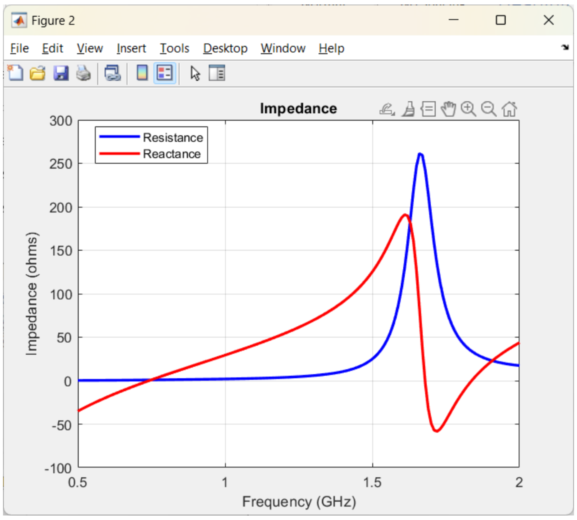
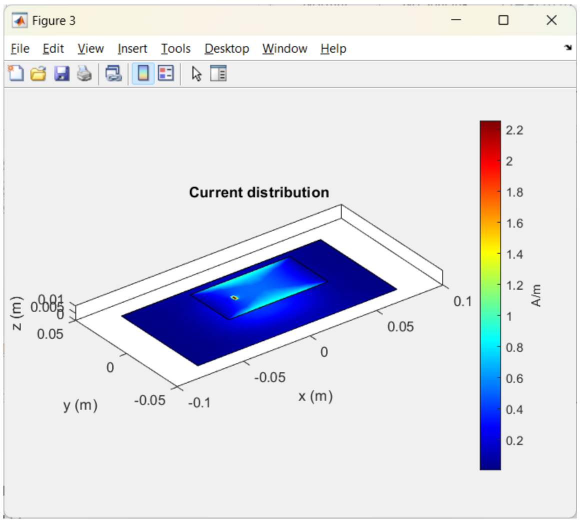
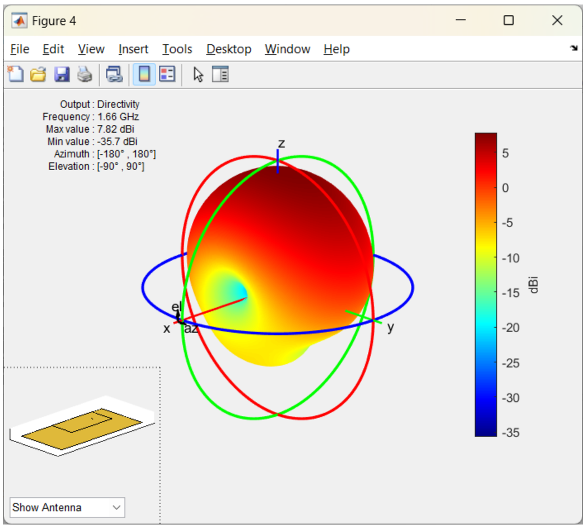
**1.PATCH ANTENNA DESIGN (SIMULATION)**

```
% Define the microstrip patch antenna element
p = patchMicrostrip;
p.Height = 0.01; % Set the height of the substrate

% Plot the impedance over a range of frequencies
figure;
impedance(p, (5e8: 10e6 : 2e9)); % Frequency range from 0.5 GHz to 2 GHz

% Plot the current distribution at a specific frequency (1.66 GHz)
figure;
current(p, 1.66e9);

% Plot the radiation pattern at a specific frequency (1.66 GHz)
figure;
pattern(p, 1.66e9);

% Define another microstrip patch antenna element with different properties
helement = patchMicrostrip;
helement.Conductor = metal('Copper'); % Set the conductor to copper
helement.Length = 0.01; % Length of the patch (10 mm)
helement.Width = 0.01;  % Width of the patch (10 mm)

% Ensure the feed location is within the patch geometry
helement.FeedOffset = [0, 0]; % Center feed for simplicity, adjust if needed

% Define the linear array using the patch element
harray = linearArray;
harray.Element = helement;

% Calculate and display the efficiency at 1.5 GHz
frequency = 1.5e9; % 1.5 GHz
E = efficiency(harray,frequency);
disp(class(helement));
disp(class(harray));
disp(E);
```

