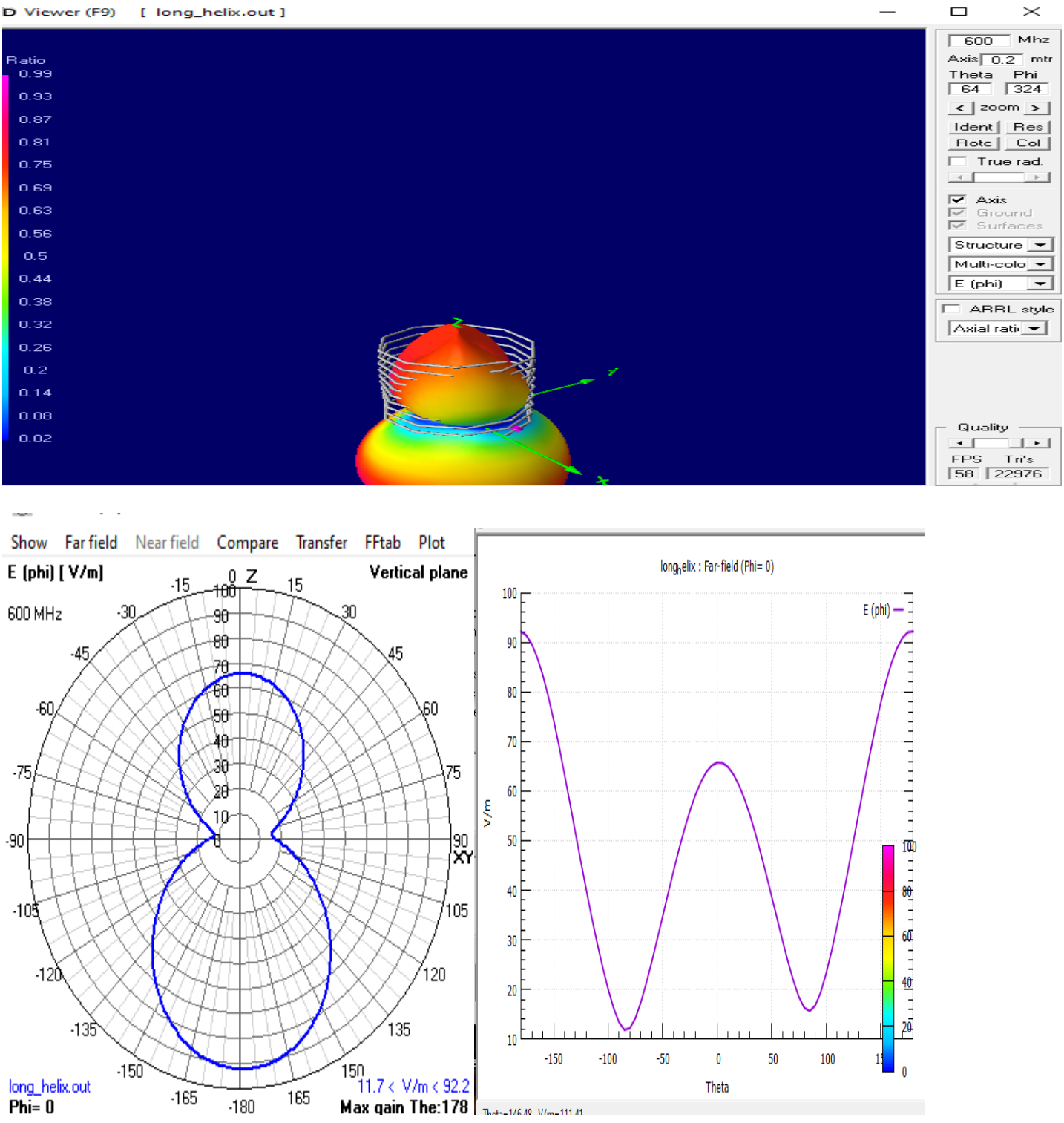


Fig 7.