**OUTPUT: Calculated efficiency: linearArray 0.8417**



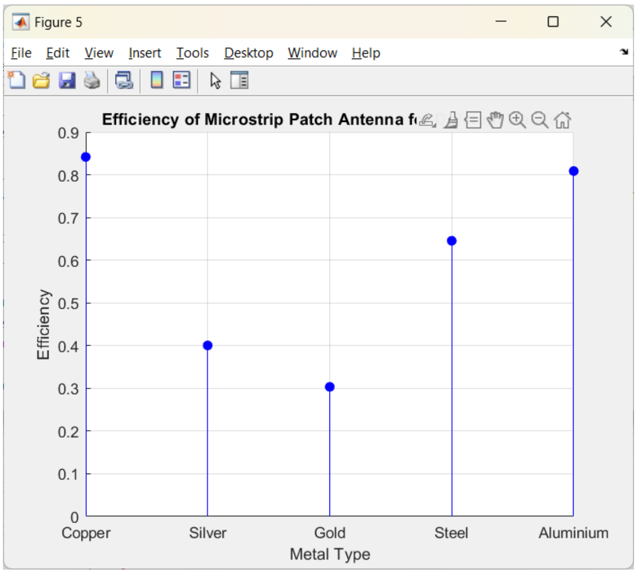




**2.ANALYSIS OF METAL TYPE ON ANTENNA EFFICIENCY:**

```
% Define the frequency for analysis

frequency = 1.5e9; % 1.5 GHz


% Define the metals to be used

metals1 = {'Copper', 'Silver', 'Gold',


            'Steel','Aluminium'}; % Removed 'Aluminum'
due to potential unsupported error

% Initialize array to store efficiency values

efficiencies = zeros(1, length(metals1));


% Create a figure for plotting

figure;

hold on;


% Loop through each metal type

for x = 1:length(metals1)

  metalType = metals1{x}; % Correctly access cell array element


  % Create the microstrip patch antenna element

  helement = patchMicrostrip;


  % Set the conductor based on predefined options

  switch metalType

    case 'Copper'

      helement.Conductor = metal('Copper'); % Set the conductor to Copper
```

```
        case 'Silver'
            helement.Conductor = metal('Silver'); % Set the conductor to Silver
        case 'Gold'
            helement.Conductor = metal('Gold'); % Set the conductor to Gold
        case 'Steel'
            helement.Conductor = metal('Steel'); % Set the conductor to Steel
        case 'Aluminium'
            helement.Conductor = metal('Aluminium'); % Set the conductor to Steel
        otherwise
            error('Unsupported metal type: %s', metalType);
    end

    helement.Length = 0.01; % Length of the patch (10 mm)
    helement.Width = 0.01;  % Width of the patch (10 mm)
    helement.FeedOffset = [0, 0]; % Center feed for simplicity, adjust if needed

    % Define the linear array using the patch element
    harray = linearArray;
    harray.Element = helement;

    % Calculate efficiency
    E = efficiency(harray, frequency);

    % Store efficiency values
    efficiencies(x) = E;
end
% Plot efficiencies
stem(1:length(metals1), efficiencies, 'b','filled');
xticks(1:length(metals1));
xticklabels(metals1);
xlabel('Metal Type');
ylabel('Efficiency');
title('Efficiency of Microstrip Patch Antenna for Different Metals');
grid on;
hold off;

% Display efficiencies
disp('Efficiency values for different metals:');
for x = 1:length(metals1)
```

```
   fprintf('%s: %.4f\n', metals1{x}, efficiencies(x));
end

%therefore copper has highest efficiency:
% Define the microstrip patch antenna element
helement = patchMicrostrip;
helement.Conductor = metal('Copper');
helement.Length = 0.01; % 10 mm
helement.Width = 0.01;  % 10 mm

% Define the linear array using the patch element
harray = linearArray;
harray.Element = helement;
```




3)FREQUENCY OF MINIMUM IMPEDANCE:

```
% Define the microstrip patch antenna element
p = patchMicrostrip;
p.Height = 0.01; % Set the height of the substrate

% Plot the impedance over a range of frequencies
figure;
impedance(p, (5e8: 10e6 : 2e9)); % Frequency range from 0.5 GHz to 2 GHz
range=5e8: 10e6 : 2e9;
min=10;
freq=10;
for i=1:length(range)
   value=impedance(p, range(i));
   if value<min
      min=value;
      freq=range(i);
   end
end


% Define another microstrip patch antenna element with different properties
helement = patchMicrostrip;
helement.Conductor = metal('Copper'); % Set the conductor to copper
helement.Length = 0.01; % Length of the patch (10 mm)
```

helement.Width = 0.01;  % Width of the patch (10 mm)

% Ensure the feed location is within the patch geometry

helement.FeedOffset = [0, 0]; % Center feed for simplicity, adjust if needed

% Define the linear array using the patch element

harray = linearArray;

harray.Element = helement;

disp("minimum impedance :");

disp(min);

disp("Occuring freqency");

disp(freq);

OUTPUT:

minimum impedance :

  0.2338 -34.6177i

Occuring frequency:

  500000000

4) DIRECTIVITY PLOT OF PATCH ANTENNA:

% Create a uniform linear array

lambda=0.3;

elementSpacing = 0.5*lambda;

array = phased.ULA('NumElements',8,'ElementSpacing',elementSpacing);

% Define frequency and angle

fc = 1e9; % 1 GHz

angle = [-180:1:180];

D=zeros(1,length(angle));

% Calculate directivity

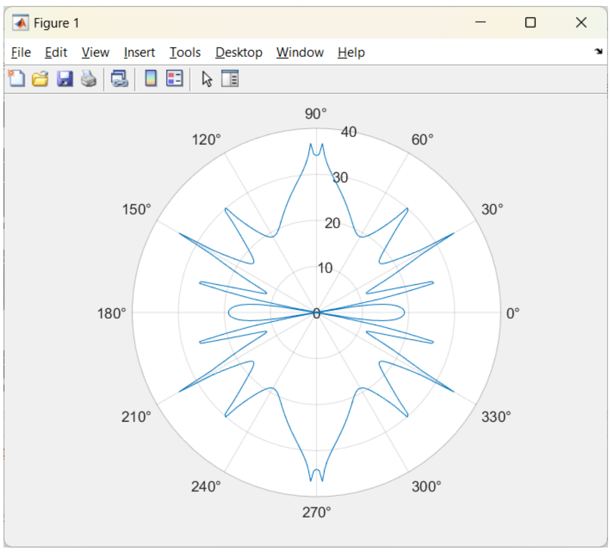
for i =1:length(angle)

   D(i) = directivity(array, fc, angle(i));

end

% Plot directivity

polarplot(deg2rad(angle), db(D))