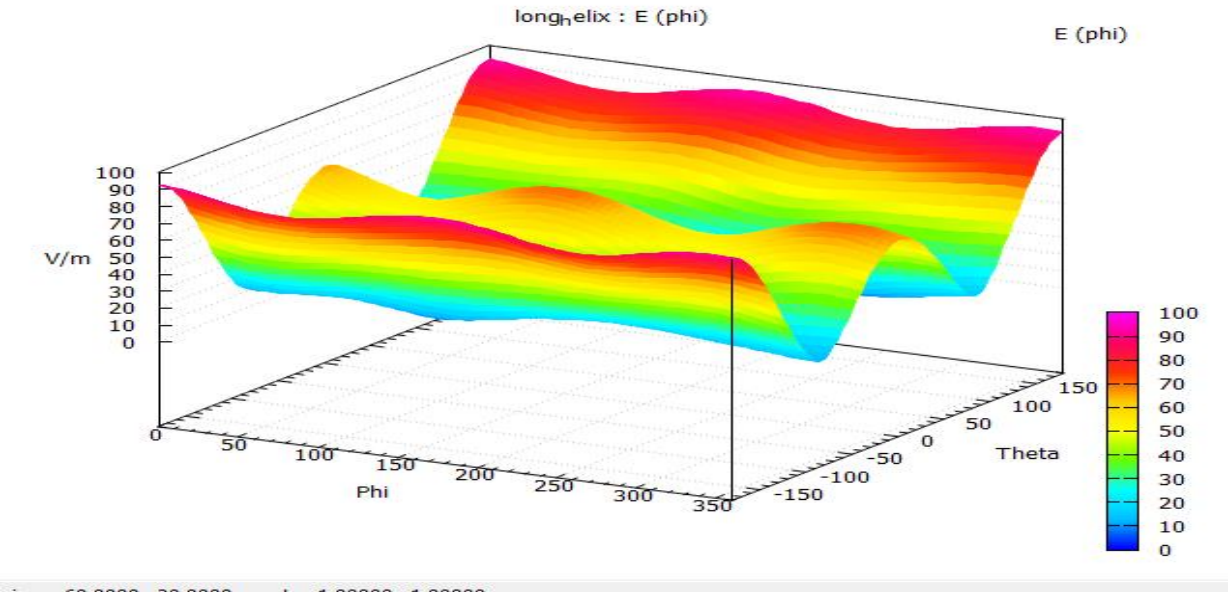


Fig. 7 Axial, 2d, 3d and polar plots of long helix

The short helical antenna and the long helical antenna differ in characteristics because of geometry. The short helix, having shorter turns and reduced size (much smaller radius and spacing compared to a wavelength), typically operates in the normal mode, radiating primarily omnidirectional patterns with low gain and is well suited for low-frequency, small space applications. On the other hand, the long helix operates in the axial mode, where the size of the helix is close to or greater than a wavelength, and this produces a highly directional radiation pattern along the helix axis and much higher gain, and is hence adequate for satellite and long-haul communications. The structural disparity leads to largely different performance, in which optimized space saving and wider coverage are achieved by the short helix, and optimized direction efficiency and gain are achieved by the long helix.

3.3 ANTENNA ARRAYS

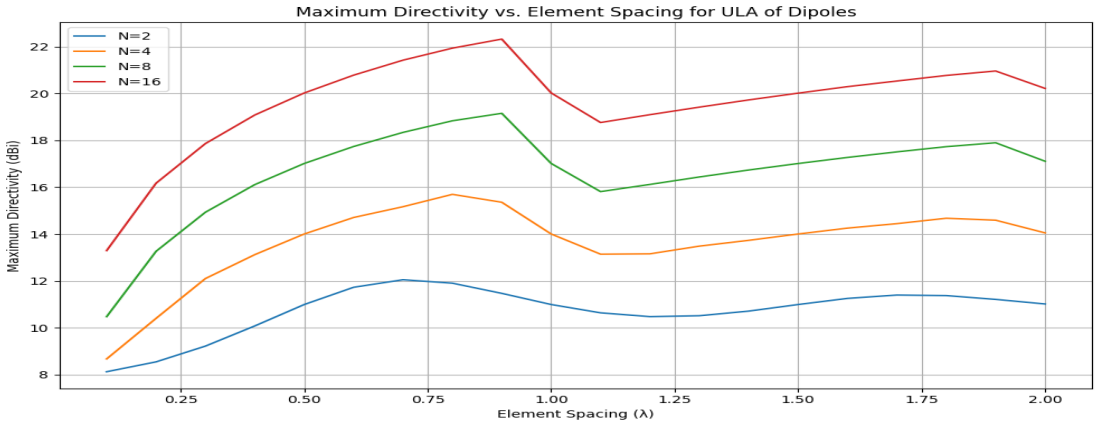


Fig 8. Max Directivity vs Element Spacing

The graph illustrates the relationship between maximum directivity (in dB) vs. element spacing (in wavelengths, λ) for an array of uniform linear dipoles having varying numbers of elements (N = 2, 4, 8, 16). The larger the number of elements, the greater is the maximum directivity achievable at all spacings between the elements, which implies that larger arrays provide more directional beam patterns. The directivity increases with spacing up to a certain value (around 0.8λ to 0.9λ), after which it either levels off or experiences mild oscillations. Conveniently, for N = 16, directivity increases sharply at around 0.9λ but decreases slightly at 1.0λ before continuing to rise steadily again. This suggests that directivity is enhanced with increased spacing to a certain extent, but extremely large spacing will severely degrade performance most likely due to grating lobes or loss of constructive interference.