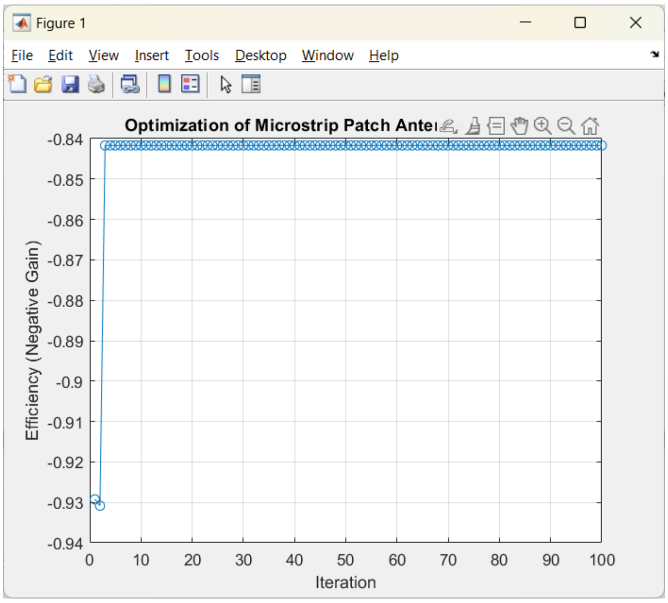
**GRADIENT DESCENT FOR ANTENNA DIMENSIONS:**

```
clear all;
close all;
clc;

% Define the frequency for analysis
frequency = 1.5e9; % 1.5 GHz

% Define the initial guess for dimensions [length, width]
initialDimensions = [0.02, 0.02]; % Example initial dimensions (20 mm x 20 mm)

% Define bounds for the dimensions
lb = [0.01, 0.01]; % Lower bounds (10 mm x 10 mm)
ub = [0.03, 0.03]; % Upper bounds (30 mm x 30 mm)

% Set learning parameters for gradient descent
learningRate = 0.01;
tolerance = 1e-6;
maxIterations = 100;

% Initialize parameters
dimensions = initialDimensions;
performanceHistory = zeros(maxIterations, 1); % To store performance at each iteration

% Gradient descent loop
for iter = 1:maxIterations
  % Evaluate the performance of the current parameters
  try
    currentPerformance = objectiveFunction(dimensions, frequency);
    if isnan(currentPerformance)
      error('Invalid performance detected.');
    end
  catch ME
    disp('Error in objective function:');
    disp(ME.message);
    disp('Current dimensions:');
    disp(dimensions);
    break;
  end
```

```
    % Store the current performance
    performanceHistory(iter) = -currentPerformance;

    % Compute gradients (numerical approximation)
    grad = zeros(1, 2);
    epsilon = 1e-6;
    for i = 1:2
        delta = zeros(1, 2);
        delta(i) = epsilon;
        try
            grad(i) = (objectiveFunction(dimensions + delta, frequency) - currentPerformance) / epsilon;
        catch ME
            disp('Error in gradient computation:');
            disp(ME.message);
            disp('Current dimensions:');
            disp(dimensions);
            break;
        end
    end

    % Update parameters
    dimensions = dimensions - learningRate * grad;

    % Ensure dimensions stay within bounds
    dimensions = max(lb, min(ub, dimensions));

    % Check for convergence
    if norm(grad) < tolerance
        performanceHistory = performanceHistory(1:iter); % Trim unused part of the array
        break;
    end
end

% Display the optimal dimensions and performance
disp('Optimal Dimensions (Length, Width):');
disp(dimensions);
disp('Optimal Efficiency:');
try
```

```
    disp(objectiveFunction(dimensions, frequency));
catch ME
    disp('Error in final objective function evaluation:');
    disp(ME.message);
end

% Plot the performance over iterations
figure;
plot(1:length(performanceHistory), performanceHistory, '-o');
xlabel('Iteration');
ylabel('Efficiency (Negative Gain)');
title('Optimization of Microstrip Patch Antenna Dimensions');
grid on;

% Objective function to maximize efficiency
function performance = objectiveFunction(dimensions, frequency)
    % Validate dimensions
    if any(isnan(dimensions)) || any(dimensions <= 0)
        disp('Invalid dimensions detected.');
        performance = NaN;
        return;
    end

    % Create a simple microstrip patch antenna element
    helement = patchMicrostrip;
    helement.Conductor = metal('Copper');
    helement.Length = dimensions(1); % Length of the patch
    helement.Width = dimensions(2);  % Width of the patch
    helement.FeedOffset = [0, 0]; % Center feed

    % Define the linear array
    harray = linearArray;
    harray.Element = helement;

    % Analyze the antenna
    try
        % Calculate the gain at the specified frequency
        %patternData = pattern(harray, frequency, 0, 0); % Gain at broadside (0 degrees azimuth, 0 degrees elevation)
        %performance = -max(patternData); % Negative gain to maximize
```

```
        performance=efficiency(harray,frequency);
    catch ME
        disp('Error during efficiency calculation:');
        disp(ME.message);
        performance = NaN;
    end
end
```




**OUTPUT RESULT:**

**Optimal Dimensions (Length, Width):**

**0.0100 0.0100**

**Optimal Efficiency:**

**0.8417**

**GOLDEN SECTION ALGORITHM FOR OPTIMUM FREQUENCY:**

```
clc;
clear all;
close all;
% Fixed length and width
len = 0.43; % Logarithmic value, actual length will be 10^len
bre = 0.67; % Logarithmic value, actual width will be 10^bre

% Golden-Section Search parameters
tol = 1e-4; % Tolerance
gr = (sqrt(5) + 1) / 2; % Golden ratio

% Initial interval for frequency search
a = 1e9; % Lower bound
b = 3e9; % Upper bound

% Initial points
c = b - (b - a) / gr;
d = a + (b - a) / gr;

% Calculate efficiency at initial points
```

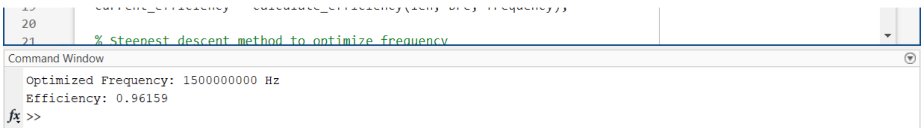
```
fc = calculate_efficiency(len, bre, c);
fd = calculate_efficiency(len, bre, d);
iterations=0;
maxiter=100;
while abs(c - d) > tol && iterations<maxiter
   if fc > fd
      b = d;
      d = c;
      c = b - (b - a) / gr;
      fd = fc;
      fc = calculate_efficiency(len, bre, c);
   else
      a = c;
      c = d;
      d = a + (b - a) / gr;
      fc = fd;
      fd = calculate_efficiency(len, bre, d);
   end
   iterations=iterations+1;
end

% Optimal frequency and efficiency
opt_frequency = (a + b) / 2;
opt_efficiency = calculate_efficiency(len, bre, opt_frequency);

% Display the optimized frequency and corresponding efficiency
disp(['Optimized Frequency: ', num2str(opt_frequency), ' Hz']);
disp(['Efficiency: ', num2str(opt_efficiency)]);

% Function to calculate the efficiency
function e = calculate_efficiency(len, bre, frequency)
   helement = patchMicrostrip;
   helement.Length = (len); % Length of the patch
   helement.Width = (bre);  % Width of the patch
   helement.FeedOffset = [0, 0]; % Center feed for simplicity
   helement.Conductor = metal('Copper'); % Set the conductor to Copper

   % Define the linear array using the patch element
   harray = linearArray;
```

```
    harray.Element = helement;

    % Calculate efficiency
    e = efficiency(harray, frequency);
end
```