3.4 YAGI-UDA

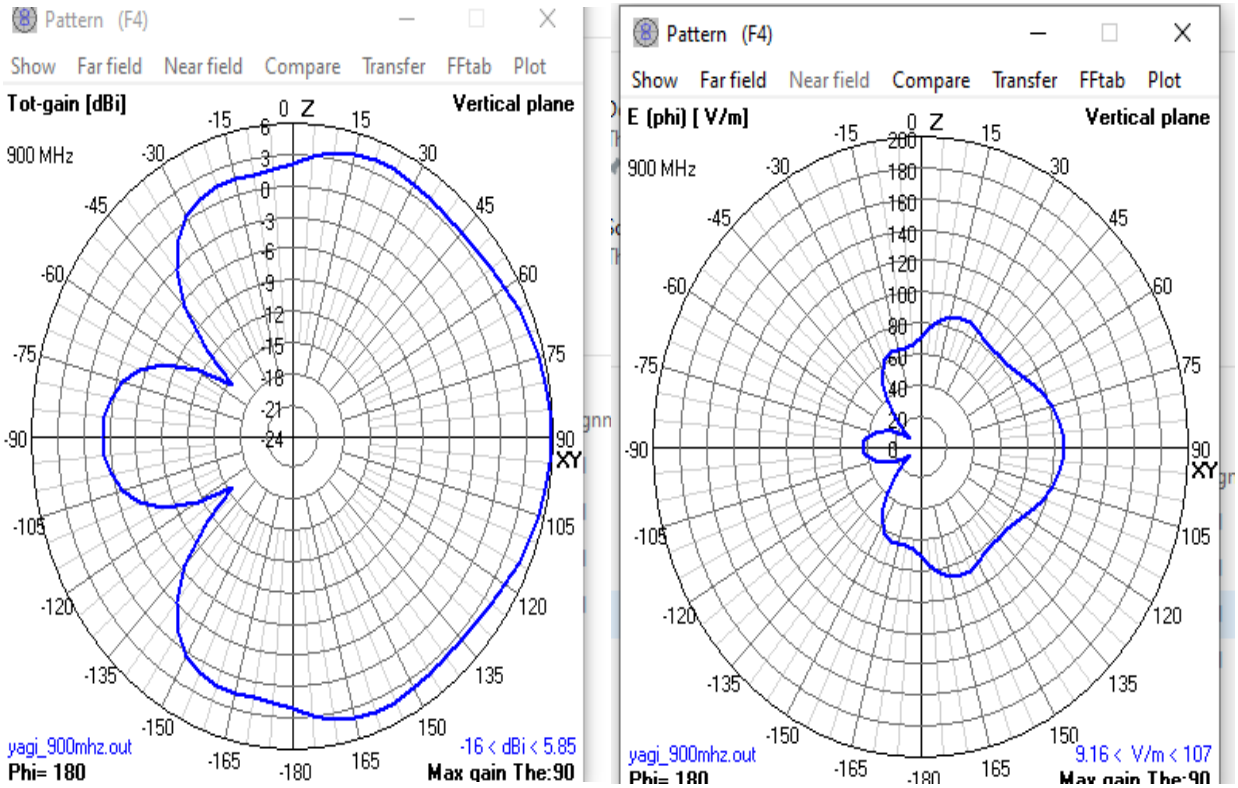


Fig 9. Polar plot of the total gain and Radiation Intensity

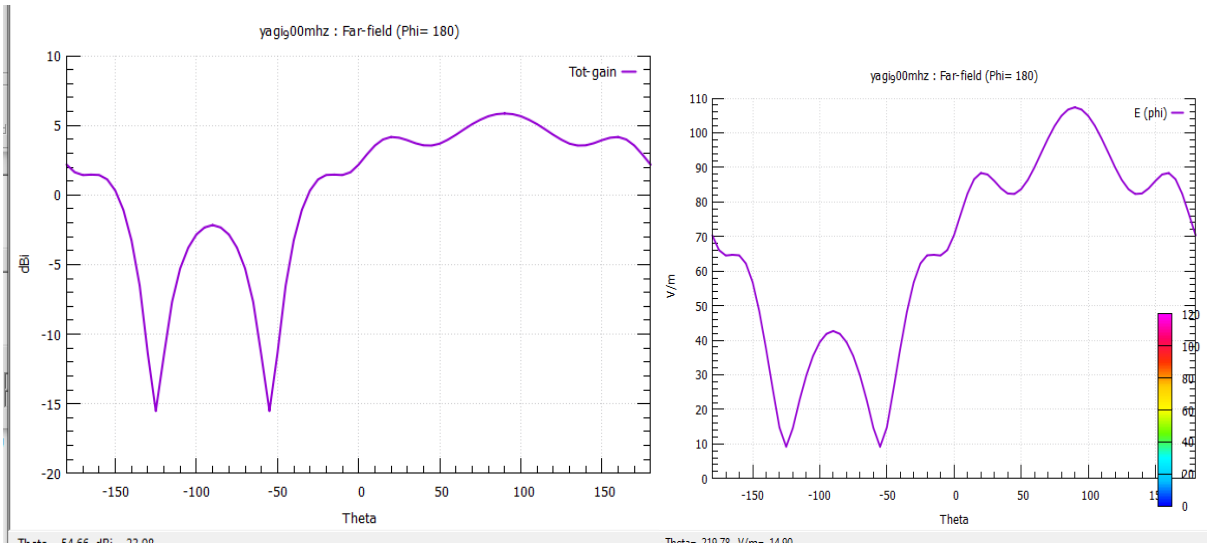


Fig 10. 2d plots of the total gain and Radiation Intensity

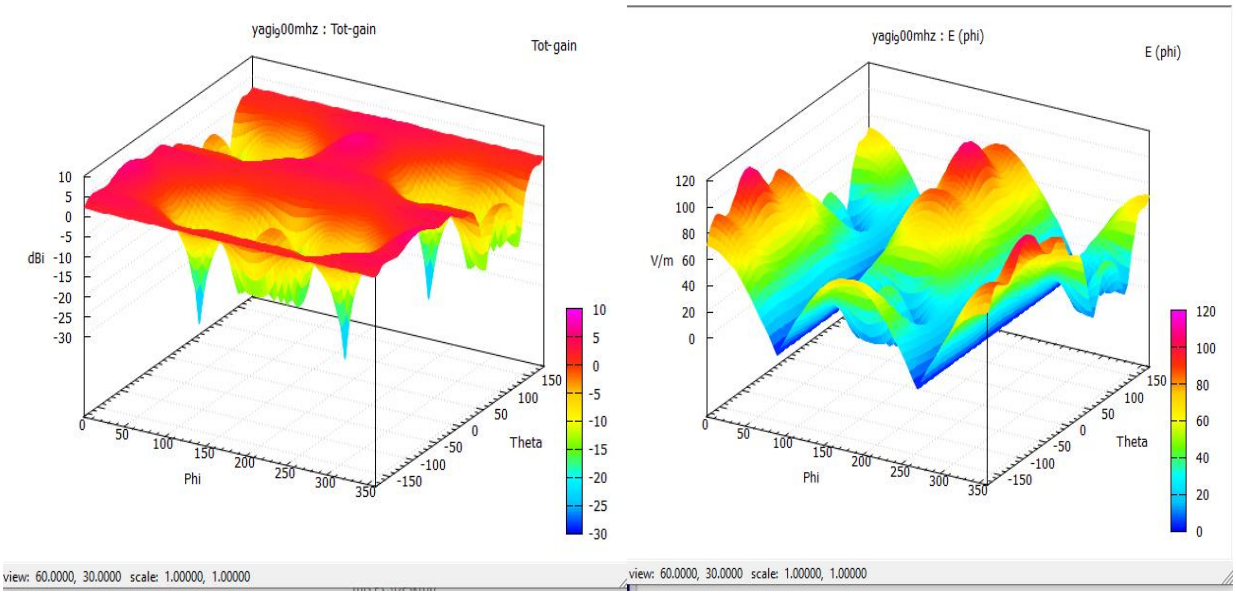


Fig 11. 3D plot of the total gain and the radiation intensity

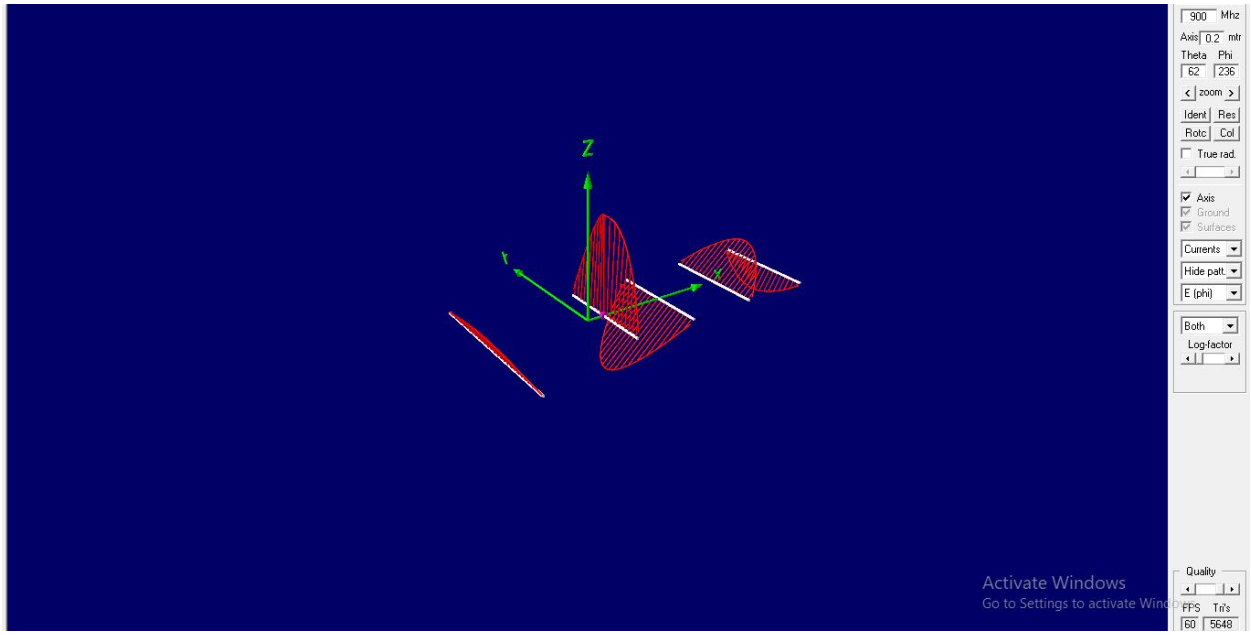
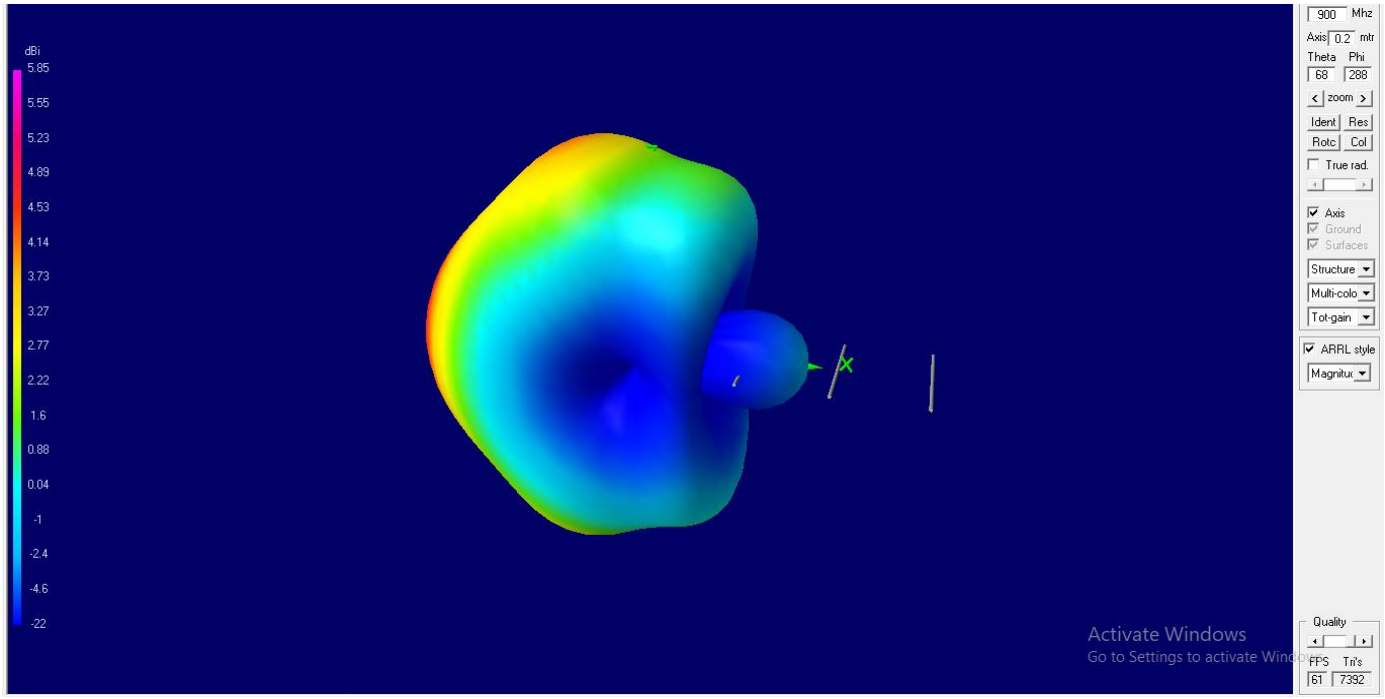