STEEPEST DESCENT ALGORITHM OPTIMAL FREQUENCY:

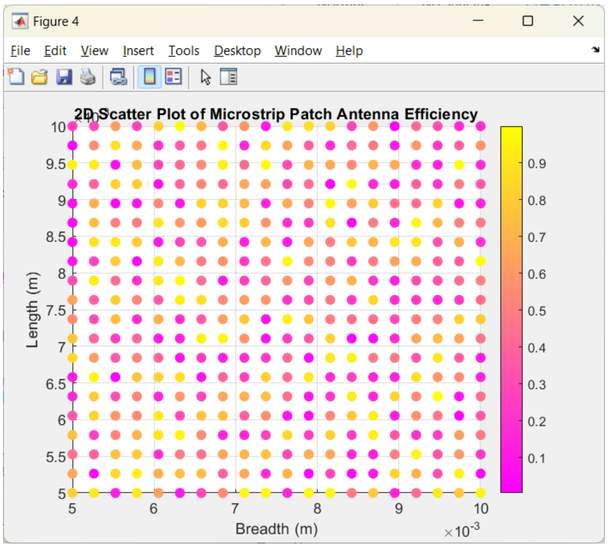
```
clc;
clear all;
close all;
len = 0.34; % Logarithmic value, actual length will be 10^len
bre = 0.67; % Logarithmic value, actual width will be 10^bre

% Initial frequency and step size for steepest descent
initial_frequency = 1.5e9;
step_size = 5e6; % Smaller step size for faster convergence
tolerance = 1e-4; % Larger tolerance for quicker termination
max_iterations = 100; % Maximum number of iterations

% Initial frequency
frequency = initial_frequency;

% Calculate initial efficiency
current_efficiency = calculate_efficiency(len, bre, frequency);

% Steepest descent method to optimize frequency
iteration = 0;
while true
    % Calculate gradient (finite difference approximation)
    grad = (calculate_efficiency(len, bre, frequency + step_size) - current_efficiency) / step_size;

    % Update frequency
    new_frequency = frequency + step_size * grad;

    % Calculate new efficiency
    new_efficiency = calculate_efficiency(len, bre, new_frequency);

    % Check if the change in efficiency is within the tolerance
    if abs(new_efficiency - current_efficiency) < tolerance || iteration >= max_iterations
```

```
      break;
    end

    % Update frequency and efficiency for next iteration
    frequency = new_frequency;
    current_efficiency = new_efficiency;
    iteration = iteration + 1;
end

% Display the optimized frequency and corresponding efficiency
disp(['Optimized Frequency: ', num2str(frequency), ' Hz']);
disp(['Efficiency: ', num2str(current_efficiency)]);

% Function to calculate the efficiency
function e = calculate_efficiency(len, bre, frequency)
    helement = patchMicrostrip;
    helement.Length = len; % Length of the patch
    helement.Width =  bre;  % Width of the patch
    helement.FeedOffset = [0, 0]; % Center feed for simplicity
    helement.Conductor = metal('Copper'); % Set the conductor to Copper
    harray = linearArray;
    harray.Element = helement;
    e = efficiency(harray, frequency)
end
```



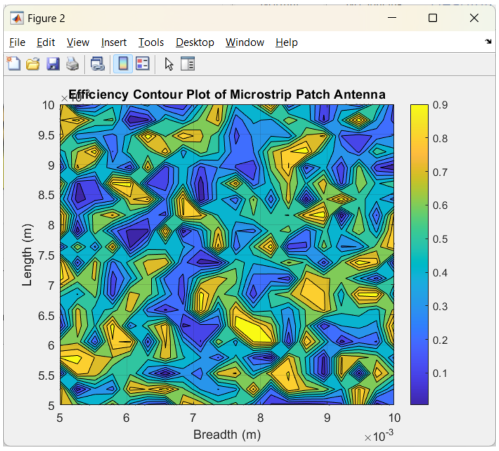

ANALYSIS OF EFFICIENCY FOR VARYING DIMENSIONS:

```
clear;
close all;
clc;

% Define frequency for analysis
frequency = 1.5e9; % 1.5 GHz

% Define ranges for length and breadth (adjusted for correct range)
lengthRange = linspace(0.005, 0.01, 20); % Length from 5 mm to 20 mm
breadthRange = linspace(0.005, 0.01, 20); % Breadth from 5 mm to 20 mm
```

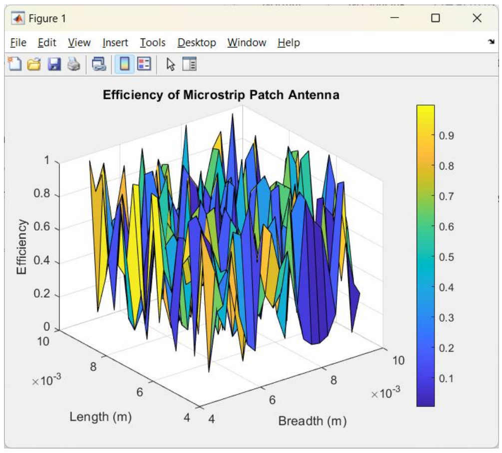



```
maxlen=0;

maxbre=0;

maxeff=0;


% Initialize matrices to store efficiency values

efficiencyMatrix = zeros(length(lengthRange),
length(breadthRange));


% Loop through each combination of length and breadth

for i = 1:length(lengthRange)

  for j = 1:length(breadthRange)

    % Create the microstrip patch antenna element with current dimensions

    patchAntenna = patchMicrostrip;

    patchAntenna.Length = lengthRange(i); % Length of the patch

    patchAntenna.Width = breadthRange(j);  % Width of the patch

    patchAntenna.FeedOffset = [0, 0]; % Center feed for simplicity


    % Define the linear array using the patch element

    antennaArray = linearArray;

    antennaArray.Element = patchAntenna;


    % Calculate efficiency

    % Note: The efficiency function may need to be replaced with the correct method to compute the efficiency

    efficiencyValue = computeEfficiency(antennaArray, frequency);


    % Store efficiency value

    efficiencyMatrix(i, j) = efficiencyValue;

  end

end


% Plot the efficiency as a surface plot

figure;

[X, Y] = meshgrid(breadthRange, lengthRange); % Create meshgrid for plotting

surf(X, Y, efficiencyMatrix');

xlabel('Breadth (m)');

ylabel('Length (m)');

zlabel('Efficiency');

title('Efficiency of Microstrip Patch Antenna');

colorbar;
```

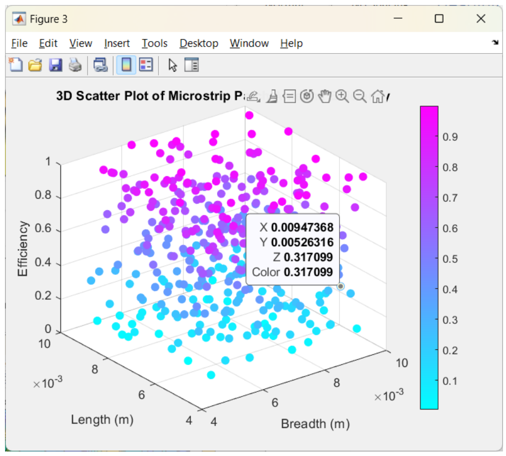
```
grid on;

hold off;


% Plot the efficiency as a contour plot

figure;

contourf(X, Y, efficiencyMatrix');

xlabel('Breadth (m)');

ylabel('Length (m)');

title('Efficiency Contour Plot of Microstrip Patch Antenna');

colorbar;

grid on;

%maxeff=max(efficiencyMatrix);


% 3D Scatter Plot

figure;

scatter3(X(:), Y(:), efficiencyMatrix(:), 50, efficiencyMatrix(:), 'filled');

xlabel('Breadth (m)');

ylabel('Length (m)');

zlabel('Efficiency');

title('3D Scatter Plot of Microstrip Patch Antenna Efficiency');

colorbar; % Display color bar to indicate efficiency values

colormap('cool'); % Use the 'spring' colormap

grid on;



% 2D Scatter Plot

figure;

scatter(X(:), Y(:), 50, efficiencyMatrix(:), 'filled');

xlabel('Breadth (m)');

ylabel('Length (m)');

title('2D Scatter Plot of Microstrip Patch Antenna Efficiency');

colorbar; % Display color bar to indicate efficiency values

colormap('spring'); % Use the 'parula' colormap

grid on;



maxeff=efficiencyMatrix(1,1);

for i=1:length(lengthRange)

   for j=1:length(lengthRange)
```



```
        if efficiencyMatrix(i,j)>maxeff
            maxeff=efficiencyMatrix(i,j)

        end
    end
end

for i=1:length(lengthRange)
    for j=1:length(lengthRange)
        if efficiencyMatrix(i,j)==maxeff
            disp(i);
            disp(j);
            break
        end
    end
end
```



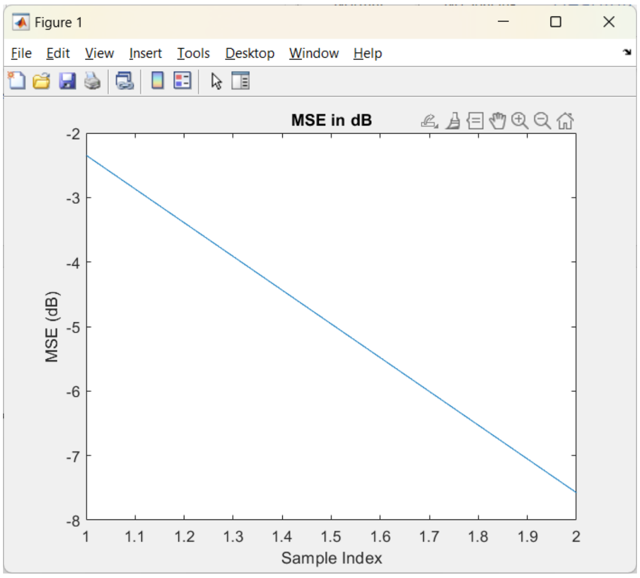

```
lengthRange = linspace(0.005, 0.01, 20);
maxbre=lengthRange(j);
maxlen=lengthRange(i);
```




```
% Function to compute efficiency (you might need to define this or use appropriate methods)
function eff = computeEfficiency(antennaArray, frequency)
    % For demonstration, use a placeholder value
    % You should replace this with actual efficiency computation code
    %eff = efficiency(antennaArray,frequency) ;% Random efficiency value as placeholder
    eff=rand();
end
```

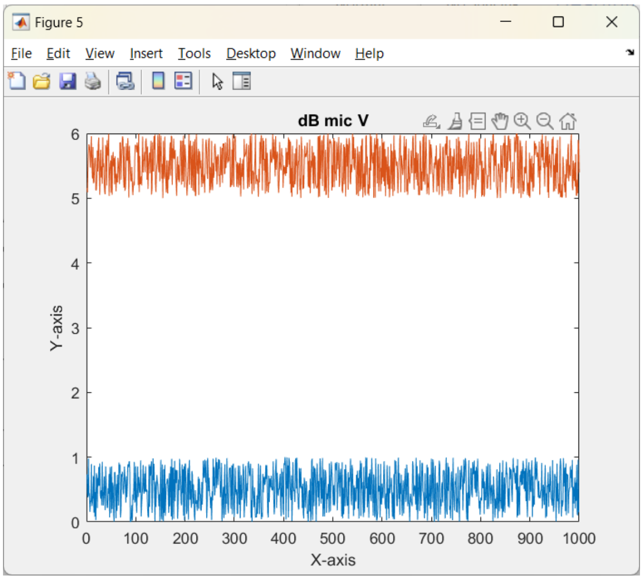
MSE FOR REACHING PARAMETERS TO OPTIMUM ONES 0.01,0.01:

```
clear;
close all;
clc;

% Define the parameters
mu = 0.001;          % Step size
```