Fig 12. Polarization



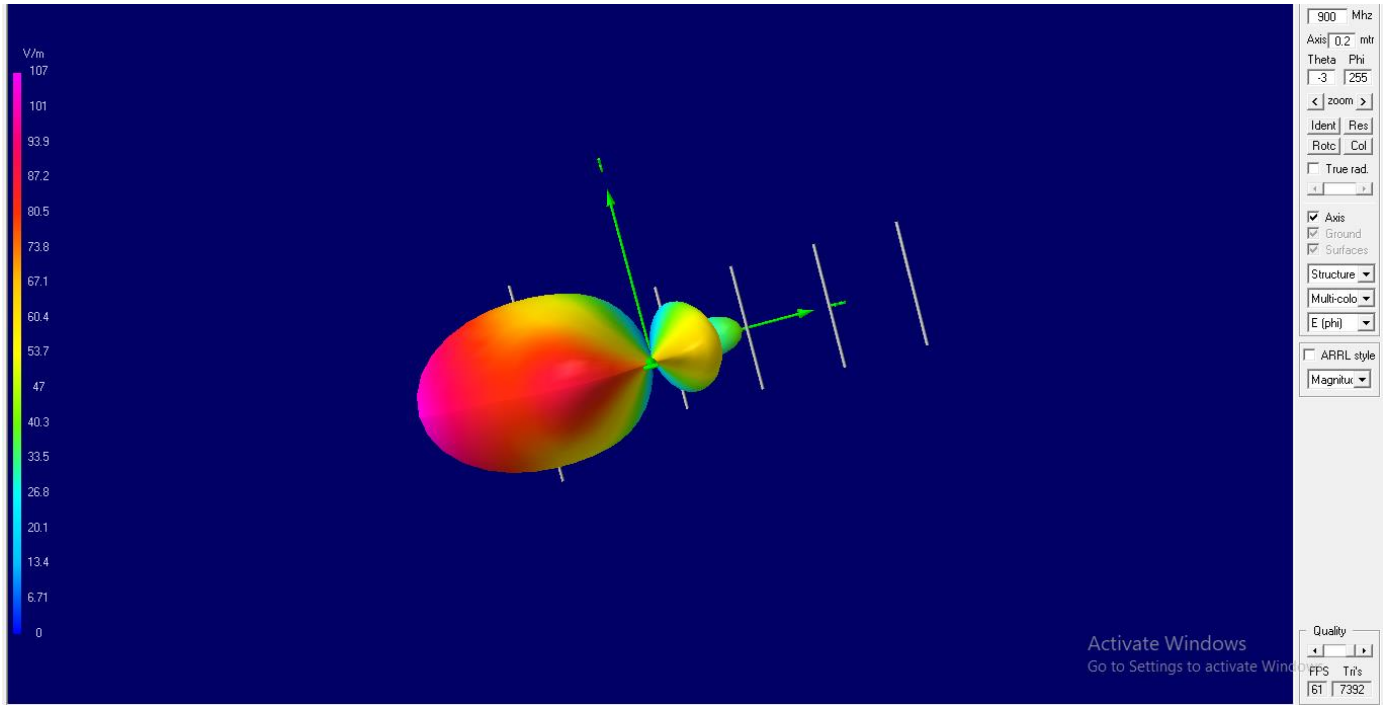


Fig 13. Actual Radiation intensity and total gain directions

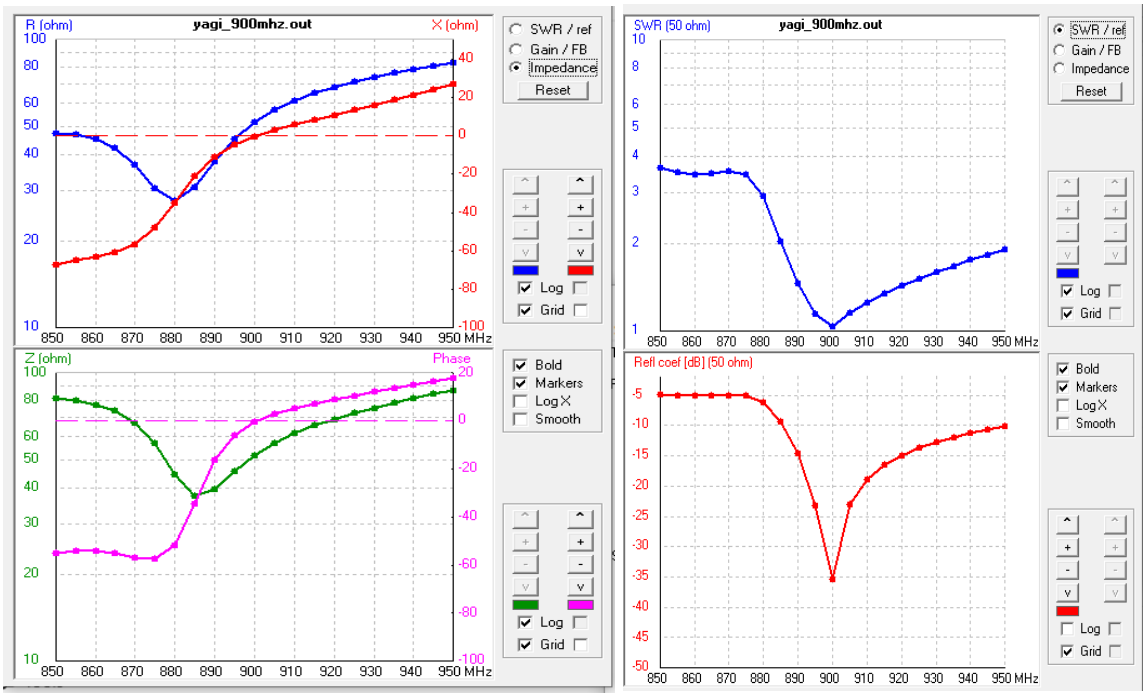


Fig 14. Input Impedance

The tuning process of the Yagi-Uda antenna made a drastic improvement in its performance in terms of various key parameters. The original antenna had a very bad impedance match of $2.74 + j1.46\,\Omega$ and an excessively high SWR of over 20, showing heavy reflection and inefficiency of the signal. After tuning, the impedance became almost ideal $51.07 - j0.3\,\Omega$ with just an SWR of 1.03, there was very good power transfer with no reflection. This optimization also enhanced the radiation characteristics: the 2D and 3D patterns exhibited a clear, forward-oriented major lobe with reduced side lobes, which corresponded to enhanced gain and directivity. Polarization enhancements were observed from the balanced field distributions, which confirmed consistent linear polarization and high radiation efficiency. The Smith Chart showed that the impedance point shifted close to the center upon tuning, confirming enhanced return loss and matching. Besides, the antenna had broader usable bandwidth with low SWR and field uniformity over a greater frequency range. In summary, tuning transformed an initially inefficient design into a stable, high-gain antenna for secure 900 MHz communication applications with significantly improved impedance matching, gain, bandwidth, and field uniformity.

3.5 Dolph–Tschebyscheff Linear Arrays

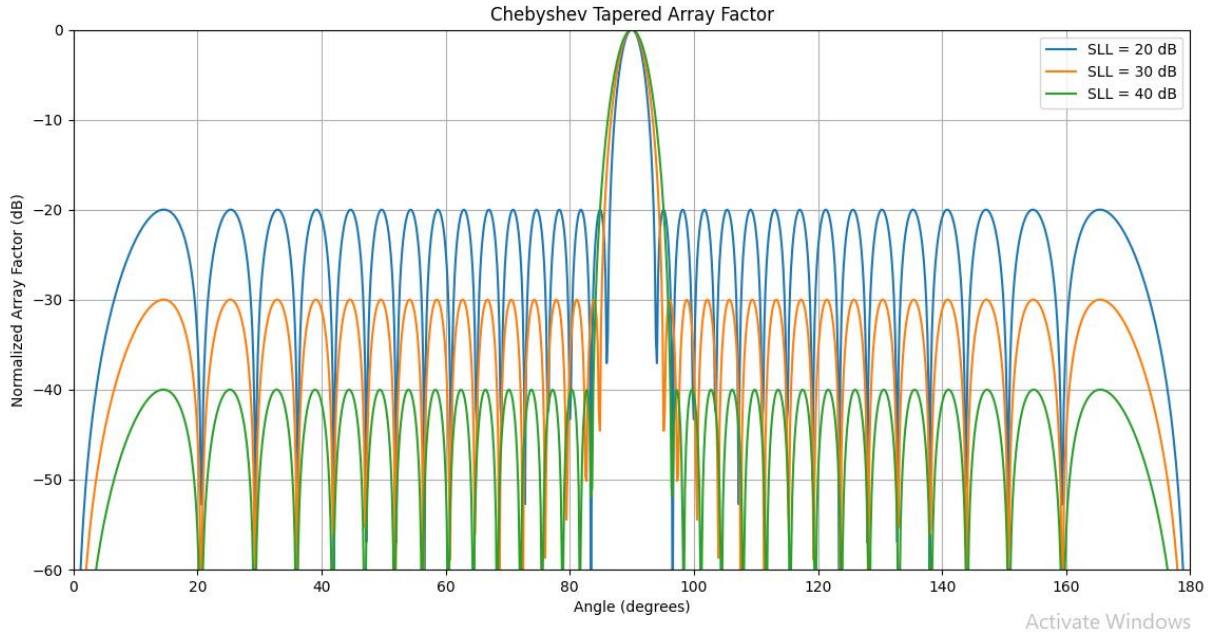


Fig 15. Normalized array factor of each taper

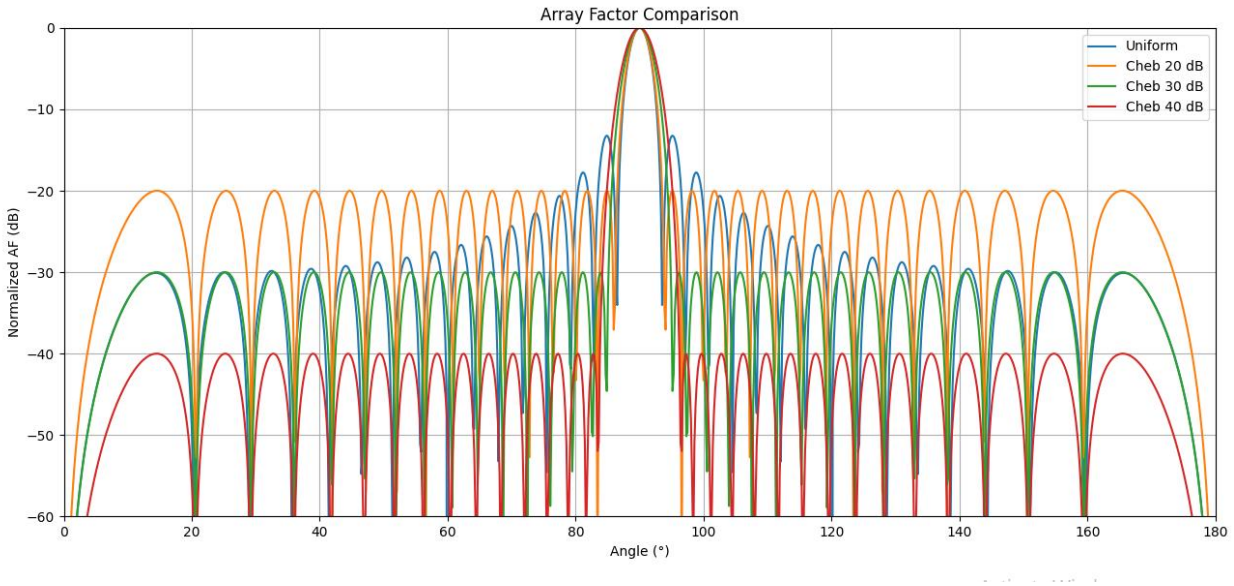


Fig 16. Normalized Array factors against uniform taper

Through the simulation, we have seen how different Chebyshev tapers with different side-lobe levels (20 dB, 30 dB, and 40 dB) influence the array factor of a 10-element uniform linearly spaced antenna array. Using Python libraries like NumPy, SciPy, and Matplotlib, we have computed and graphed the normalized array factor to observe the change in beamwidth and the side-lobe levels. The results demonstrate side-lobe suppression can be enhanced with Chebyshev tapering to suppress unwanted side-lobes effectively but widen the main beam, thereby highlighting the inherent trade-off between directivity control and interference in array design.