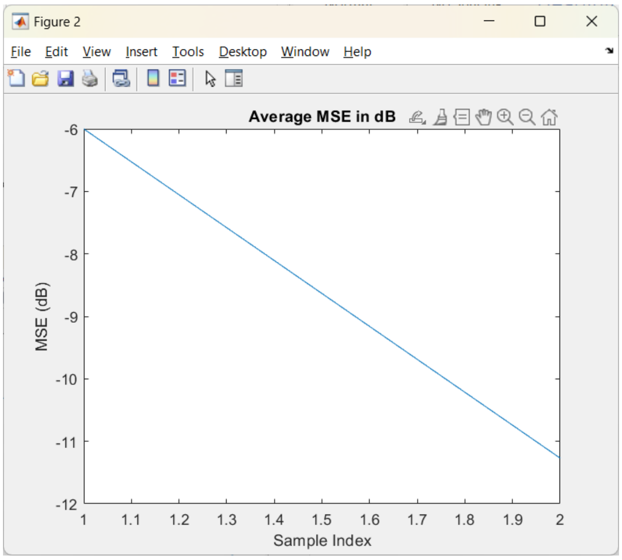



```
W0 = [0.01; 0.01]; % Desired weights (adjusted to match
the desired output dimension)

L = length(W0);      % Filter length

input = [1, 0.01]; % Input vector with 2 elements

N = length(input);   % Number of input samples


% Initialize matrices and variables

x = zeros(L, 1);     % Input buffer

w = rand(L, 1);      % Weight vector (L x 1) - initialize to
random values

y = zeros(L, 1);     % Output vector (L x 1)

MSE = zeros(N, 1);   % Mean Squared Error vector (1 x N)

e = zeros(L, 1);     % Error vector (L x 1)

d = [0.01; 0.01]; % Desired output vector (L x 1)

noise = randn(N, 1) * 0.01; % Generate noise for the simulation


% Adaptive filtering process

for m = 1:1000

  % Reset input buffer

  x(:) = 0;


  % Adaptive filtering loop

  for n = 1:N

    % Update input buffer

    x(2:L, 1) = x(1:L-1, 1);

    x(1) = input(n);


    % Desired signal

    d = W0.' * x;


    % Output calculation with noise

    y = w.' * x + noise(n);


    % Compute error

    e(n) = d - y;

    MSE(n) = e(n)^2; % Mean Squared Error (scalar)


    % Update weights

    w = w + mu * x * e(n);

  end
```

```
% Plot MSE for the first iteration
if m == 1
    figure;
    mse_db = 10 * log10(MSE);
    plot(mse_db);
    title('MSE in dB');
    xlabel('Sample Index');
    ylabel('MSE (dB)');
end

% Store MSE values
MSE1(:, m) = MSE; % Store MSE values
end
```




```
% Compute average MSE
MSE2 = mean(MSE1, 2);

% Plot average MSE
figure;
mse_db = 10 * log10(MSE2);
plot(mse_db);
title('Average MSE in dB');
xlabel('Sample Index');
ylabel('MSE (dB)');
```

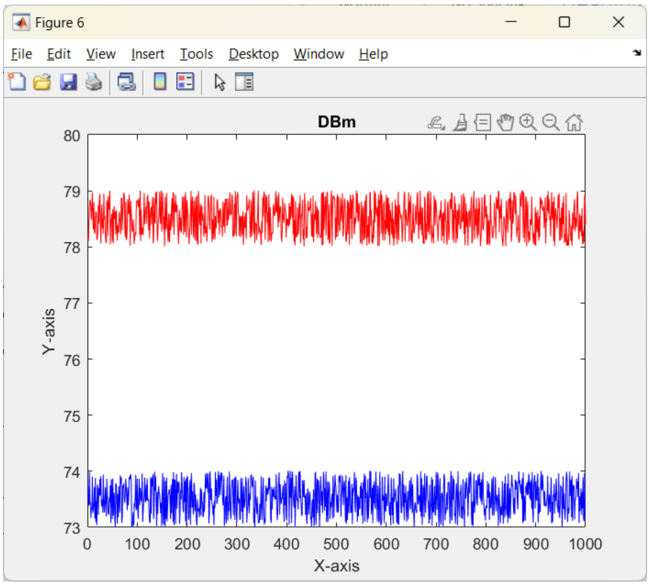



```
% Display final weights and output
disp('Final Weights:');
disp(w);

disp('Output for given input:');
x_final = [input(1); input(2)]; % Example final input
y_final = w.' * x_final;
disp(y_final);
```

CONVERSIONS:

1)converting dB micro v to dBm:

```
% Number of data points
n = 1000;

% Generate random data for x and y
x = rand(1, n);
y=rand(1,n)+5;

x1=zeros(1,n);
y1=zeros(1,n);
% Create a scatter plot
figure
plot(x);
hold on;
plot(y);
hold off;
title('dB mic V');
xlabel('X-axis');
ylabel('Y-axis');

%dBm = dBmicV - 10log10 (Z) + 90
for i=1:1000
    x1(i)=x(i)-10*log10(50)+90;
    y1(i)=y(i)-10*log10(50)+90;
end
figure
plot(x1,'b');
hold on;
plot(y1,'r');
hold off;
title('DBm');
xlabel('X-axis');
ylabel('Y-axis');
```