4. CONCLUSION AND DISCUSSION

This project provided a practical tour of simple and advanced concepts in antenna theory using Python as an analysis and simulation tool. Through simulations of various wire antenna geometries, such as dipoles, helical antennas, and Yagi–Uda structures, and antenna arrays like uniform linear and Dolph–Tschebyscheff arrays, we gained experience in radiation patterns, impedance behavior, and directivity patterns.

Each assignment focused on antenna's analytical modeling and visualization of its performance across different parameters. The use of NumPy and SciPy facilitated numerical computations with ease, especially in handling mathematical equations and designing Chebyshev tapers. Matplotlib enabled the creation of clearly defined, descriptive 2D and 3D plots of radiation patterns, impedance plots, and array factors. In addition, NEC-Python or pynec also provided an optional but highly capable infrastructure for electromagnetic behavior simulation under more realistic circumstances, verifying theoretical findings with near-physical accuracy. For S-parameter management operations, scikit-rf was referred to as a useful tool.

Overall, this project was broadening our understanding of practice and theory in antenna design. It demonstrated the versatility of Python and open-source libraries in RF engineering and solidified fundamental concepts that are the foundation of wireless communication systems.

5. APPENDIX

5.1. Wire antennas

Input Impedance

```
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

```
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()
```

5.2. Circularly Polarized Helix Antennas

```
import numpy as np
def generate_helix_nec(filename, freq_mhz, mode='short'):
    c = 3e8
    wavelength = c / (freq_mhz * 1e6)
    if mode == 'short':
        a = wavelength / 40
        S = a
        N = 5
    elif mode == 'long':
        a = wavelength / (2 * np.pi)
        pitch_angle_deg = 13
        pitch_angle_rad = np.radians(pitch_angle_deg)
        S = 2 * np.pi * a * np.tan(pitch_angle_rad) / (2 * np.pi)
        N = 10
    else:
        raise ValueError("mode must be 'short' or 'long'")
    wire_radius = 0.001  # 1 mm wire
    segments_per_turn = 10
    total_segments = N * segments_per_turn
    L = S * N
```

```
    theta = np.linspace(0, 2 * np.pi * N, total_segments)

    x = a * np.cos(theta)

    y = a * np.sin(theta)

    z = np.linspace(0, L, total_segments)

    with open(filename, 'w') as f:

        f.write(f"CM {mode.title()} Helix Antenna @ {freq_mhz} MHz\n")

        f.write("CE\n")

        tag = 1

        for i in range(total_segments - 1):

            f.write(f"GW  {tag}  1  {x[i]:.5f}  {y[i]:.5f}  {z[i]:.5f}  {x[i+1]:.5f}
{y[i+1]:.5f} {z[i+1]:.5f} {wire_radius:.5f}\n")

            tag += 1

        f.write("GE 0\n")

        f.write(f"EX 0,1,{total_segments//2},0,1.,0.\n")

        f.write(f"FR 0,1,0,0,{freq_mhz:.2f},0\n")

        f.write("RP 0,181,1,1000,0.,0.,1.,1.\n")

        f.write("EN\n")

# Generate .nec files

generate_helix_nec("short_helix.nec", freq_mhz=600, mode='short')

generate_helix_nec("long_helix.nec", freq_mhz=600, mode='long')
```

5.3 Yagi-Uda

```
import matplotlib.pyplot as plt

# Step 1: Define constants and compute wavelength

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

f = 900e6  # frequency in Hz (900 MHz)

wavelength = c / f  # in meters

print(f"Wavelength at 900 MHz: {wavelength:.3f} meters")

# Step 2: Calculate element lengths
```

```
reflector_length = 0.55 * wavelength

driven_length = 0.47 * wavelength

director_length = 0.45 * wavelength  # same length for all directors

print(f"Reflector Length: {reflector_length:.3f} m")

print(f"Driven Element Length: {driven_length:.3f} m")

print(f"Director Length: {director_length:.3f} m (same for all 3)")

# Step 3: Element positions along x-axis (start from reflector at x=0)

positions = {}

positions["Reflector"] = 0

positions["Driven"] = positions["Reflector"] + 0.2 * wavelength

positions["Director 1"] = positions["Driven"] + 0.15 * wavelength

positions["Director 2"] = positions["Director 1"] + 0.3 * wavelength

positions["Director 3"] = positions["Director 2"] + 0.3 * wavelength

# Display positions

for name, pos in positions.items():

    print(f"{name} position: {pos:.3f} m")

# Element lengths (in meters)

lengths = {

    "Reflector": reflector_length,

    "Driven": driven_length,

    "Director 1": director_length,

    "Director 2": director_length,

    "Director 3": director_length

}

# Plotting

fig, ax = plt.subplots()

for name, x_pos in positions.items():

    length = lengths[name]
```

```
    y_top = length / 2
    y_bottom = -length / 2
    ax.plot([x_pos, x_pos], [y_bottom, y_top], label=name, linewidth=3)
# Labels and styling
ax.set_title("Yagi–Uda Antenna Geometry @ 900 MHz (Top View)")
ax.set_xlabel("Position along x-axis (meters)")
ax.set_ylabel("Element length (meters)")
ax.legend()
ax.grid(True)
plt.axis('equal')
plt.show()
# Save Yagi-Uda antenna to NEC format
def generate_nec_file(filename, wavelength, positions, lengths, num_segments=11):
    with open(filename, 'w') as f:
        f.write("CM Yagi-Uda Antenna @ 900 MHz\n")
        f.write("CE\n")
        wire_id = 1
        radius = 0.001  # 1 mm wire radius
        for name, x in positions.items():
            length = lengths[name]
            y1 = -length / 2
            y2 = length / 2
            f.write(f"GW {wire_id} {num_segments} {x:.3f} {y1:.3f} 0 {x:.3f} {y2:.3f}
0 {radius:.4f}\n")
            wire_id += 1
        # Excitation (center segment of driven element, tag 2)
        f.write(f"EX 0 2 {num_segments//2+1} 0 1 0\n")
        # Frequency (1 freq point at 900 MHz)
```