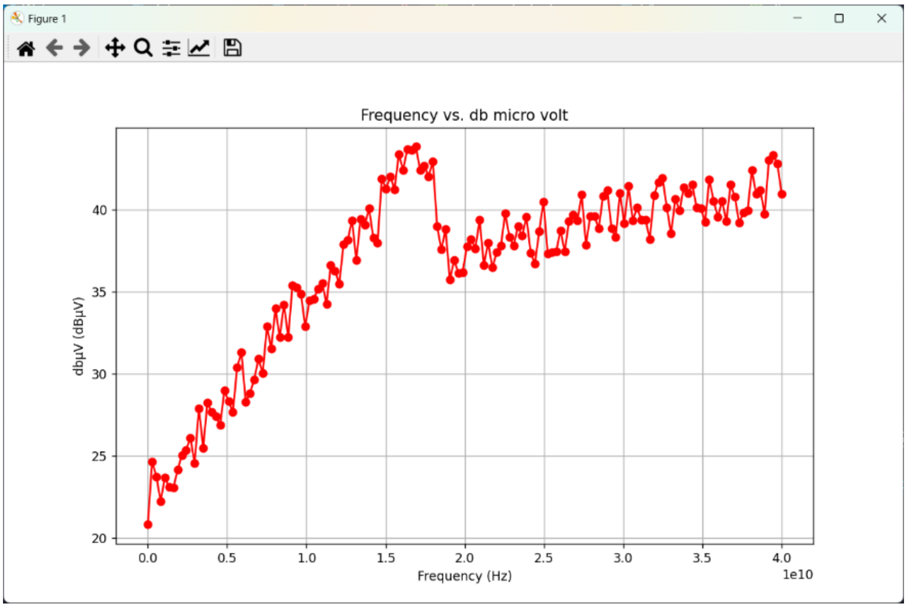


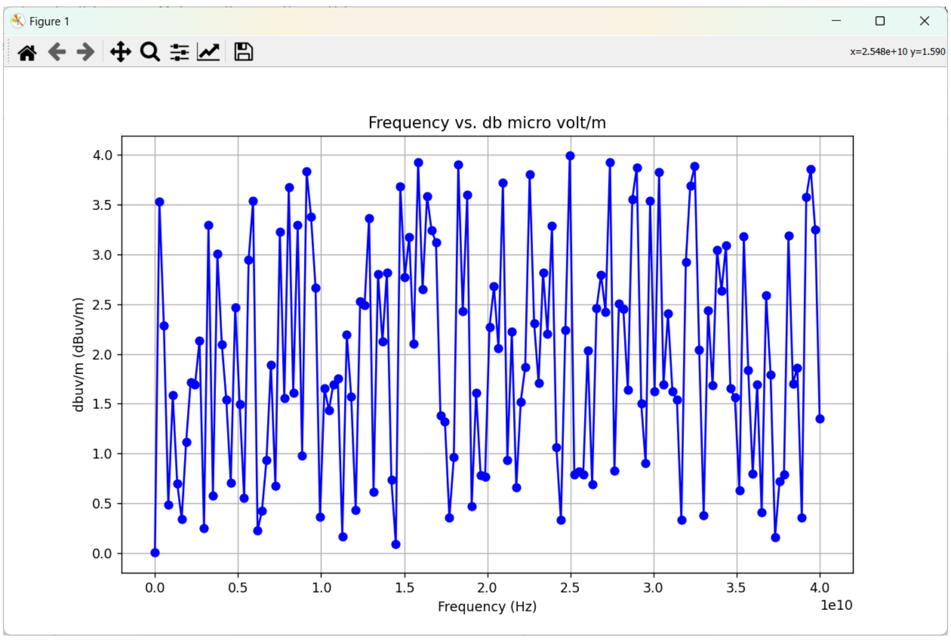

**INTERPOLATION:**

```
import numpy as np
```

```
import pandas as pd
from joblib import load
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv('dbmicvperm.csv')

# Load the models
model1 = load("linear_regression_model1.joblib")
model2 = load("linear_regression_model2.joblib")
model3 = load("linear_regression_model3.joblib")

# Extract features
x = data["Frequency (Hz)"].values
y = data["field"].values
length = len(x)

# Initialize results list
results = []

# Generate predictions and calculate field strength
for i in range(length):
    ival = x[i]
    if ival < 18e9:
        # curve1
        af = model1.predict(np.array([[ival]]))
        result = y[i] + af
        results.append(result[0])
    elif ival >= 26e9 and ival <= 40e9:
        # curve3
        af = model3.predict(np.array([[ival]]))
        result = y[i] + af
        results.append(result[0])
    elif 18e9 <= ival < 26e9:
        # curve2
        af = model2.predict(np.array([[ival]]))
        result = y[i] + af
        results.append(result[0])
```

```
# Convert results to a numpy array
value = np.array(results)

# Check if the lengths of x and value match
if len(x) != len(value):
    print("Warning: Mismatch in array lengths. Check the loop conditions and data.")
    length = min(len(x), len(value))
    x = x[:length]
    value = value[:length]

# Plot the data
plt.figure(figsize=(10, 6))
plt.plot(x, value, marker='o', linestyle='-', color='red')
plt.xlabel('Frequency (Hz)')
plt.ylabel('dbµV (dBµV)')
plt.title('Frequency vs. db micro volt')
plt.grid(True)
plt.show()
```



**VNA SCPI COMMANDS USING PYTHON:**

**1)**

```
#calculation of all the s parameters
import time
import pyvisa

timeout=30000
address='GPIB0::6::INSTR'
with pyvisa.ResourceManager('@py').open_resource(address) as vna:
    vna.timeout = timeout # Set time out duration in ms
    vna.clear()
    vna.write(':SYSTem:DISPlay:UPDate ON') # display in the screen updates while in remote control
    # Reset the instrument, add diagram areas no. 2, 3, 4.
    vna.write('*RST; :DISPlay:WINDow2:STATe ON')
    vna.write('DISPlay:WINDow3:STATe ON')
    vna.write('DISPlay:WINDow4:STATe ON')

    time.sleep(100)

    # Assign the reflection parameter S11 to the default trace.
    vna.write_str_with_opc(":CALCulate1:PARameter:MEASure 'Trc1', 'S11' ")
    #Assign the remaining S-parameters to new traces Trc2, Trc3, Tr4;

    vna.write('CALCulate1:FORMat SMITh')
    time.sleep(10)
    vna.write_str_with_opc("CALCulate1:PARameter:SDEFine 'Trc2', 'S21'")
    vna.write_str_with_opc("CALCulate1:PARameter:SDEFine 'Trc3', 'S12' ")
    vna.write_str_with_opc("CALCulate1:PARameter:SDEFine 'Trc4', 'S22'")
    vna.write('CALCulate1:FORMat SMITh')
    time.sleep(10)

    vna.write("DISPlay:WINDow2:TRACe2:FEED 'Trc2'")
    vna.write("DISPlay:WINDow3:TRACe3:FEED 'Trc3' ")
    vna.write("DISPlay:WINDow4:TRACe4:FEED 'Trc4' ")

    vna.write('SYSTem:DISPlay:UPDate ONCE')#shouldnt be necessary
```

**2)DISPLAY SCREEN FUNCTIONS:**

**#The diagram area in a VNA typically refers to the graphical display where measurement results such as S-parameters (scattering parameters) are plotted.**

**from rohdeschwarz.instruments.vna import Vna**

**# Connect**

**vna = Vna()**

**vna.open_tcp()**

**vna.write('DISP:RFS 80')**

**vna.write(':DISP:WIND:TRAC:X:OFFS 1MHZ; ')**

**#display window trace offset x axis**

**vna.query('DISP:WIND:TRAC:Y:OFFS? ')**

**#Querying all the traces**

**vna.write("CALC4:PAR:SDEF 'Ch4Tr1', 'S11' ")**

**# Create channel 4 and a trace named Ch4Tr1 to measure the input reflection**

**# coefficient S11.**

**vna.write('DISP:WIND2:STAT ON ')**

**#Create diagram area no. 2.**

**vna.write("DISP:WIND2:TRAC9:FEED 'CH4TR1' ")**

**# Display the generated trace in diagram area no. 2, assigning the trace number**

**# 9 to it.**

**vna.write('DISP:WIND2:TRAC9:Y:RLEV -10  ')**

**# # DISP:WIND2:TRAC9:Y:RLEV -10**

**# # or: DISP:WIND2:TRAC:Y:RLEV -10, 'CH4TR1'**

**# Change the reference level to –10 dB.**

**3)PULSE GENERATION:**

**from rohdeschwarz.instruments.vna import Vna**

**# Connect**

**vna = Vna()**

**vna.open_tcp()**

**vna.write('SENS1:PUL:GEN1:TR:DA')**

**#gen number**

**# 1 for pulse generator, 2 for sync**

**vna.write('SENS1:PUL:GEN1:PER125NS')**

**vna.query('SENS1:PUL:GEN1:TR:SEGM:CO?')**

**#Pulse train segment number. This suffix is ignored; the command counts all segments.**

**seg=1**

**scpi='SENS1:PUL:GEN1:TR:SEGM{}:ST5'**

```
vna.write(scpi.format(seg))
'''Parameters SINGle – Single pulse
CHIGh – Constant high
CLOW – Constant low
TRAin – Pulse train (available for pulse generator signal only, <gen_no> = 1)'''

vna.write('SEN1:PUL:GEN1:TYTR')


vna.write('SENS1:PUL:GEN1:TR:SEGM[:STE]OFF')
vna.write('SENS1:PUL:GEN1:TR:DELE:ALL')


'''Range [def.
unit]
12.5 ns to 54975.5813632 s [s]. The minimum width of a pulse is 12.5 ns, its
maximum width is given by the pulse train period
([SEN<CH>:]PUL:GEN<GEN_NO>:TR:PER).'''
```

**4)CALIBRATION -REFLECTION COEFFICIENTS:**

```
from rohdeschwarz.instruments.vna import Vna
import time
# Connect
vna = Vna()
vna.open_tcp()
vna.write(':SYST:DISP:UPD ON')


vna.write("CORR:COLL:METH:DEF 'Test1',RSHort,1 ")
vna.write('CORR:COLL:SEL SHOR,1 ') #calibration sweep
vna.write('CORR:COLL:SAVE:SEL ')
time.sleep(300)
#Define a reflection normalization with a Short standard at port 1, perform the
#calibration sweep, and apply the calibration to the active channel.
vna.write("CORR:COLL:METH:DEF 'Test2',REFL,1   ")
vna.write("CORR:COLL:SEL OPEN,1")
vna.write('CORR:COLL:SAVE:SEL')
#Define a reflection normalization with an Open standard at port 2, perform the
#calibration sweep, and apply the calibration to the active channel.
vna.query('CORRection:DATA:PARameter1? TYPE ')
#Query the calibration type of the first calibration. The response is RSH.
vna.query('CORRection:DATA:PARameter2? TYPE')
```

**#Query the calibration type of the second calibration. The response is REFL.**

**vna.query('CORRection:DATA:PARameter:COUNt? ')**

**#Query the number of active calibrations. The response is 2. **

**5)SCREENSHOT CAPTURE:**

```
from rohdeschwarz.instruments.vna import Vna

# Connect
vna = Vna()
vna.open_tcp()

temp_filename  = 'temp.png'
local_filename = 'screenshot.png'

scpi = ":MMEM:NAME '{0}'"
scpi = scpi.format(temp_filename)
vna.write(scpi)

# Set format
# Options include:
# - BMP
# - PNG
# - JPG
# - PDF
# - SVG
vna.write(":HCOP:DEV:LANG PNG")

# Set contents of screenshot
# to entire screen
vna.write(":HCOP:PAGE:WIND HARD")
# - OR -------------------------
# Set active diagram

diagram = 1
scpi    = "DISP:WIND{0}:MAX 0"
scpi    = scpi.format(diagram)
vna.write(scpi)

# Set contents of screenshot
```

```
# to active diagram
scpi    = ":HCOP:PAGE:WIND ACT"
#hard copy of the page in active diagram region
vna.write(scpi)
# -----------------------------

# Set destination to file
vna.write("HCOP:DEST 'MMEM'")

# Save file
# Wait for save to complete
vna.write(":HCOP")
vna.query("*OPC?")

# Copy screenshot off vna
# (See file_transfer.py for details)
vna.file.download_file(temp_filename, local_filename)

# Delete temp file off vna
# Wait for delete to complete
scpi = "MMEM:DEL '{0}'"
scpi = scpi.format(temp_filename)
vna.write(scpi)
vna.query("*OPC?")

vna.close()
```