```
        f.write("FR 0 1 0 0 900 0\n")
        # Radiation pattern (3D sweep)
        f.write("RP 0 181 1 1000 0 0 1 0\n")
        # End
        f.write("EN\n")
    print(f"NEC file '{filename}' generated.")
# Call the function
generate_nec_file("yagi_900mhz.nec", wavelength, positions, lengths)
```

5.4. Antenna Arrays

```
import numpy as np
import matplotlib.pyplot as plt
def array_factor(theta, N, d, wavelength=1.0):
    k = 2 * np.pi / wavelength
    psi = k * d * np.cos(theta)
    numerator = np.sin(N * psi / 2)
    denominator = np.sin(psi / 2) + 1e-9  # Avoid division by zero
    AF = numerator / denominator
    return np.abs(AF)
def compute_max_directivity(N, d, num_points=1000):
    theta = np.linspace(0, np.pi, num_points)
    AF = array_factor(theta, N, d)
    U = AF ** 2
    Prad = np.trapz(U * np.sin(theta), theta)
    Umax = np.max(U)
    D = 4 * np.pi * Umax / Prad
    D_dBi = 10 * np.log10(D)
    return D_dBi
```

```
spacings = np.arange(0.1, 2.05, 0.1)

element_counts = [2, 4, 8, 16]

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

for N in element_counts:

    directivities = [compute_max_directivity(N, d) for d in spacings]

    plt.plot(spacings, directivities, label=f'N={N}')

plt.xlabel('Element Spacing (λ)')

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

plt.title('Maximum Directivity vs. Element Spacing for ULA of Dipoles')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```

5.5. Dolph–Tschebyscheff Linear Arrays

Array Factor Tapers

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.signal.windows import chebwin

# Parameters

N = 32

d = 0.5

theta = np.linspace(0, np.pi, 1000)

# Centered element indices

n = np.arange(N) - (N - 1)/2

def array_factor(weights, theta, d, n):

    k = 2 * np.pi

    # outer over cos(theta) and element positions
```

```
    phase = np.exp(1j * k * d * np.outer(np.cos(theta), n))
    af = np.sum(weights * phase, axis=1)
    af /= np.max(np.abs(af))
    return 20 * np.log10(np.abs(af) + 1e-12)
# Plot everything together
plt.figure(figsize=(8, 5))
for sll, color in zip([20, 30, 40], ["C0", "C1", "C2"]):
    w = chebwin(N, at=sll, sym=True)
    af_db = array_factor(w, theta, d, n)
    plt.plot(np.degrees(theta), af_db, label=f"SLL = {sll} dB", color=color)
plt.title("Chebyshev Tapered Array Factor")
plt.xlabel("Angle (degrees)")
plt.ylabel("Normalized Array Factor (dB)")
plt.ylim(-60, 0)
plt.xlim(0, 180)
plt.grid(True)
plt.legend()
plt.show()
```

Uniform Taper

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.signal.windows import chebwin
# 1) Array & angle setup
N = 32
d = 0.5             # element spacing (λ)
theta = np.linspace(0, np.pi, 1000)
```

```
angles_deg = np.degrees(theta)

# center the element indices for symmetry

n = np.arange(N) - (N - 1)/2

# 2) Define the four tapers

tapers = {

    "Uniform": np.ones(N),

    "Cheb 20 dB": chebwin(N, at=20, sym=True),

    "Cheb 30 dB": chebwin(N, at=30, sym=True),

    "Cheb 40 dB": chebwin(N, at=40, sym=True),

}

# 3) Array-factor function

def array_factor_db(weights):

    k = 2 * np.pi

    phase = np.exp(1j * k * d * np.outer(np.cos(theta), n))

    af = np.abs(np.sum(weights * phase, axis=1))

    af /= np.max(af)

    return 20 * np.log10(af + 1e-12)

# 4) Compute & collect metrics

results = []

af_curves = {}

for name, w in tapers.items():

    af_db = array_factor_db(w)

    af_curves[name] = af_db

    # find indices around the main lobe to get the −3 dB beamwidth

    peak_idx = np.argmax(af_db)

    # walk left and right until AF drops below −3 dB

    left = peak_idx

    while left > 0 and af_db[left] >= -3:
```

```
        left -= 1
    right = peak_idx
    while right < len(af_db)-1 and af_db[right] >= -3:
        right += 1
    bw = angles_deg[right] - angles_deg[left]
    # sidelobe level: max in the regions outside [left, right]
    sidelobes = np.concatenate((af_db[:left], af_db[right:]))
    sll = -np.max(sidelobes)  # positive dB
    results.append({
        "Taper": name,
        "Beamwidth (°)": round(bw, 3),
        "SLL (dB)": round(sll, 3),
    })
# 5) Plot all four in one figure
plt.figure(figsize=(8, 4))
for name, af_db in af_curves.items():
    plt.plot(angles_deg, af_db, label=name)
plt.title("Array Factor Comparison")
plt.xlabel("Angle (°)")
plt.ylabel("Normalized AF (dB)")
plt.ylim(-60, 0)
plt.xlim(0, 180)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
# 6) Print the results table
df = pd.DataFrame(results)
```

```
print(df.to_string(index=False))
```